NIST Special Publication 500-207

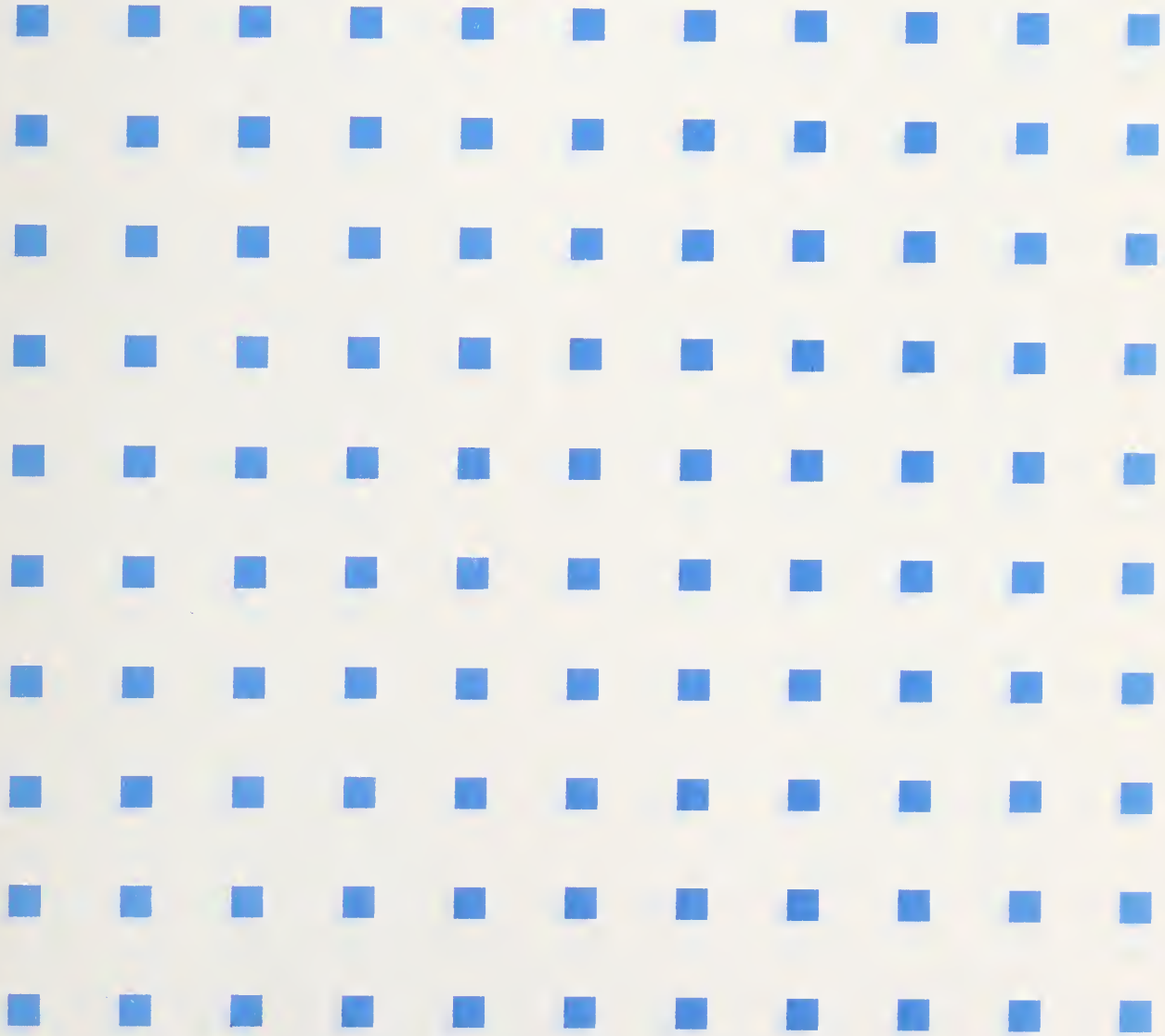
Computer Systems Technology

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards and Technology

NIST

# The First Text REtrieval Conference (TREC-1)

D. K. Harman



The National Institute of Standards and Technology was established in 1988 by Congress to "assist industry in the development of technology . . . needed to improve product quality, to modernize manufacturing processes, to ensure product reliability . . . and to facilitate rapid commercialization . . . of products based on new scientific discoveries."

NIST, originally founded as the National Bureau of Standards in 1901, works to strengthen U.S. industry's competitiveness; advance science and engineering; and improve public health, safety, and the environment. One of the agency's basic functions is to develop, maintain, and retain custody of the national standards of measurement, and provide the means and methods for comparing standards used in science, engineering, manufacturing, commerce, industry, and education with the standards adopted or recognized by the Federal Government.

As an agency of the U.S. Commerce Department's Technology Administration, NIST conducts basic and applied research in the physical sciences and engineering and performs related services. The Institute does generic and precompetitive work on new and advanced technologies. NIST's research facilities are located at Gaithersburg, MD 20899, and at Boulder, CO 80303. Major technical operating units and their principal activities are listed below. For more information contact the Public Inquiries Desk, 301-975-3058.

## Technology Services
- Manufacturing Technology Centers Program
- Standards Services
- Technology Commercialization
- Measurement Services
- Technology Evaluation and Assessment
- Information Services

## Electronics and Electrical Engineering Laboratory
- Microelectronics
- Law Enforcement Standards
- Electricity
- Semiconductor Electronics
- Electromagnetic Fields[1]
- Electromagnetic Technology[1]

## Chemical Science and Technology Laboratory
- Biotechnology
- Chemical Engineering[1]
- Chemical Kinetics and Thermodynamics
- Inorganic Analytical Research
- Organic Analytical Research
- Process Measurements
- Surface and Microanalysis Science
- Thermophysics[2]

## Physics Laboratory
- Electron and Optical Physics
- Atomic Physics
- Molecular Physics
- Radiometric Physics
- Quantum Metrology
- Ionizing Radiation
- Time and Frequency[1]
- Quantum Physics[1]

## Manufacturing Engineering Laboratory
- Precision Engineering
- Automated Production Technology
- Robot Systems
- Factory Automation
- Fabrication Technology

## Materials Science and Engineering Laboratory
- Intelligent Processing of Materials
- Ceramics
- Materials Reliability[1]
- Polymers
- Metallurgy
- Reactor Radiation

## Building and Fire Research Laboratory
- Structures
- Building Materials
- Building Environment
- Fire Science and Engineering
- Fire Measurement and Research

## Computer Systems Laboratory
- Information Systems Engineering
- Systems and Software Technology
- Computer Security
- Systems and Network Architecture
- Advanced Systems

## Computing and Applied Mathematics Laboratory
- Applied and Computational Mathematics[2]
- Statistical Engineering[2]
- Scientific Computing Environments[2]
- Computer Services[2]
- Computer Systems and Communications[2]
- Information Systems

[1] At Boulder, CO 80303.
[2] Some elements at Boulder, CO 80303.

NIST Special Publication 500-207

# The First Text REtrieval Conference (TREC-1)

D. K. Harman, Editor

Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

March 1993


**U.S. DEPARTMENT OF COMMERCE**
Ronald H. Brown, Secretary

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
Raymond G. Kammer, Acting Director

## Reports on Computer Systems Technology

The National Institute of Standards and Technology (NIST) has a unique responsibility for computer systems technology within the Federal government. NIST's Computer Systems Laboratory (CSL) develops standards and guidelines, provides technical assistance, and conducts research for computers and related telecommunications systems to achieve more effective utilization of Federal information technology resources. CSL's responsibilities include development of technical, management, physical, and administrative standards and guidelines for the cost-effective security and privacy of sensitive unclassified information processed in Federal computers. CSL assists agencies in developing security plans and in improving computer security awareness training. This Special Publication 500 series reports CSL research and guidelines to Federal agencies as well as to organizations in industry, government, and academia.

**National Institute of Standards and Technology Special Publication 500-207**
**Natl. Inst. Stand. Technol. Spec. Publ. 500-207, 513 pages (March 1993)**
**CODEN: NSPUE2**

**U.S. GOVERNMENT PRINTING OFFICE**
**WASHINGTON: 1993**

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402

## Preface

This report constitutes the proceedings of the first Text REtrieval Conference (TREC-1) held in Gaithersburg, Maryland, November 4-6, 1992. The conference was co-sponsored by the National Institute of Standards and Technology (NIST) and the Defense Advanced Research Projects Agency (DARPA), and was attended by 92 people involved in the 25 participating groups. The conference was the first in an on-going series of workshops to evaluate new technologies in text retrieval.

The workshop included plenary sessions and six discussion groups. Because the participants in the workshop drew on their personal experiences, they sometimes cited specific vendors and commercial products. The inclusion or omission of a particular company or product does not imply either endorsement or criticism by NIST.

The sponsorship of the Software and Intelligent Systems Technology Office of the Defense Advanced Research Projects Agency is gratefully acknowledged, along with the tremendous work of the program committee.

Donna Harman
February 20, 1993

TREC-1 Program Committee

Donna Harman, NIST, Chair
Ed Addison, ConQuest, Inc.
Chris Buckley, Cornell University
Darryl Howard, U.S. Department of Defense
David Lewis, AT&T Bell Labs
Jan Pedersen, Xerox Parc
John Prange, U.S. Department of Defense
Alan Smeaton, Dublin City University, Ireland
Richard Tong, Advanced Decision Systems

# TABLE OF CONTENTS

REPORTS OF DISCUSSION GROUPS



APPENDICES

## Abstract

This report constitutes the proceedings of the first Text REtrieval Conference (TREC-1) held in Gaithersburg, Maryland, November 4-6, 1992. The conference was co-sponsored by the National Institute of Standards and Technology (NIST) and the Defense Advanced Research Projects Agency (DARPA), and was attended by 92 people involved in the 25 participating groups.

The goal of the conference was to bring research groups together to discuss their work on a new large test collection. There was a large variety of retrieval techniques reported on, including methods using automatic thesauri, sophisticated term weighting, natural language techniques, relevance feedback, and advanced pattern matching. Because results had been run through a common evaluation package, groups were able to compare the effectiveness of different techniques, and discuss how differences among the systems affected performance.

The conference included paper sessions and six discussions groups. This proceedings includes papers from all participants, including the poster sessions, and papers from the panel, along with reports from some of the discussions groups.

# Overview of the First Text REtrieval Conference (TREC-1)

Donna Harman
National Institute of Standards and Technology
Gaithersburg, Md. 20899

## 1. Introduction

There is a long history of experimentation in information retrieval. Research started with experiments in indexing languages, such as the Cranfield I tests (Cleverdon 1962), and has continued with over 30 years of experimentation with the retrieval engines themselves. The Cranfield II studies (Cleverdon et al. 1966) showed that automatic indexing was comparable to manual indexing, and this and the availability of computers created a major interest in the automatic indexing and searching of texts. The Cranfield experiments also emphasized the importance of creating test collections and using these for comparative evaluation. The Cranfield collection, created in the late 1960's, contained 1400 documents and 225 queries, and has been heavily used by researchers since then. Subsequently other collections have been built, such as the CACM collection (Fox 1983), and the NPL collection (Sparck Jones & Webster 1979).

In the 30 or so years of experimentation there have been two missing elements. First, although some research groups have used the same collections, there has been no concerted effort by groups to work with the same data, use the same evaluation techniques, and generally compare results across systems. The importance of this is not to show any system to be superior, but to allow comparison across a very wide variety of techniques, much wider than only one research group would tackle. Karen Sparck Jones in 1981 commented that:

> Yet the most striking feature of the test history of the past two decades is its lack of consolidation. It is true that some very broad generalizations have been endorsed by successive tests: for example...but there has been a real failure at the detailed level to build one test on another. As a result there are no explanations for these generalizations, and hence no means of knowing whether improved systems could be designed (p. 245).

This consolidation is more likely if groups can compare results across the same data, using the same evaluation method, and then meet to discuss openly how methods differ.

The second missing element, which has become critical in the last 10 years, is the lack of a realistically-sized test collection. Evaluation using the small collections currently available may not reflect performance of systems in large full-text searching, and certainly does not demonstrate any proven abilities of these systems to operate in real-world information retrieval environments. This is a major barrier to the transfer of these laboratory systems into the commercial world. Additionally some techniques such as the use of phrases and the construction of automatic thesauri seem intuitively workable, but have repeatedly failed to show improvement in performance using the small collections. Larger collections might demonstrate the effectiveness of these procedures.

The overall goal of the Text REtrieval Conference (TREC) was to address these two missing elements. It is hoped that by providing a very large test collection, and encouraging interaction with other groups in a friendly evaluation forum, a new thrust in information retrieval will occur. There is also an increased interest in this field within the DARPA community, and TREC is designed to be a showcase of the state-of-the-art in retrieval research. NIST's goal as co-sponsor of TREC is to encourage communication and technology transfer among academia, industry, and government.

## 2. The Task

### 2.1 Introduction

TREC is designed to encourage research in information retrieval using large data collections. Two types of retrieval are being examined -- retrieval using an "adhoc" query such as a researcher might use in a library environment, and retrieval using a "routing" query such as a profile to filter some incoming document stream. The TREC task is not tied to any given application, and is not concerned with interfaces or optimized response time for searching. However it is helpful to have some potential user in mind when designing or testing a retrieval system. The model for a user in TREC is a dedicated searcher, not a novice searcher, and the model for the application is one needing monitoring of data streams for information on specific topics (routing), and the ability to do adhoc searches on archived data for new topics. It should be assumed that the users need the ability to do both high precision and high recall searches, and are willing to look at many documents and repeatedly modify queries in order to get high recall. Obviously they would like a system that makes this as easy as possible, but this ease should be reflected in TREC as added intelligence in the system rather than as special interfaces.

Since TREC has been designed to evaluate system performance both in a routing (filtering or profiling) mode, and in an adhoc mode, both functions need to be tested. The test design was based on traditional information retrieval models, and evaluation used traditional recall and precision measures. The following diagram of the test design shows the various components of TREC (fig. 1).

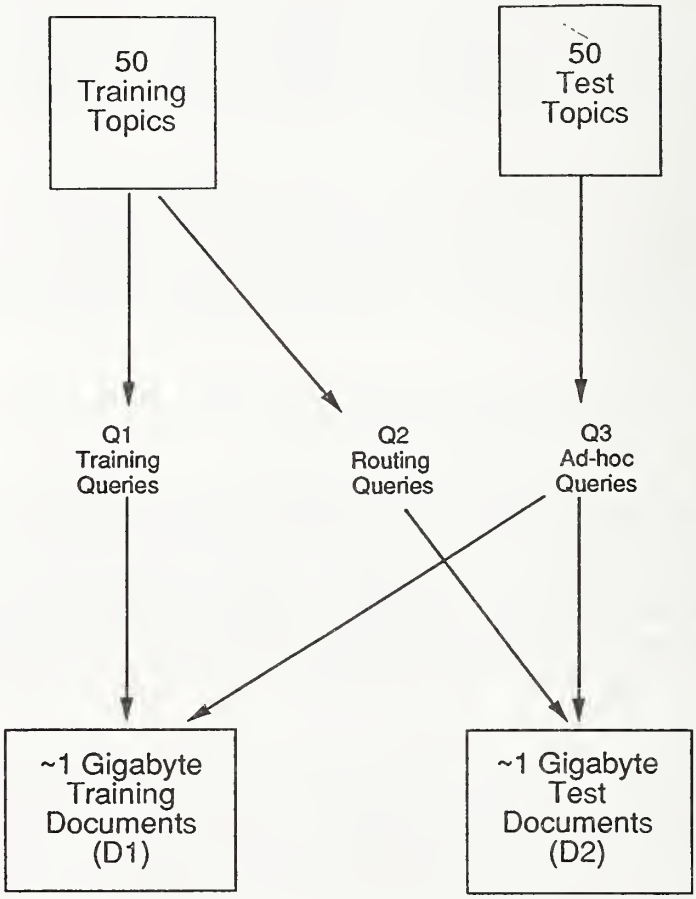



Figure 1. The TREC Task.

This diagram reflects the four data sets (2 sets of topics and 2 sets of documents) that were provided to participants. These data sets (along with a set of sample relevance judgments for the 50 training topics) were used to

construct three sets of queries. Q1 is the set of queries (probably multiple sets) created to help in adjusting a system to this task, to create better weighting algorithms, and in general to train the system for testing. The results of this research were used to create Q2, the routing queries to be used against the test documents. Q3 is the set of queries created from the test topics as adhoc queries for searching against the combined documents (both training documents and test documents). The results from searches using Q2 and Q3 were the official test results. The documents were full-length text from various sources such as newspapers, newswires, magazines and journals (see sect. 3.2 for more details).

### 2.2 Specific Task Guidelines

The various TREC participants used a wide variety of indexing/knowledge base building techniques, and a wide variety of approaches to generate search queries. Therefore it was important to establish clear guidelines for the TREC task and to develop some methods of standardized reporting to allow comparison. The guidelines deal with the methods of indexing/knowledge base construction, and with the methods of generating the queries from the supplied topics. In general they were constructed to reflect an actual operational environment, and to allow as fair as possible a separation among the diverse query construction approaches.

There were guidelines for constructing and manipulating the system data structures. These structures were defined to consist of the original documents, any new structures built automatically from the documents (such as inverted files, thesauri, conceptual networks, etc.) and any new structures built manually from the documents (such as thesauri, synonym lists, knowledge bases, rules, etc.). The following guidelines were provided to the participants.

1. System data structures can be built using the initial training set (documents D1, training topics, and relevance judgments). They may be modified based on the test documents D2, but not based on the test topics. In particular, the processing of one test topic should not affect the processing of another test topic. For example, it would not be allowed to update a system knowledge base based on the analysis of one test topic in such a way that the interpretation of subsequent test topics was changed in any fashion.

2. There are several parts of the Wall Street Journal and the Ziff material (see sect. 3.2) that contain manually assigned controlled or uncontrolled index terms. These fields are delimited by SGML tags, as specified in the documentation files included with the data. Other parts of the TREC data contain no manual indexing. Since the primary focus of TREC is on retrieval and routing of naturally occurring text, these manually indexed terms should not be indiscriminately used as if they are a normal part of the text. If your group decides to use these terms, they should be part of a specific experiment that utilizes manual indexing terms, and their use should be declared.

3. Special care should be used in handling the routing topics. In a true routing situation, a single document would be indexed and "passed" against the routing topics. Since most of you will be indexing the test documents as a complete set, routing should be simulated by not using any test document information (such as IDF based on the test collection, total frequency based on the test collection, etc.) in the searching. It is perfectly permissible to use training-set collection information however. If your system bases system data structures on the entire test data and is unable to operate in a proper routing mode, then you should either have a different method for handling routing, or only submit results for the adhoc part of TREC.

Additionally there were guidelines for constructing the queries from the provided topics (see sect. 3.3 for more on the topics). These guidelines were considered of great importance for fair system comparison and were therefore carefully constructed. Three generic categories were defined, based on the amount and kind of manual intervention used.

1. Method 1 -- completely automatic initial query construction.

   adhoc queries -- The system will automatically extract information from the topic (the topic fields used should be identified) to construct the query. The query will then be submitted to the system (with no manual modifications) and the results from the system will be the results submitted to NIST. There should be no manual intervention that would affect the results.

   routing queries -- The queries should be constructed automatically using the training topics, the training relevance judgments and the training documents. The queries should then be submitted to NIST before the

test documents are released and should not be modified after that point. The unmodified queries should be run against the test documents and the results submitted to NIST.

2. Method 2 -- manual initial query construction.

   adhoc queries -- The query is constructed in some manner from the topic, either manually or using machine assistance. The methods used should be identified, along with the human expertise (both domain expertise and computer expertise) needed to construct a query. Once the query has been constructed, it will be submitted to the system (with no manual intervention), and the results from the system will be the results submitted to NIST. There should be no manual intervention after initial query construction that would affect the results. (Manual intervention is covered by Method 3.)

   routing queries -- The queries should be constructed in the same manner as the adhoc queries for Method 2, but using the training topics, relevance judgments, and training documents. They should then be submitted to NIST before the test documents are released and should not be modified after that point. The unmodified queries should be run against the test documents and the results submitted to NIST.

3. Method 3 -- automatic or manual query construction with feedback.

   adhoc queries -- The initial query can be constructed using either Method 1 or Method 2. The particular technique used should be described. The query is submitted to the system, and a subset of the retrieved documents is used for manual feedback, i.e., a human makes judgments about the relevance of the documents in this subset. These judgments may be communicated to the system, which may automatically modify the query, or the human may simply choose to modify the query himself. In either case, the expertise of the person or persons examining the documents should be described, both their domain expertise and their experience in online searching, and the manner of system feedback (i.e., automatic system modification of query or human modification) should be also described. At some point, feedback should end, and the query should be accepted as final.

   Three sets of results should be sent to NIST for each topic. The first set should be the results without feedback, i.e., the top 200 documents retrieved from an initial query produced without feedback, whether produced manually or automatically. This set should be exactly the same as the results from Method 1 or Method 2, but should be submitted again as one part of Method 3. The second set should be the results after only one iteration of feedback, with the top X documents used in the first iteration of feedback frozen. For example, if you "used" the top 20 documents for feedback, then the second set of results should have these documents as the top 20 documents, followed by the top 200 documents retrieved based on feedback. The term "used" means all documents for which some information has been seen by the judger, and are deemed by the system to have been seen. These two sets of results will be used by NIST to calculate a residual evaluation measure. The third set of results should be a record of your feedback, i.e., a list of documents in the exact order they were seen and judged, with an indication of iteration boundaries. For example, if you ran six iterations of feedback, with 10 documents looked at for each iteration, the record would be a list of the 60 documents seen by the "user", marked at 10, 20, 30, etc. You should also indicate what information the user communicated to your system about each document (relevant/not relevant, too general/too specific/on target, etc.). We will be specifying a format for these record files later. These files will be used to calculate measures based on the total number of relevant documents retrieved both across iterations and across a given document level.

   routing queries -- Method 3 cannot be used for routing queries because routing systems have typically not supported feedback.

In general these guidelines served well, although there was some misunderstanding about what constituted feedback. The guidelines will be clarified for TREC-2.

### 2.3 The Participants

There were 25 participating systems in TREC-1, using a wide range of retrieval techniques. The participants were able to choose from three levels of participation: Category A, full participation, Category B, full participation using a reduced dataset (25 topics and 1/4 of the full document set), and Category C for evaluation only (to allow commercial systems to protect proprietary algorithms). The program committee selected only 20 category A and B groups to present talks because of limited conference time, and requested that the rest of the groups present posters. All groups were asked to submit papers for the proceedings.

Each group was provided the data, and asked to turn in either one or two sets of results for each topic. When two sets of results were sent, they could be made using different methods of creating queries (Methods 1, 2, or 3), or by using different parameter settings for one query creation method. Groups could choose to do the routing task, the adhoc task, or both, and were requested to submit the top 200 documents retrieved for each topic for evaluation.

## 3. The Test Collection

### 3.1 Introduction

Critical to the success of TREC was the creation of the test collection. Like most traditional retrieval collections, there are three distinct parts to this collection. The first is the documents themselves -- the training set (D1) and the test set (D2). Both were distributed as CD-ROMs with about 1 gigabyte of data each, compressed to fit. The training topics, the test topics and the relevance judgments were supplied by email. TREC-1 used the same test collection (documents and topics) used in the DARPA TIPSTER project. (The DARPA TIPSTER project involves the same tasks as TREC, but with four contractors doing more intense research than is being expected from TREC participants (Harman 1993)). However a major increase in the number of relevance judgments for this collection became available from the TREC-1 evaluation.

The components of the test collection -- the documents, the topics, and the relevance judgments, are discussed in the rest of this section.

### 3.2 The Documents

The documents came from the following sources.

- Disk 1
  - WSJ -- Wall Street Journal (1986, 1987, 1988, 1989)
  - AP -- AP Newswire (1989)
  - ZIFF -- Information from Computer Select disks (Ziff-Davis Publishing)
  - FR -- Federal Register (1989)
  - DOE -- Short abstracts from the Department of Energy
- Disk 2
  - WSJ -- Wall Street Journal (1990, 1991, 1992)
  - AP -- AP Newswire (1988)
  - ZIFF -- Information from Computer Select disks (Ziff-Davis Publishing)
  - FR -- Federal Register (1988)

The particular sources were selected because they reflected the different types of documents used in the imagined TREC application. Specifically they had a varied length, a varied writing style, a varied level of editing and a varied vocabulary. All participants were required to sign a detailed user agreement for the data in order to protect the copyrighted source material.

The documents were uniformly formatted into an SGML-like structure, as can be seen in the following example.

<DOC>
<DOCNO> WSJ880406-0090 </DOCNO>
<HL> AT&T Unveils Services to Upgrade Phone Networks Under Global Plan </HL>
<AUTHOR> Janet Guyon (WSJ Staff) </AUTHOR>
<DATELINE> NEW YORK </DATELINE>
<TEXT>
American Telephone & Telegraph Co. introduced the first of a new generation of phone services with broad implications for computer and communications equipment markets.

AT&T said it is the first national long-distance carrier to announce prices for specific services under a world-wide standardization plan to upgrade phone networks. By announcing commercial services under the plan, which the industry calls the Integrated Services Digital Network, AT&T will influence evolving communications standards to its advantage, consultants said, just as International Business Machines Corp. has created de facto computer standards favoring its products.

.
.
.
</TEXT>
</DOC>

All documents had beginning and end markers, and a unique DOCNO id field. Additionally other fields taken from the initial data appeared, but these varied widely across the different sources. The documents also had different amounts of errors, which were not checked or corrected. Not only would this have been an impossible task, but the errors in the data provided a better simulation of the real-world task. Errors in missing document separators or bad document numbers were screened out, although a few were missed and later reported by participants.

Table 1 shows some basic document collection statistics.

TABLE 1. DOCUMENT STATISTICS

| Subset of collection | WSJ | AP | ZIFF | FR | DOE |
|---|---|---|---|---|---|
| Size of collection (megabytes) | | | | | |
| (disk 1) | 295 | 266 | 251 | 258 | 190 |
| (disk 2) | 255 | 248 | 188 | 211 | |
| Number of records | | | | | |
| (disk 1) | 98,736 | 84,930 | 75,180 | 26,207 | 226,087 |
| (disk 2) | 74,520 | 79,923 | 56,920 | 20,108 | |
| Median number of terms per record | | | | | |
| (disk 1) | 182 | 353 | 181 | 313 | 82 |
| (disk 2) | 218 | 346 | 167 | 315 | |
| Average number of terms per record | | | | | |
| (disk 1) | 329 | 375 | 412 | 1017 | 89 |
| (disk 2) | 377 | 370 | 394 | 1073 | |

Note that although the collection sizes are roughly equivalent in megabytes, there is a range of document lengths from very short documents (DOE) to very long (FR). Also the range of document lengths within a collection varies. For example, the documents from AP are similar in length (the median and the average length

are very close), but the WSJ and ZIFF documents have a wider range of lengths. The documents from the Federal Register (FR) have a very wide range of lengths.

The distribution of terms in these subsets show interesting variations. Table 2 shows some term distribution statistics found using a small stopword list of 25 terms and no stemming. For example the AP has more unique terms than the others, probably reflecting both more proper names and more spelling errors. The DOE collection, while very small, is highly technical and has many domains, resulting in many specific technical terms.

TABLE 2. DICTIONARY STATISTICS

| Subset of collection | WSJ | AP | ZIFF | FR | DOE |
|---|---|---|---|---|---|
| Total number of unique terms | | | | | |
| (disk 1) | 156,298 | 197,608 | 173,501 | 126,258 | 186,225 |
| (disk 2) | 153,725 | 186,500 | 147,405 | 116,586 | |
| Occurring once | | | | | |
| (disk 1) | 64,656 | 89,627 | 85,992 | 58,677 | 95,782 |
| (disk 2) | 64,844 | 83,019 | 72,053 | 54,823 | |
| Occurring more > 1 | | | | | |
| (disk 1) | 91,642 | 107,981 | 87,509 | 67,581 | 90,443 |
| (disk 2) | 88,881 | 103,481 | 75,352 | 61,763 | |
| Average number of occurrences > 1 | | | | | |
| (disk 1) | 199 | 174 | 165 | 106 | 159 |
| (disk 2) | 178 | 169 | 139 | 91 | |

How does this document set compare with the older collections? Table 3 shows a comparison of these collections with the Cranfield 1400 collection mentioned earlier. Not only has the size of the document collection increased by a factor of about 200, but the average length of the documents has at least doubled, and in some cases (FR), increased by a factor of 10. Also, the dictionary sizes have increased by a factor of 20.

TABLE 3. COMPARISON TO OLDER COLLECTIONS

| | Subset of collection | | | | |
|---|---|---|---|---|---|
| Size of collection (megabytes) | 295 | 266 | 251 | 258 | 1.5 |
| Number of records | 98,736 | 84,930 | 75,180 | 26,207 | 1400 |
| Median number of terms per record | 182 | 353 | 181 | 313 | 79 |
| Average number of terms per record | 329 | 375 | 412 | 1017 | 88 |
| Total number of unique terms | 156,298 | 197,608 | 173,501 | 126,258 | 8226 |

What does this mean to the TREC task? First, a major portion of the effort for TREC-1 was spent in the system engineering necessary to handle the huge number of documents. This means that little time was left for system tuning or experimental runs, and therefore the TREC-1 results can best be viewed as a baseline for later research. The longer documents also required major adjustments to the algorithms themselves (or loss of performance). This is particularly true for the very long documents in FR. Since a relevant document might contain only one or two relevant sentences, many algorithms needed adjustment from working with the abstract length documents found in the old collections. Additionally many documents were composite stories, with different topics, and this caused problems for most algorithms.

### 3.3 The Topics

In designing the TREC (and TIPSTER) tasks, there was a conscious decision made to provide "user need" statements rather than more traditional queries. Two major issues were involved in this decision. First there was a desire to allow a wide range of query construction methods by keeping the topic (the need statement) distinct from the query (the actual text submitted to the system). The second issue was the ability to increase the amount of information available about each topic, in particular to include with each topic a clear statement of what criteria make a document relevant.

The topics were designed to mimic a real user's need, and were written by people who are actual users of a retrieval system. Although the subject domain of the topics was diverse, some consideration was given to the documents to be searched. The topics were constructed by doing trial retrievals against a sample of the document set, and then those topics that had roughly 25 to 100 hits in that sample were used. This created a range of broader and narrower topics.

The following is one of the topics used in TREC.

<top>
<head> Tipster Topic Description
<num> Number: 066
<dom> Domain: Science and Technology
<title> Topic: Natural Language Processing

<desc> Description:
Document will identify a type of natural language processing technology which is being developed or marketed in the U.S.

<narr> Narrative:
A relevant document will identify a company or institution developing or marketing a natural language processing technology, identify the technology, and identify one or more features of the company's product.

<con> Concept(s):
1. natural language processing
2. translation, language, dictionary, font
3. software applications

<fac> Factor(s):
<nat> Nationality: U.S.
</fac>
<def> Definition(s):
</top>

Each topic was formatted in the same standard method to allow easier automatic construction of queries. Besides a beginning and an end marker, each topic had a number, a short title, and a one-sentence description. There was a narrative section which was aimed at providing a complete description of document relevance for the assessors. Each topic also had a concepts section with a list of assorted concepts related to the topic. This section was designed to provide a mini-knowledge base about a topic such as a real searcher might possess. Additionally each topic could have a definitions section and/or a factors section. The definition section had one or two of the definitions critical to a human understanding of the topic. The factors section was included to allow easier automatic query building by listing specific items from the narrative that constrain the documents that are relevant. Two particular factors were used in the TREC-1 topics: a time factor (current, before a given date, etc.) and a nationality factor (either involving only certain countries or excluding certain countries).

While the TREC topics did not present a problem in scaling, the challenge of either automatically constructing a query, or manually constructing a query with little foreknowledge of its searching capability, was a major challenge for TREC participants. In addition to filtering the relatively large amount of information provided in the topics into queries, the sometimes narrow definition of relevance as stated in the narrative was difficult for

most systems to handle. The two narratives shown below illustrate this point.

<num> Number: 051

A relevant document will cite or discuss assistance to Airbus Industrie by the French, German, British or Spanish government(s), or will discuss a trade dispute between Airbus or the European governments and a U.S. aircraft producer, most likely Boeing Co. or McDonnell Douglas Corp., or the U.S. government, over federal subsidies to Airbus.

<num> Number: 058

A relevant document will either report an impending rail strike, describing the conditions which may lead to a strike, or will provide an update on an ongoing strike. To be relevant, the document will identify the location of the strike or potential strike. For an impending strike, the document will report the status of negotiations, contract talks, etc. to enable an assessment of the probability of a strike. For an ongoing strike, the document will report the length of the strike to the current date and the status of negotiations or mediation.

In a preliminary analysis, the narratives and the factors played a strange and unpredictable role in the results for TREC-1. Systems did as well on topics with very restrictive narratives, such as that of topic 58, as on topics with non-restrictive narratives, such as topic 51. The subject and terms in the entire topic were more important in determining success than the restrictiveness of the narrative. The factors also did not play a major role in system performance. This could change in TREC-2 when groups have more time to adjust their systems to the TREC task.

### 3.4 The Relevance Judgments

The relevance judgments are of critical importance to a test collection. For each topic it is necessary to compile a list of relevant documents; hopefully as comprehensive a list as possible. For the TREC task, three possible methods for finding the relevant documents could have been used. In the first method, full relevance judgments could have been made on all 742,611 documents, for each topic, resulting in over 74 million judgments. This was clearly impossible. As a second approach, a random sample of the documents could have been taken, with relevance judgments done on that sample only. The problem with this approach is that a random sample that is large enough to find on the order of 200 relevant documents per topic is a very large random sample, and is likely to result in insufficient relevance judgments. The third method, the one used in TREC, was to make relevance judgments on the sample of documents selected by the various participating systems. This method is known as the pooling method, and has been used successfully in creating other collections. It was the recommended method in 1975 proposal to the British Library to build a very large test collection (Sparck Jones & van Rijsbergen).

To construct the pool, the following was done.

1. Divide each set of results into results for a given topic
2. For each topic within a set of results, select the top 200 ranked documents for input to the pool
3. For each topic, merge results from all systems
4. For each topic, sort results based on document numbers
5. For each topic, remove duplicate documents

Pooling proved to be an effective method. There was little overlap among the 25 systems in their retrieved documents. Table 4 shows the overlap statistics. The first overlap statistics are for the adhoc topics (test topics against both training documents D1 and test documents D2), and the second statistics are for the routing topics (training topics against test documents D2 only).

| TABLE 4. OVERLAP OF SUBMITTED RESULTS | | | | |
|---|---|---|---|---|
| | Top 200 Possible | Top 200 Actual | Top 100 Possible | Top 100 Actual |
| Average Number of Unique Documents Per Topic (Adhoc, 33 runs, 16 groups) | 6600 | 2398.4 | 3300 | 1278.86 |
| Average Number of Unique Documents Per Topic (Routing, 22 runs, 16 groups) | 4400 | 1932.42 | 2200 | 1066.86 |

For example, out of a maximum of 6600 unique documents (33 groups times 200 documents), over one-third were actually unique. The top 100 documents retrieved contained about the same percentage of unique documents. This means that the different systems were finding different documents as likely relevant documents for a topic. Whereas this might be expected (and indeed has been shown to occur, Katzer et. al. 1982) from widely differing systems, these overlaps were often between two runs for a given system, or between two systems run on the same basic retrieval engine. One reason for the lack of overlap is the very large number of documents that contain many of the same keywords as the relevant documents, but probably a larger reason is the very different sets of keywords in the constructed queries (this needs further analysis). This lack of overlap should improve the coverage of the relevance set, and verifies the use of the pooling methodology to produce the sample.

The merged list of results was then shown to the human assessors. Only the top 100 documents were judged, resulting in an average of 1462.24 documents judged for each topic, and ranging from a high of 2893 for topic 74 to a low of 611 for topic 46. Each topic was judged by a single assessor to insure the best consistency of judgment and varying numbers of documents were judged relevant to the topics. Figure 3 shows the number of documents judged relevant for each of the 100 topics. The topics are sorted by the number of relevant documents to better show their range and median.

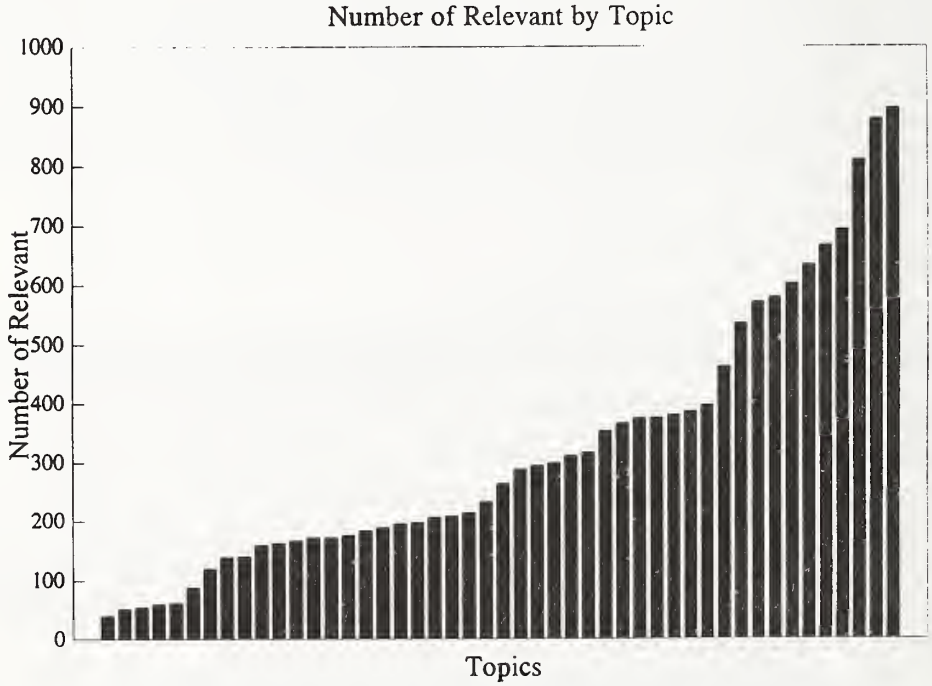



Figure 3. Number of Relevant Documents on a Per Topic Basis.

## 4. Evaluation

### 4.1 Existing Evaluation Methodology

An important element of TREC was to provide a common evaluation forum. Standard recall/precision figures were calculated for each system and the tables and graphs for the results are presented in Appendix A. Figure 4 shows a typical recall/precision curve for illustration purposes. The x axis plots the recall values at fixed levels of recall, where

$$Recall = \frac{number\ of\ relevant\ items\ retrieved}{total\ number\ of\ relevant\ items\ in\ collection}$$

The y axis plots the average precision values at those given recall values, where precision is calculated by

$$Precision = \frac{number\ of\ relevant\ items\ retrieved}{total\ number\ of\ items\ retrieved}$$

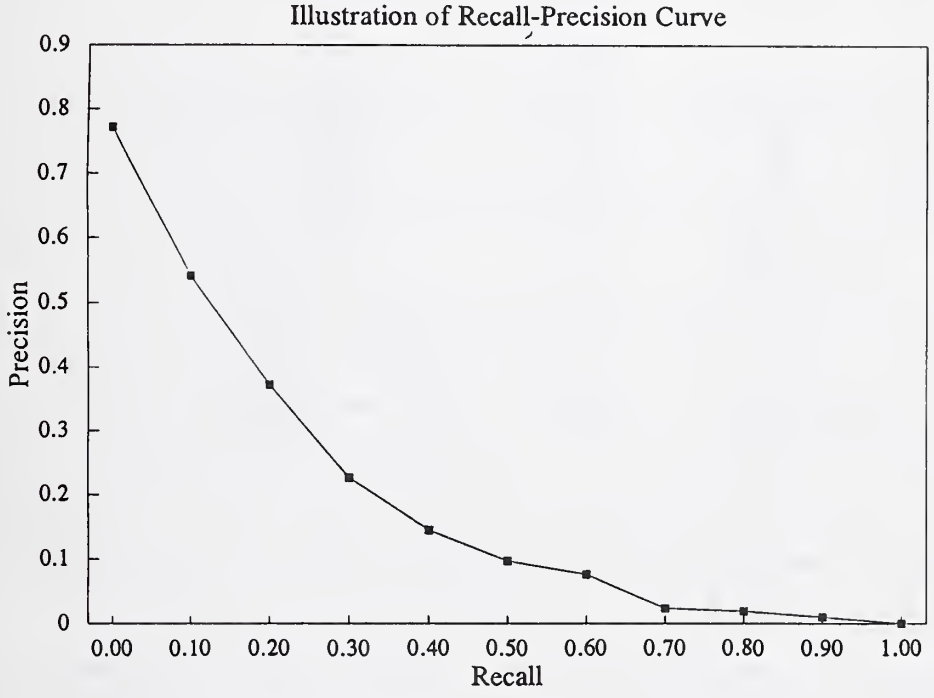



Figure 4. A Sample Recall/Precision Curve.

There is a standard table and graph in Appendix A for each run for each system, with the runs identified by their unique tags. A map for matching the tags to the systems is also provided. Note that the tables for the TIPSTER panel are in Appendix B as the results are not directly comparable to the TREC results. The tables show some total statistics for each run, plus both the recall-level and document-level recall/precision averages.

A second type of information about each system is shown in Appendix C. These standardized forms describe system features and system timing, and allow some primitive comparison of the amount of effort needed to produce the results.

### 4.2 Problems with Evaluation

Since this was the first time that such a large collection of text has been used in evaluation, there were some problems using the existing methods of evaluation. First, groups were asked to send in only the top 200 documents retrieved by their systems. This artificial document cutoff is relatively low and systems did not retrieve all the relevant documents for most topics within the cutoff. All documents retrieved beyond the 200 were considered nonrelevant by default and therefore the recall/precision curves become inaccurate after about 40% recall on average. Table 5 shows a comparison of one system using no threshold (so relevant documents found beyond the 200 limit are marked as relevant) versus using the 200 document threshold.

TABLE 5. COMPARISON OF TABLES FROM TIPSTER

| Full Ranking | | Top 200 Ranking | |
|---|---|---|---|
| Recall | Precision | Precision | |
| 0.0 | 0.821 | 0.8208 | |
| 0.1 | 0.672 | 0.6710 | |
| 0.2 | 0.581 | 0.5759 | |
| 0.3 | 0.528 | 0.5030 | |
| 0.4 | 0.472 | 0.3819 | |
| 0.5 | 0.424 | 0.2999 | |
| 0.6 | 0.368 | 0.1773 | |
| 0.7 | 0.315 | 0.1075 | |
| 0.8 | 0.244 | 0.0487 | |
| 0.9 | 0.154 | 0.0117 | |
| 1.0 | 0.039 | 0.0000 | |
| 11 pt. average | 0.421 | 0.3271 | |
| Recall | Precision | Recall | Precision |
| 0.25 | 0.559 | 0.20 | 0.5759 |
| 0.50 | 0.424 | 0.50 | 0.2999 |
| 0.75 | 0.280 | 0.80 | 0.0487 |
| 3 pt. average | 0.421 | | 0.3082 |

It can be seen from these tables that not only are the recall-level statistics beyond about 40% recall inaccurate, but both the 11 pt. and the 3 pt. averages based on this table are also inaccurate. Since all systems were compared using the same measures, this problem is not serious in terms of comparing methods within TREC-1. However, it could be improved by lowering the threshold, and TREC-2 will be run such that at least the top 500 documents are used for evaluation.

A related problem occurred because some systems in TREC-1 worked on a variable thresholding system, with that threshold set for each topic. Documents not matching sufficient system criteria were rejected, even if fewer than 200 were returned. Sometimes as few as 10 documents were sent as results, and the evaluation method again assumed all documents beyond the 10 were not relevant. This hurt performance for these systems badly in some cases and the individual system papers discuss this. The plans for TREC-2 are to include some additional thresholding tests, so that these systems can evaluate how their thresholding performs and evaluate the standard ranking as done by other systems.

The third problem was more general in nature. The current recall/precision measures do not include any indication of the collection size. This means that the recall and precision of a system based on a 1400 document collection could be the same as that of a system based on a million document collection, but obviously the discrimation powers on a million document collection would be much greater. This may not have been a problem on the smaller collections, but the discrimation power of systems on TREC-sized collections is very important. Clearly some new evaluation measures are needed for this.

One new measure being tried in TREC-1 is the ROC (Relative Operating Characteristic) curves used in signal processing. These curves are similar to the recall/precision curves, but allow the total size of the collection to influence performance. The two variables being used here are the probability of detection or probability of a

"hit" versus the probability of false alarm, or the probability of a "false drop". The x axis plots the probability of false alarm, calculated as follows

$$Probability\ of\ false\ alarm\ =\ \frac{number\ of\ nonrelevant\ items\ retrieved}{total\ number\ of\ nonrelevant\ items\ in\ collection}$$

The y axis plots the probability of detection, calculated as

$$Probability\ of\ detection\ =\ \frac{number\ of\ relevant\ items\ retrieved}{total\ number\ of\ relevant\ items\ in\ collection}$$

Note that the probability of detection is the same as recall, and the probability of false alarm is the same as fallout, an older measure in information retrieval (Salton & McGill 1983). These measures are for a single topic, but averages can be computed similarly to the recall-level averages by using probability of detection at fixed false alarm rates. The tables in Appendix A show both this average ROC curve and the same curve plotted on probability scales (Swets 1969).

## 5. Preliminary Results

### 5.1 Introduction

The results of the TREC-1 conference should be viewed only as a preliminary baseline for what can be expected from systems working with large test collections. There are several reasons for this. First, the deadlines for results were very tight, and most groups had minimal time for experiments. As discussed earlier, the huge scale-up in the size of the document collection required major work from all groups in rebuilding their systems. Much of this work was simply a system engineering task: finding reasonable data structures to use, getting indexing routines to be efficient enough to finish indexing the data, finding enough storage to handle the large inverted files and other structures, etc.

The second reason these results are preliminary is that groups were working blindly as to what constitutes a relevant document. There were no reliable relevance judgments for training, and the use of the long topics was completely new. This means that results were heavily influenced by an almost random selection of what parts of the topic to use. Groups also had to make often primitive adjustments to basic algorithms in order to get results, with little evidence of how well these adjustments were working. The large scale of the whole evaluation precluded any tuning without some relevance judgments, and the relevance judgments that were provided were generally sparse and sometimes inaccurate. These problems particularly affected those systems that needed training for routing.

Many of the papers in the proceedings show some new results from work done in the short amount of time between the conference and the due date of the papers (less than 2 months). Some of the improvements are very significant, and the improvements seen in the TIPSTER results (where the results are a second-try at this task) are large. It can be expected that the results seen at the second TREC conference will be much better, and also more indicative of how well a method works.

Because these results are preliminary, they should be compared very carefully. Some very broad conclusions can be drawn, but no methods should be conclusively judged inferior or superior at this point.

### 5.2 Adhoc Results

The adhoc evaluation used new topics (51-100) against the two disks of documents (D1 + D2). There were 33 sets of results for adhoc evaluation in TREC, with 20 of them based on runs for the full data set. Of these, 13 used automatic construction of queries, 6 used manual construction, and 1 used feedback. Figure 5 shows the recall/precision curve for the three TREC-1 runs with the highest 11-point averages using automatic construction of queries. These curves were all based on the use of the Cornell SMART system, but with important variations. The "fuhrp1" results came from using the training data to find parameter weights (see Fuhr & Buckley paper), the "crnlp1" results came from doing local and global term weighting without training data (see Buckley, Salton & Allan paper), and the "siems1" results came from using term expansion with terms from "Wordnet" (see Voorhees & Hou paper).

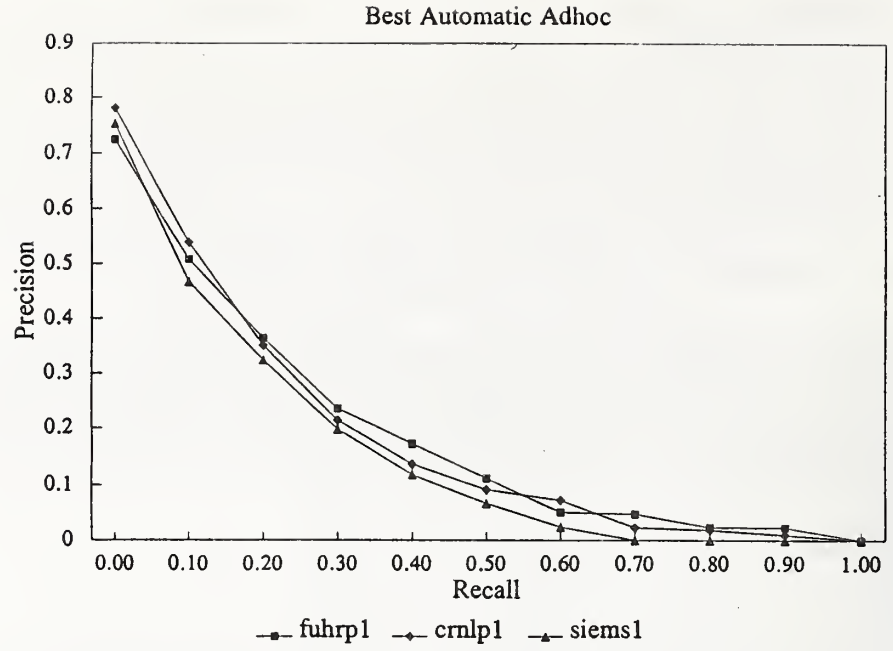



Figure 5. The Best Adhoc Results using Automatic Query Construction.

Figure 6 shows the recall/precision curve for the three TREC-1 runs with the highest 11-point averages using manual construction of queries. It should be noted that varying amounts of manual intervention were used, and this should be considered when comparing results. These curves show differences in that the "clartb" and "gecrd2" have initially a high precision, but lose this precision as recall increases, whereas the "cnqst2" method has a lower initial precision, but higher precision at the higher recall levels. This may be a function of the very different methods being used. The "clartb" system adds noun phrases found in likely relevant documents to improve the query terms taken from the topic (see Evans paper), whereas the "cnqst2" system uses more general thesaurus entries to expand the query (see Nelson paper). The "gecrd2" system uses a totally different approach of constructing elaborate Boolean pattern matchers (see Jacobs, Krupka & Rau paper).

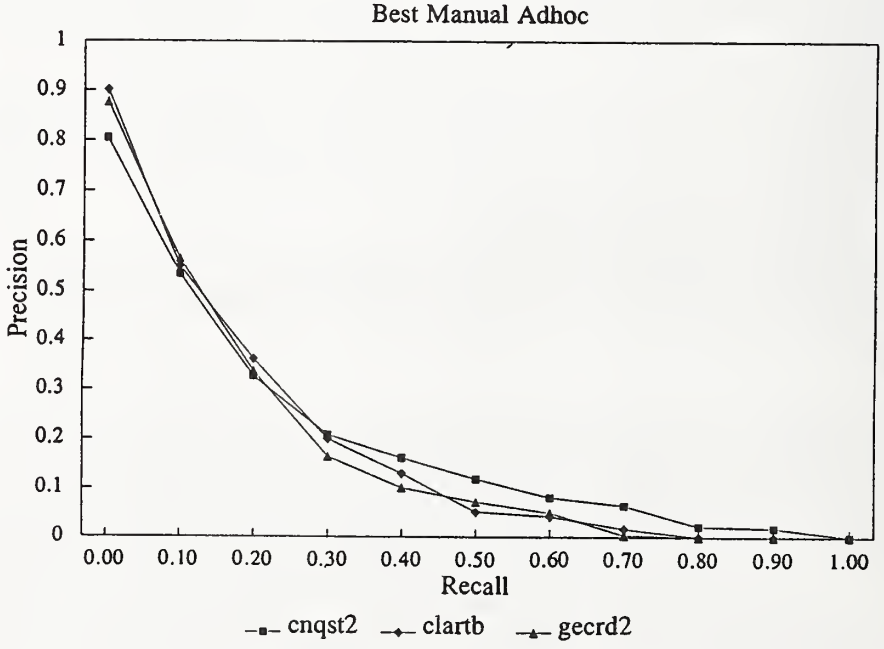



Figure 6. The Best Adhoc Results using Manual Query Construction.

It is useful to contrast the three methods of query construction. Figure 7 shows a comparison of five sets of results, two from automatic query construction, two using manual query construction, and the one relevance feedback run. It should be noted that there is relatively little difference between the results from automatic query construction versus manual query construction, although the relevance feedback results (citym2) were poor in this case. Figure 8 shows a histogram of the same information, but for all adhoc systems working with the full data set. In general it shows that the automatic query construction seems to work well for many systems, and that certainly it can be concluded that for TREC-1 the automatic construction of queries was as effective as the manual construction.

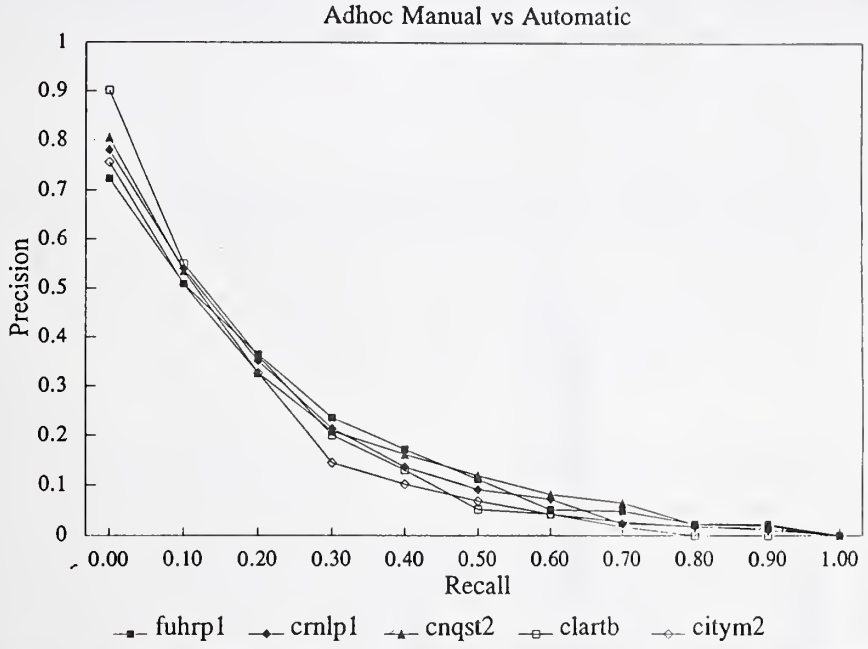



Figure 7. A Comparison of Adhoc Results using Different Query Construction Methods.

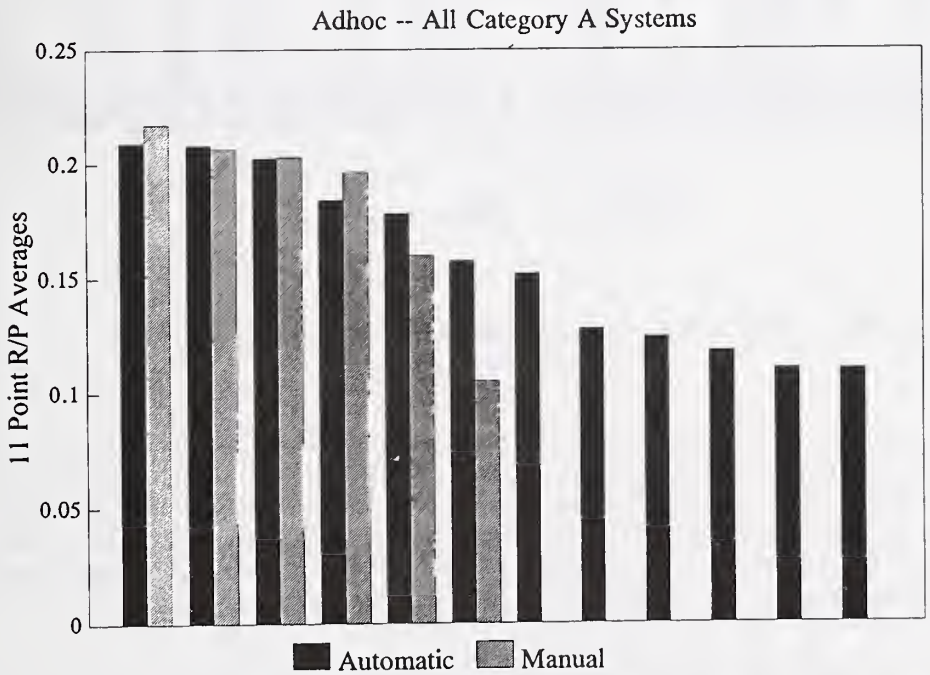



Figure 8. A Comparison of Adhoc Results using Automatic and Manual Query Construction.

Figure 9 shows the comparison of automatic and manual query construction on a per topic basis. It is interesting to note that, for the two systems shown, most topics show equal performance in terms of the percentage of relevant documents retrieved by 100 documents. Some topics, like topic 51, show much better manual performance, whereas other topics, like topic 69 show better automatic performance. This is somewhat different results from earlier comparisons of Boolean systems (usually manual indexing and manual query construction) versus the automatic systems such as the SMART system. In the Medlars study (Salton 1969) the manual (Boolean) systems seemed to do either very well or very poorly, whereas the automatic systems produced consistent "medium" results. The difference in the TREC task is likely that the topics are very long and complex, and sometimes are easy to express manually, but sometimes very difficult, whereas the automatic construction is hampered by the existence of difficult narratives. This is only a hypothesis and needs further investigation.

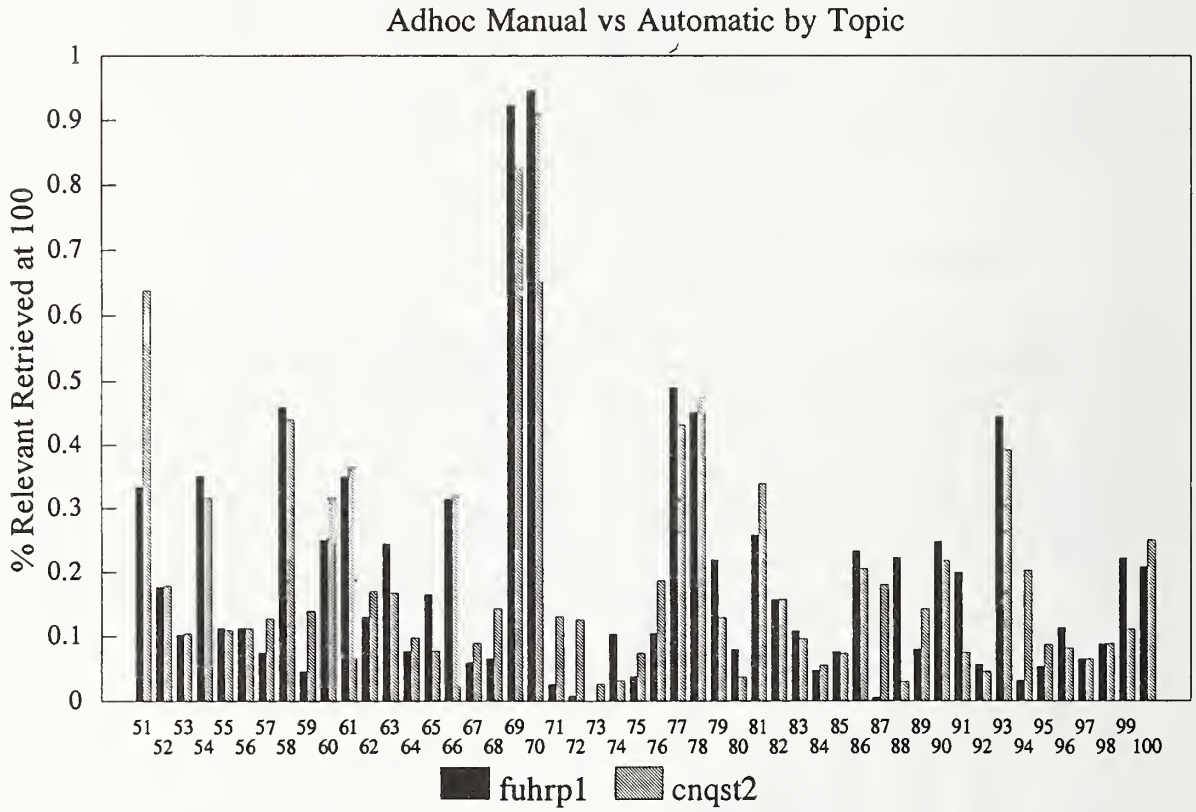



Figure 9. Adhoc Results using Automatic and Manual Query Construction.
on a Per Topic Basis

There were also some category B results for adhoc, and the best of these are shown in Figure 10, with results from the Cornell system run as a category B run to show some comparison. There is a wide spread in the curves here, with widely differing systems being shown. The "pircs4" results represent a very successful relevance feedback method, with the "pircs1" being an automatic query construction using the same system (see Kwok, Padadopoulos & Kwan paper). The "nyuir1" results come from a system using natural language techniques (see Strzalkowski paper).

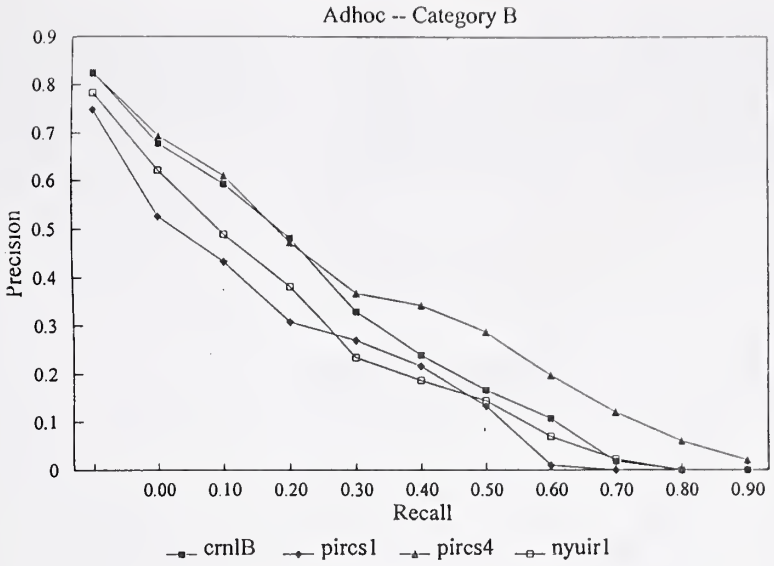



Figure 10. Adhoc Results for Category B.

### 5.3 Routing Results

There were 22 sets of results for routing evaluation, with 16 of them based on runs for the full data set. Note that all routing techniques suffered from the lack of sufficient and accurate training data, and therefore these results are even more preliminary than the adhoc results. Of the 16 systems using the full data set, 8 used automatic construction of queries, and 8 used manual construction. Figure 11 shows the recall/precision curve for the three TREC-1 runs with the highest 11-point averages using automatic construction of queries. Two of the curves, based on the use of the Cornell SMART system, show very different results. The "fuhra2" results came from using a probabilistically-based relevance feedback (see Fuhr & Buckley paper), whereas the "crnla2" results came from doing traditional relevance feedback methods using the vector space model (see Buckley, Salton & Allan paper), The "cityr1" results also came from using traditional relevance feedback, but using a different probabilistic model and term weighting (see Robertson, Walker, Hancock-Beaulieu, Gull & Lau paper). The "cpgcn2" system used filtering methods rather than more traditional information retrieval methods to achieve results similar to the feedback results (see Jones, Leung, and Pape paper).

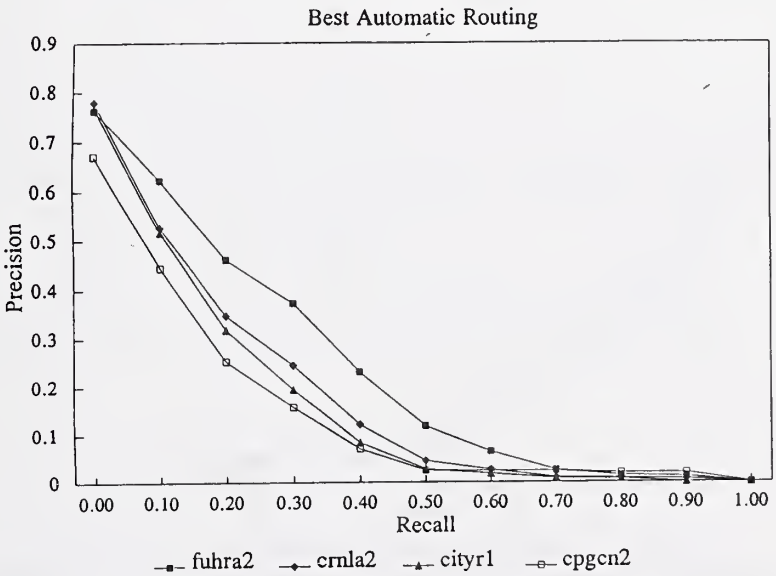



Figure 11. The Best Routing Results using Automatic Query Construction.

Figure 12 shows the recall/precision curve for the three TREC-1 runs with the highest 11-point averages using manual construction of queries. The systems used manually-built filters, with the "clartb" and "gecrd2" results done similarly to their corresponding adhoc systems, but using the sample relevant documents as input to the filter-building process. The "paraz1" system used manually-constructed filters based on clusters of interesting terms (see Zimmerman paper). The "cpghc2" group hand-crafted these queries as a contrast to their automatic pattern filtering methods (see Jones, Leung & Pape).

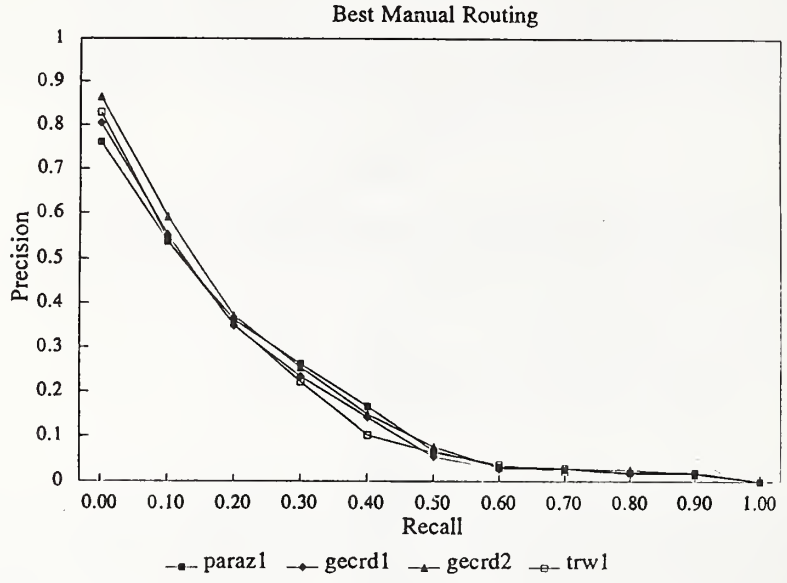



Figure 12. The Best Routing Results using Manual Query Construction.

Again it is useful to contrast the methods of query construction. Figure 13 shows a comparison of four sets of results, two from automatic query construction and two using manual query construction. Here, unlike the adhoc results, the automatic query building seems to be clearly superior, with the "fuhr1" results having higher performance throughout the significant part of the recall/precision curve.

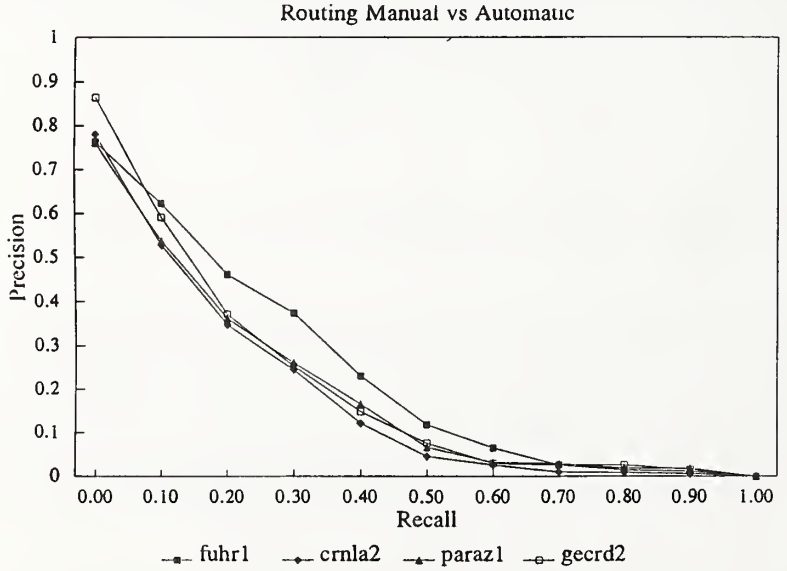



Figure 13. A Comparison of Routing Results using Different Query Construction Methods.

There were also some category B results for routing, and the best of these are shown in Figure 14. Again there is a much wider spread in the curves here, and widely differing systems being shown. The "pircs1" and "pircs2" results are correspondingly automatic and manually constructed queries using relevance feedback learning from the training sample (see Kwok, Padadopoulos & Kwan paper). The "fairs1" system uses a combination of different term weighting methods (see Chang, Dediu, Assam & Du paper). The "nyuir1" results come from a system using natural languages techniques and probably reflect the short amount of time available to construct this very complicated system (see Strzalkowski paper).

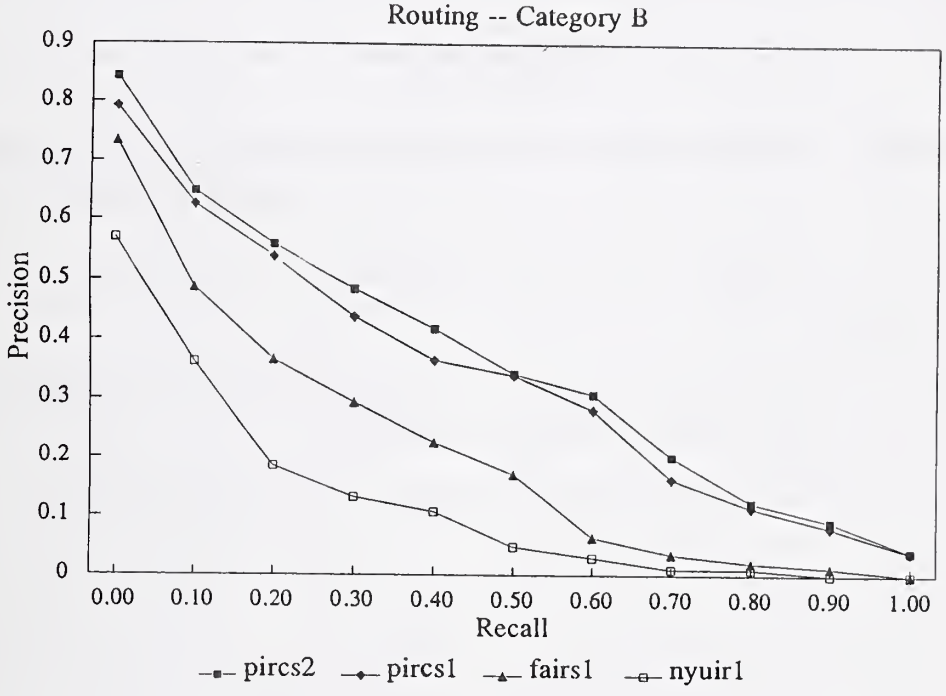



Figure 14. Routing Results for Category B.

### 5.4 Summary

The TREC-1 conference demonstrated a wide range of different approaches to the retrieval of text from large document collections. Because of the preliminary nature of the results, very little can be said about which techniques seem to perform the best. It was clear that the simple systems did the task well, but it is too early to pass judgment on the more complicated systems. The automatic construction of queries from the topics did as well as, or better than, manual construction of queries, and this is encouraging for groups supporting the use of simple natural language interfaces for retrieval systems.

There will be a second TREC conference in 1993, and all the systems that participated in TREC-1 will be back, along with additional groups. The results from this second conference should better identify the more promising retrieval techniques.

## REFERENCES

Cleverdon C.W., (1962). *Report on the Testing and Analysis of an Investigation into the Comparative Efficiency of Indexing Systems*. College of Aeronautics, Cranfield, England, 1962.

Cleverdon C.W., Mills, J. and Keen E.M. (1966). *Factors Determining the Performance of Indexing Systems, Vol. 1: Design, Vol. 2: Test Results*. Aslib Cranfield Research Project, Cranfield, England, 1966.

Harman D. (1993). The DARPA TIPSTER Project. *SIGIR Forum*, 26(2), 26-28.

Katzer J., McGill M.J., Tessier J.A., Frakes W., and DasGupta P. (1982). A Study of the Overlap among Document Representations. *Information Technology: Research and Development*, 1(2), 261-274.

Fox E. (1983). Characteristics of Two New Experimental Collections in Computer and Information Science Containing Textual and Bibliographic Concepts. *Technical Report TR 83-561*, Cornell University: Computing Science Department.

Salton G. (1969). A Comparison Between Manual and Automatic Indexing Methods. *American Documentation*, 20(1).

Salton G. and McGill M. (1983). *Introduction to Modern Information Retrieval*. New York, NY.: McGraw-Hill.

Sparck Jones K. (1981). *Information Retrieval Experiment*. London, England: Butterworths.

Sparck Jones K. and Van Rijsbergen C. (1975). *Report on the Need for and Provision of an "Ideal" Information Retrieval Test Collection*, British Library Research and Development Report 5266, Computer Laboratory, University of Cambridge.

Sparck Jones K. and Webster (1979). *Research in Relevance Weighting*, British Library Research and Development Report 5553, Computer Laboratory, University of Cambridge.

Swets J. (1969). Effectiveness of Information Retrieval Methods. *American Documentation*, 20(1).

# Okapi at TREC

Stephen E. Robertson, Stephen Walker,
Micheline Hancock-Beaulieu, Aarron Gull, Marianna Lau

Centre for Interactive Systems Research
Department of Information Science
City University
Northampton Square
London EC1V 0HB, UK

**Advisers:** Karen Sparck Jones (University of Cambridge); Peter Willett (University of Sheffield); E. Michael Keen (University of Wales).

**Abstract:** The Okapi retrieval system is described, technically and in terms of its design principles. These include simplicity, robustness and ease of use. The version of Okapi used for TREC is further discussed. Designing experiments within the TREC constraints but using Okapi's supposed strengths proved problematic, and some compromise was necessary. The official TREC runs were (a) very simple automatic processing of the ad-hoc topics; (b) manually constructed ad-hoc queries; (c) feedback on the manual queries from searchers' relevance judgements; and (d) routing queries automatically obtained using the training set in a form of relevance feedback. The best run (manual with feedback), although not up to the best reported TREC results, was respectable, and an encouragement to further development within the same principles.

## 1. Introduction

Okapi is an experimental text retrieval system, designed to use simple, robust techniques internally and to present a user interface which requires no training and no knowledge of searching methods or techniques. It is presently accessible by academic users at City University, with the library catalogue and a scientific abstracts journal as databases. It is used for experimentation with and evaluation of novel retrieval techniques.

A design principle of Okapi is that simple techniques, without Boolean logic but with best-match searching, and with little in the way of a manually constructed knowledge base, can give effective and efficient retrieval. 'Simple' also implies minimum effort, either manual or machine, either at the set-up stage or at input or at search time. In particular, relevance feedback (which requires little or no additional user effort, since users must make such judgements anyway), provides a mechanism whereby an initial query formulated with no great effort can be improved. Such a search process might be regarded as having something of the character of browsing: an exploration of a topic rather than a precise specification.

In some respects (e.g. highly elaborate topic specifications; no evaluation of interactive systems) TREC does not at all represent the kind of retrieval activities for which Okapi was designed. However, our approach to TREC has been to try to arrive at some compromise between the aims of Okapi and those of TREC. The resulting performance was not spectacular, but was (we believe) respectable enough to encourage us to pursue the ideas further.

## 2. Background: the Okapi project

The following is a description of Okapi as it existed before the start of TREC-related work ("interactive Okapi"). Section 3 discusses some

changes which happened concurrently with, and were necessary for, the TREC work.

Okapi is a family of bibliographic retrieval systems, developed under a series of grants from the British Library. It is suitable for searching files of records whose fields contain textual data of variable length up to a few tens of thousands of characters. It allows the implementation of a variety of search techniques based on the probabilistic retrieval model, with easy-to-use interfaces, on databases of operational size and under operational conditions (Walker, 1989; Walker & De Vere, 1990; Walker & Hancock-Beaulieu, 1991; Hancock-Beaulieu & Walker, 1992).

The main purpose of the Okapi installation at City is to allow the use of a variety of evaluation methods, including live-user evaluation in the context of user information-seeking behaviour.

### 2.1 Search techniques

The interactive Okapi system uses probabilistic "best match" searching, and can handle queries of up to 32 terms. (There is no Boolean search facility in interactive Okapi -- but see 3.2 below concerning the development system.) Search terms may be keywords or phrases, or any other record component which has been indexed, and are extracted automatically by very simple "parsing" of an initial natural language query. Search terms are assigned weights, based on inverse document frequency in the absence of relevance information and on the F4 formula given in Robertson & Sparck Jones (1976) when relevance information is available. The match function is a simple sum-of-weights. There are facilities for "adjusting" the weighting to favour (for example) terms occurring in specified fields. There is also a limited alphabetical browsing facility (of records in index term order).

The F4 formula, point-5 version, is:

$$w = \log \frac{(r+0.5)\ (N-R-n+r+0.5)}{(R-r+0.5)\ (n-r+0.5)}$$

where N = collection size
n = number of postings of term
R = total known relevant documents
r = number of these posted to the term

The inverse document frequency (IDF) weight is F4 with R=r=0, i.e.

$$w = \log (N-n+0.5)/(n+0.5)$$

### 2.2 Relevance feedback and query expansion

The system can invite relevance judgments from the user, and following one or more positive relevance assessments it can perform an "expanded" search, using the original query terms together with additional terms extracted automatically from the relevant records. This procedure can be iterated.

### 2.3 Language processing

Very simple text and linguistic processing is applied during indexing and searching.

There are two levels of automatic stemming, and a mainly rule-based procedure for conflating British and American spellings.

There are facilities for constructing and using a simple linguistic knowledge base containing "go" phrases, classes of terms to be treated as synonymous, prefixes, stopwords and phrases, and "semi-stopwords" --- words and phrases to be treated as relatively unimportant in processing a query.

### 2.4 Usage

The interactive system is intended for highly interactive use by untrained users.

### 2.5 Logging

The system can produce detailed logs of both user and system activity, down to keystroke level and sub-second granularity.

### 2.6 Present use and status

The present use of Okapi is primarily as a tool for the evaluation of highly interactive bibliographic search systems with untrained users. It is also to be used in an investigation of the use of linguistic knowledge structures (e.g. thesauri) in text retrieval systems.

The system is not commercially available. It is not finished, maintained or documented to commercial standards. It is, however, designed for live use, and there has, over the years, been a

considerable amount of use under live conditions. It is a set of functions from which experienced designers and programmers can construct retrieval systems, rather than a finished "product".

## 3. Concurrent developments

### 3.1 Towards a distributed system

This development reflects a long-standing plan for the Okapi project, but was brought forward to facilitate work on the TREC database.

Okapi has been split into a Basic Search System (BSS) and a number of front-end systems. The BSS is essentially a database engine offering basic text retrieval functionality, extended in various ways to allow weighting, ranking and relevance feedback etc. Although the front-end systems at present reside on the same machine, the dialogue between the front-end and the BSS is roughly comparable to that which might take place using the Z39.50 or Search & Retrieve protocols. It concerns mainly specifications for and descriptions of search sets, and involves actual records only at the time of display.

All automatic searching for the TREC project involved purpose-written front-ends to the BSS. A further front-end was developed for manual searching. This was designed to include most of the functionality of the old interactive version of Okapi, but not to emulate its user interface; it is command-driven.

### 3.2 Mixing Boolean and weighted searching

One characteristic of the BSS needs explaining. The BSS is capable of conducting Boolean searches as well as weighted (best match) searches. Furthermore, any Boolean expression (resulting in an undifferentiated search set) can be treated as if it were a single term in the weighted searching model. This is compatible with the approach taken in the Cirt system (which acted as a front-end to a Boolean host) (Robertson et al., 1986); particular examples of uses in Cirt include ORed synonyms and phrases constructed with the ADJ operator. The Okapi BSS does not at present allow proximity operators such as ADJ, but the principle is the same.

To a very limited extent, this facility was used by the manual searchers (see 5.3).

### 3.3 Term selection for query expansion

Interactive Okapi automatically selected terms from relevant documents for query expansion by taking the top x (=20) terms according to their relevance weights. The BSS version uses the Robertson selection value (Robertson, 1990), approximately `r*w` (where `w` is the usual F4 weight). (See also discussion in section 6.3, which shows that there was an error in taking this approximation.) Also, the interface used in the manual TREC experiments allows semi-automatic query expansion, in that the list of candidate terms can be displayed for the searcher to make selections from (and then entered manually), or the top 20 terms can be used automatically.

Terms once selected are weighted using F4 in the usual way, except with the modification indicated below.

### 3.4 Bias towards query terms

In interactive Okapi, the terms in the original query held no special position in the query expansion process, except in the sense that a "semi-stopword" in the original query would be a candidate for the feedback query, whereas the same term occurring in a relevant document but not in the query would not be considered.

For the TREC experiments, some bias in favour of query terms was built in, in the form of some hypothetical relevant documents assumed to contain the query terms (Harman, 1992; Bookstein, 1983). These hypothetical relevant documents then contributed to the calculation of F4. Different quantitative assumptions were made in different TREC experiments (see section 5), but once again an error crept into the implementation of this facility (see section 6.3).

## 4. Input processing

### 4.1 Converting the raw files

The Okapi system needs databases to be in its own format, in which each record consists of an identical sequence of fields in the form of terminated text strings. Fields are identified by sequence number only. Using the given

information about the makeup and structure of the source material, together with quite a lot of trial and error, a program (lex + C) was written to convert all the raw datasets into a unified 25-field structure. The only fields common to all input were "text", document-ID and a "source" field containing "fr", "doe" etc., so all records consisted mainly of empty fields. The "source" fields were intended solely as "limit" criteria, but were unused except perhaps by one or two of the human searchers. Fields other than the three mentioned were used solely for display. Any records longer than 64K were truncated at 64K. This truncation only affected the text field (field 25).

Conversion, which included decompression, conversion to Okapi format using the lex-C program and a second-stage conversion to runtime format ran at about 10 records/sec on a SPARC machine.

### 4.2 Inversion

The text field was reduced to "words", stemmed using the moderate-strength Porter algorithm (Porter, 1980) with modifications aimed at conflating British and American spellings, filtered through a local database (GSL, see below) containing stopwords, semi-stopwords, prefixes, a few "go" phrases (phrases to be treated as words), and a list of classes of words and phrases to be treated as synonymous. The document-ID field was extracted unchanged. Inversion took about 33 hours CPU on a SPARC machine (about 6 documents per second, but this increases more than linearly with number of documents). The result was a simple inverted file structure with no within-field positional information (insufficient disk space). There were facilities for limiting searches by source dataset, by various document length ranges and by odd/even half-collection (for comparison experiments).

### 4.3 The local GSL database

The Go-See-List (GSL) for the TREC experiments was based on existing databases, but was somewhat extended for TREC. Both original and extensions were derived in a fairly ad-hoc fashion (some entries were identified by examining a list of the most frequent terms in the first part of the TREC collection). This is not a sophisticated facility, and can only be said to scratch the surface of the problem.

Stopwords: 120

Semi-stopwords: 256

These were humanly selected following trial indexing runs. The criteria were (1) small retrieval value and (2) high posting count. Examples: 100, begin, carry, date, december, enough, include, meanwhile, run, take, why, without, yesterday.

Prefixes: 18

The purpose of this list is to cause <prefix>-<word> and <prefix><word> to be treated identically for any value of <word>

Go phrases: 27

Examples: cold war, middle class, saudi arabia

Synonym classes: 300, containing about 700 words

Examples:

australia, australian, australasia, australasian
buyout, buy out
mit, massachusetts institute of technology
porn, porno, pornography, pornographic

### 4.4 Some statistics

| | First part | Second part | Both |
|---|---|---|---|
| Total documents | 511514 | 230936 | 742450 |
| Truncated (over 64K) | 603 | 531 | 1134 |
| Size (MB) (bibfile only, runtime format) | 1107 | 759 | 1866 |
| Inversion overheads (%) | 44 | N/K | 44 |
| Unique index terms (excluding document numbers) | | | 1040415 |
| Mean unique index terms/document | | | 1.43 |
| Postings (excluding document numbers) | | | 95898880 |
| Mean postings/document (document "length") | | | 132 |

## 5. Experiments

Following the TREC design, results were submitted for routing queries on the second set of records, and for ad-hoc queries on the combined set. Routing queries were processed automatically only (section 5.2; results table **cityr1**); ad-hoc queries were processed automatically (5.1; **citya1**) and manually, with feedback on the manual searches (5.3; results without feedback in table **citym1**, and with

feedback in **citym2**).

In accordance with the general philosophy of Okapi, the TREC experiments were used to test the use of simple statistical techniques, with minimal linguistic processing, minimal searcher knowledge of techniques of searching, and indeed minimal effort generally. The routing test was intended to address mainly the value of relevance feedback (term selection and weighting) in a routing context where relevance judgements are accumulated from earlier runs. Automatic ad-hoc queries tested the weighting scheme without relevance information. Manual ad-hoc queries tested the combination of human intelligence with a simple weighting scheme, with and without feedback.

### 5.1 Automatic processing of topics

The basic principle was to take specific section(s) of the topic and parse them in standard Okapi fashion, as if they had been typed in verbatim by a searcher. Thus stopwords were removed; a few phrases and/or members of synonym classes were identified; remaining words were stemmed; all search terms (stems or phrases or synonym classes) were weighted using IDF (see section 2.1). No special account was taken of the negative phrases which appear in some of the TREC topics, so that negated words would have been given positive weights by Okapi.

The selection of the topic sections was the subject of a very small amount of initial experimentation using the training set. The differences were not very consistent and in some cases small, and more testing would have been useful. However, marginally the best overall was Concepts only, and that was what we used for the returned results.

The results for the above automatic analysis of ad-hoc queries are given in the official tables as citya1.

### 5.2 Routing queries

The principle on the routing queries was to assume that all the known relevant documents from the first document set were already available for a relevance feedback process. Thus any actual searches conducted on the first document set, and their actual outputs, played no direct part in the formulation of the routing queries, with one exception discussed below. However, the terms extracted from the topics took part in the relevance feedback process in the manner indicated in section 3.4, with a bias equivalent to 10 supposed relevant documents in all of which the topic terms were supposed to occur (10 out of 10 bias).

The exception to the above statement was that for *some* topics, *some* additional relevance assessments were made (that is, additional to those provided centrally). These were based on the top ranked documents retrieved in automatic searches on the first document set. (See section 6.1 for a discussion on the local relevance judgements and on the reasons for this decision.)

The results for the above analysis of routing queries are given in the official tables as cityr1.

### 5.3 Manual searching and feedback

The central idea behind these experiments was to approach as closely as possible the situation of a naive or inexperienced user. In other words, we wanted to gain some idea of how the system would perform if searched by an end-user with little or no knowledge of information retrieval. This intention reflects the design principles of the interactive Okapi, as discussed in section 2 above. To some degree, however, both the design of the TREC experiment in general and the constraints of the distributed system described in section 3.1 forced deviations from that ideal.

#### 5.3.1 Searchers

The first constraint is of course that we had no access to end-users (and more particularly, no access to end-users with the specific characteristics of the TREC analysts). We used a panel of searchers, mainly information science students who could be said to have some knowledge of searching in general, limited domain knowledge (depending on the topic), and no particular knowledge of the system. (For reasons to do with the very limited time available for these searches, it was necessary to use project staff for a few searches; these staff obviously had more knowledge of the system.) The somewhat limited interface to the BSS which was used for this experiment required some training of the searchers.

Clearly one would in general expect end-users to have more domain or subject knowledge, especially for the kinds of queries provided for TREC. Highly interactive systems in general, and Okapi in particular, may be assumed to exploit such subject knowledge; clearly relevance feedback in ad-hoc searching can only work well if it is relatively easy for the user to find some relevant items from the initial search. In this sense, we see the present experiment as to some degree unfavourable to Okapi.

### 5.3.2 Searching

Searchers were expected to make whatever interpretations of the topic they deemed appropriate for the purpose of searching. In other words, they could use words or phrases taken from any part of the topic, or from their own general or specific knowledge. They could also have used other reference sources. However, they were encouraged to use the system to help them refine the search, in the way that an end-user might explore the possibilities within the system and try out different combinations of search terms.

The combination of these ideas with the TREC rules was a little clumsy and artificial. The procedure was as follows:

(a) The searcher was given the topic in full, as received by us.

(b) The searcher examined the topic and chose some terms as candidates for searching (possibly including terms not in the topic as received).

(c) The searcher made exploratory searches, examining the results, making tentative relevance judgements and perhaps using the semi-automatic query expansion facility (see section 3.3) to suggest new terms.

(d) Having decided on an initial formulation, the searcher then finished the exploratory session and started the definitive session.

(e) The definitive session involved two stages, an initial search and a first iteration feedback search. The initial search was strictly in accordance with the selected initial formulation; the searcher examined the top few documents, making relevance judgements.

(f) The first iteration feedback was purely automatic from the relevance judgements, including re-weighting and automatic expansion. No further iterations were conducted.

The guidelines to the searchers included the following:

Time: Searchers were asked to allow very roughly 30 minutes per topic. In fact, the average was nearer 50 minutes.

Feedback: The guidance was to assess about the first 20 documents retrieved by the initial search, or to stop after finding about 8 relevant (if that was sooner).

Relevance: If it seemed to be difficult to find any relevant items, searchers were encouraged to make generous relevance judgements, so as to ensure that there was some basis for feedback (see also section 6.2 below).

### 5.3.3 Remarks on the system

The bias in favour of initial formulation terms in the relevance feedback formula was 2 out of 3 (i.e. 3 supposed relevant documents out of which 2 were supposed to contain the term).

Searchers were able to use the Boolean facility described in section 3.2, for example to treat an expression such as (A and B) as if it were a single term, to be weighted like any other. However, the emphasis was on the usual (in the Okapi context) weighted searching of single terms, and this facility was used only occasionally, and only as part of larger best-match searches. In other words, this use did not compromise the characteristic of weighted searching as truly "best match", with all the flexibility that that implies.

### 5.3.4 Choice of terms

The terms chosen by the searchers may be briefly characterized by the following statistics:

Average number of terms 12.9
Terms appearing in the topic 10.5 (81%)
Terms appearing in different fields:
Description 3.4
Narrative 6.0
Concept 7.5
Others 2.9

(these add up to more than the total because a term may occur in more than one field).

For comparison, the Concept field has around 19-20 terms on average.

The results for the manual ad-hoc queries are given in the official tables as citym1 (without feedback) and citym2 (with feedback). For a discussion of the results, and of the evaluation method for citym2, see section 7.

## 6. Some observations on the experiments

### 6.1 Local relevance judgements

We experimented with making our own relevance judgements, based on the topics as provided. Although these experiments were on a very small scale and not very systematic, our impression was that it was usually possible to reproduce the judgements provided centrally, with a high chance of agreement. If this is so, it presumably reflects (a) the relatively highly specified nature of the topics (as compared to most IR queries!), and (b) the fact that the centrally-provided judgements are being made by experts other than the original requester. Thus we felt justified in attempting to improve our routing queries by providing some more relevance judgements of our own, particularly in cases where there were few centrally-provided ones. Note that the relevance weighting method used (F4 formula in section 2.2) takes account only of positive relevance judgements; items judged non-relevant are combined with items not judged (the complement method: Harper and van Rijsbergen, 1978).

However, as indicated in section 5.3, there were topics for which (under strict relevance criteria) the relevant documents were very sparse, and relevance feedback would not have had much effect. In these cases, for the manual searches only, searchers were encouraged to make more generous relevance judgements (i.e. to accept as relevant some documents that did not meet all the criteria precisely). The argument behind this guideline was that relevance feedback should work better given some partially-relevant items than with few or no relevant items. This argument obviously requires testing.

### 6.2 Bias to query terms

The bias in favour of original query terms discussed in section 3.4 was an attempt to represent the prior knowledge that a term chosen by the original requester or a searcher is likely to be good in terms of the probabilistic model. This argument relates to, but is not limited to, Harman's argument about negative weights (Harman, 1992). The point-5 formula used in the relevance weighting model actually has a built-in bias which might be described as "0.5 out of 1". The biases used in different TREC experiments (10 out of 10 and 2 out of 3) were chosen arbitrarily; unfortunately there was no time to do any extensive testing to enable a better-informed decision.

A bias such as 2 out of 3 has the curious effect of downgrading some very good query terms (any term that occurs in all the known relevant). This was part of the reason for trying the 10 out of 10 bias. However, there may be good reason for this effect: even very good results on the known relevant should not persuade us that $p$ is actually unity.

### 6.3 Two implementation errors

There were also two errors in the implementation of this bias. In the relevance weighting formula, the probability $p$ (that the term occurs in a relevant document) is estimated directly from the known relevant documents; the bias is correctly used to modify this estimate (e.g. $r \rightarrow r+2$ and $R \rightarrow R+3$ in the formula $p=r/R$). But the corresponding non-relevance probability $q$ is normally estimated by the complement method (i.e. all documents in the collection not known to be relevant are assumed to be non-relevant, $q=(n-r)/(N-R)$). In the implementation used for TREC, the modifications to $r$ and $R$ were incorrectly carried over to the $q$ estimate.

The second error occurred in the term selection value for query expansion. The full selection value should be $w(p-q)$. Since $q$ is normally very small compared to $p$, this can be approximated by $wp$. Since in a simple relevance feedback version, $p=r/R$ and $R$ is the same for all terms (i.e. the number of known relevant), ranking in $wp$ order is the same as ranking in $wr$ order. So in the TREC implementation, $wr$ was used. However, the modification to $R$ for query terms invalidates the second assumption (that $R$ is the same for all terms), so $wp$ should have been used.

These errors will have had the effect of over-emphasizing some infrequent query-terms, but will probably not have affected the overall results greatly.

## 7. Results and discussion

Full results can be seen in the official tables. The evaluation of the feedback run was treated in a somewhat special way, by agreement with the organizers. The original plan had been to do "residual ranking" evaluation, i.e. to remove from the collection those items which were assessed for relevance for feedback purposes, and to evaluate two runs (with or without feedback) on the reduced collection. This would have allowed a comparison between these two runs, but not between the feedback run and any of the other results presented.

Instead, a "frozen rank" evaluation was used, in which the documents examined for relevance before feedback were retained as the top-ranking documents in the feedback run. This simulates a real search, in that those documents would have been seen (in some form) by the user and would therefore have to be regarded as part of the output of the system. Therefore it may be seen as a fairer evaluation of feedback than residual ranking, although it is likely to reduce the apparent effect of feedback.

A very brief summary of the results, taking just two measures from the tables, is as follows:

| | 11-point average | Precision at 5 docs | |
|---|---|---|---|
| cityal | 12.1% | 49.6% | Ad-hoc auto |
| citym1 | 15.6% | 57.6% | Manual |
| citym2* | 18.2% | 58.8% | Feedback |
| cityr1 | 17.7% | 54.8% | Routing |

(*Frozen ranks evaluation)

The performance of the automatic ad-hoc run is really rather poor. The manual run without feedback is better. Feedback does clearly produce an improvement (though not, of course, given the frozen ranks evaluation, at the high-precision end). It seems that both the choice of terms and the liberal relevance judgements by non-expert students are effective at least to some degree. (We have yet to compare the individual judgements by the students with the "correct" ones provided by the TREC organizers, or to establish whether the "correct" judgements would have given us greater performance benefits.) The routing results seem reasonable.

In general, we believe that the simple, robust and minimum-effort methods we have adopted in Okapi have been shown to work, even with very different material (both documents and queries) from that for which Okapi was originally designed. Performance, both in absolute terms and relative to the other TREC entries, is respectable but by no means wonderful. We also believe that there is much scope for improvement; there are other simple and robust methods (such as other weighting formulae or different treatments of compound terms) to which Okapi would be hospitable, and which may bring performance up to a more acceptable level. We look forward to TREC 2.

## References


**Bookstein, A.** (1983). Information retrieval: a sequential learning process. Journal of the American Society for Information Science 34(5) 331-342.

**Hancock-Beaulieu M. & Walker S.** (1992). An evaluation of automatic query expansion in an online library catalogue. Journal of Documentation 48(4) 406-421.

**Harman, D.** (1992). Relevance feedback revisited. In: SIGIR 92 -- Proc. 15th International Conference on Research and Development in Information Retrieval, ACM Press, 1-10.

**Harper, D.J. & van Rijsbergen, C.J.** (1978). An evaluation of feedback in document retrieval using co-occurrence data. Journal of Documentation 34(3), 189-216.

**Porter, M.F.** (1980). An algorithm for suffix stripping. Program 14(3) 130-137

**Robertson, S.E.** (1990). On term selection for query expansion. Journal of Documentation 46(4), 359-364.

**Robertson, S.E. & Sparck Jones, K.** (1976). Relevance weighting of search terms. Journal of the American Society of Information Science 27(3), 129-146.

**Robertson, S.E., Thompson, C.L., Macaskill, M.J. & Bovey, J.** (1986). Weighting, ranking and relevance feedback in a front-end system. Journal of Information Science 12(1/2), 71-75.

**Walker, S.** (1989). The Okapi online catalogue research projects. In: The online catalogue: developments and directions, edited by Charles R Hildreth. Library Association. 84-106.

**Walker S. & De Vere R.** (1990). Improving subject retrieval in online catalogues: 2. Relevance feedback and query expansion. British Library (British Library Research Paper 72.) ISBN 0-7123-3219-7

**Walker S. & Hancock-Beaulieu M.** (1991). Okapi at City: an evaluation facility for interactive IR. British Library Research Report 6056.

## Appendix: System architecture

Platform

The system runs on Sun hardware. It should port fairly easily to other UNIX platforms, at least of the BSD type. All the search and indexing code is in C. Source file conversion programs and log analysis programs may be written in awk.

Database structure

A database consists of
- text file (bibliographic file)
  - this is the dataset from which searches retrieve records
- up to three indexes
  - Each index consists of primary and secondary dictionaries and a posting file. There are several types of index. One contains no positional information below the level of records, and is suitable for "phrases" like personal names and titles. Others contain positional information in the form field, sentence, word number for every occurrence of every indexed term. An index can contain terms for up to 16 different types of search.
- a set of parameter files:
  - database description parameter
  - indexing parameters (one set for each index).
    - These define how indexing is to be performed in terms of linguistic knowledge base, stemming function, procedure for extracting index terms and the fields and subfields from which they are to be extracted.
  - search type (or group) parameters
    - These are closely related to indexing parameters. They are used to determine the linguistic processing to be applied to queries and the parameters to be used for index lookup and for the extraction of terms for automatic query expansion
  - search mnemonics (e.g. TI, ABS, AUTH) used in query parsing
  - display parameters, defining two levels of display
- language knowledge bases
  - Up to three of these may be associated with a database to allow linguistic processing to depend on the type of data being searched or extracted. Typically, these are common to a number of databases of similar type and usage.

Input

Source files are stored in a simple format where each record starts with a field directory giving the length of each field, followed by the text of the fields. Fields may contain a limited range of subfield or role markers, indicating the nature of the following data. There are facilities for importing a few types of bibliographic files, including UKMARC and ISO 2709. An "Okapi exchange" format also exists. Character coding is ASCII with a "shift" character ('\') to allow the encoding of characters above hex 7F. No data compression is used.

Output (interactive Okapi only)

Output is to character-based terminals or windows, or hard copy. There are two levels of record display and printout -- brief (one line) and full. Record layout is determined by parameters and is fairly flexible.

Database maintenance

There are no record editing and no index updating facilities. Source file and indexes must be completely regenerated when necessary.

Indexing storage overheads

Several types of index are available. Depending on the nature of the database and the extent of indexing required overheads range from about 10% to 120% of the bibliographic file size.

Performance

Index lookup is fast because each lookup only requires one disk access. A multi-term search runs in time approximately proportional to the total number of postings for all the terms in the query.

Bibliographic record access is also fast because there is no indirection: the postings records directly address the bibliographic records, so again there is only one disk access per record.

File inversion is relatively slow and cpu-bound because of the multi-pass linguistic processing during index term extraction. As a rough guide, inversion runs at about one minute per megabyte of indexable text on a lightly loaded Sun 4/330.

Limits

Maximum bibliographic file size: 32 gigabytes
but maximum index size 4 gigabytes
Number of records per database: no practical limit
Postings per index term: no practical limit
Maximum amount of data which can be treated as a "record" for retrieval purposes: this is a system parameter usually set to 16 kilobytes. Up to 64 K or more is acceptable.
Maximum field length: same as record size
Maximum number of fields per record: 31
Maximum index term length: 127 characters
Maximum number of terms in single query: 32
(interactive Okapi only)

# Query Improvement In Information Retrieval Using Genetic Algorithms
## - A Report on the Experiments of the TREC Project

Jing-Jye Yang, Robert R. Korfhage
*Department of Information Science*
and
Edie Rasmussen
*Department of Library Science*

*School of Library and Information Science*
*University of Pittsburgh*

### Abstract

We have been developing an adaptive method using genetic algorithms to modify user queries, based on relevance judgments. This algorithm was adapted for the Text REtrieval Conference (TREC). The method is shown to be applicable to large text collections, where more relevant documents are presented to users in the genetic modification. The algorithm also shows some interesting phenomena, such as parallel searching. Further studies are planned to adjust the system parameters to improve its effectiveness.

## 1. Introduction

Information retrieval can be viewed as searching in a high-dimensional document space to bring relevant documents to users. It is known, however, that this is not an easy task. Due to the complexity of document writing styles and the difficulty users have in presenting their information requests, the retrieval results often frustrate users. Returning to the system with a modified query becomes unavoidable if a user wants to improve the retrieval results. However, most systems provide little or no guidance to the user for modifying the original query, adding to the frustration. Using relevance feedback to help users solve the problem has long being viewed as a promising agenda.

Relevance feedback, although started more than twenty years ago, attracted more research in recent years. With advances in computer hardware and software, the feedback process, which requires more computing time and friendlier user interface than the traditional method, will dominate the information retrieval area. However, feedback techniques also need to be improved. They should include more intelligent mechanisms which embed the ability to adapt to the changing environment and handle the feedback process automatically.

We have been developing an adaptive method using genetic algorithms to modify user queries automatically. Our preliminary experiments based on several simple data collections showed promising results (Yang and Korfhage, 1992a, 1992b). This algorithm was applied to the TREC project. This paper reports the experiments and some of the results.

## 2. The Model

Our system is based on a vector space model. Within the model, both documents and queries are represented as vectors. A document vector ($D_i$ ) with $n$ keywords can be represented as:

$$D_i = (t_{i1}, t_{i2}, t_{i3}, ....., t_{in})$$

where $t_{ij}$ ($j$ = 1, ...., $n$) is the assigned value of the $j^{th}$ keyword. Either weighted or unweighted document keywords can be used. For unweighted document keywords, $t_{ij}$ is either 0 or 1. For weighted document keywords, the value of $t_{ij}$ is a real number in the range [0,1].

In this model a query ($Q$) with $m$ keywords is also represented in a vector format as:

$$Q = (qt_1, qt_2, ....., qt_m)$$

where $qt_k$($k$= 1, ..., $m$) is the $k^{th}$ keyword in the query, where the keywords are extracted from the user natural language requests. However, no weights are assigned to query terms in the initial stage; query term weights are assigned by the system during the processing to be described later.

In general, under vector space models, document retrieval is based on a similarity measurement between the query and documents. This means that documents with high similarity to the query are judged more relevant to the query and should be retrieved first. In our model, the similarity measure is based on a metric, the $L_p$ metric, which measures the distance between document vectors and query vectors. This means that the shorter the distance between a document and the query, the higher the similarity between them. The distance value between a document $D_i$ and the query $Q$ is calculated, using the $L_p$ metric, by the formula:

$$DIS(D_i, Q) = \| D_i, Q \| = (\sum_k |t_{ik} - qt_k|^p)^{1/p}$$

where $k$, ($k$ = 1, ...., $m$), ranges over the set of query terms. Theoretically, the value of $p$ can be in the range [1, ∞ ]. In the current experiments, $p$ is set to 2. That is, the distance measure is the Euclidean distance. Note that only terms which are present in the query are used in the formula. This means if a term is present in the document but not in the query, the term is assumed of no concern or not known by the user. In other words, a document will be considered in the retrieval process only if it has at least one term in common with those in the query.

However, variations exist in the actual implementation. The keyterms in documents are unweighted in the TREC databases we have used. That is, the elements in the document vectors are in binary mode, 0 or 1. This was determined for two reasons. First, for a large database of this kind, obtaining document term weights using general term weighting methods, such as the inverse document frequency (e.g., Salton and Buckley, 1988) requires counting term frequencies from all of the documents in the database, which is resource intensive for a database of this size. Second, previous experiments using the Cranfield database indicate that our system performance equally well with either weighted or unweighted document terms.

Each query vector is generated from a topic provided in the TREC project. In general, a topic includes several descriptive items: the sequential number of the topic, domain, title, descriptive, narrative, concepts, factors, nationality, etc. The terms of a query vector are derived from the title and the concepts of the topic. In some cases the nationality is added, if it is important to meet the requirement of the narrative. The query vector term weights are assigned by the system at the beginning of each run.

## 3. The Algorithm

The method we developed is based on the relevance feedback concept. However, the algorithm underlying the process is totally different from those used in other relevance feedback studies. Our system is also based on genetic algorithms which are known for their efficiency and effectiveness in searching large problem spaces to find optimal solutions (e.g., Holland, 1975; Goldberg, 1989).

The basic concept behind genetic algorithms is that there exists a population of individual variants (those individuals can be various types, depending on the actual problem to be solved) in an environment, each trying to survive by exchanging information and competing with others. The genetic algorithms use methods analogous to natural evolution mechanisms to generate those individuals who provide the optimal or near optimal solution to the problem. The adaptability of genetic algorithms is due to the fact that individual selection and manipulation are adaptive to the existing environment. The environment evaluates the fitness

of the individuals, and the results form the feedback from the environment to the system where the genetic process is applied. The efficiency of the algorithm is due to parallel searching, with a set of individuals exploring the problem space simultaneously. Information of better structures is stored, exchanged and transferred from generation to generation, and combined to make individuals having all or most of the better features of their predecessors.

A few researchers have applied genetic algorithms to information retrieval. Gordon (1988) used a genetic algorithm to modify document descriptions to facilitate the retrieval of relevant documents for a group of users who are interested in similar topics. Frieder and Siegelmann (1991) developed a genetic algorithm to obtain an optimal allocation of documents onto nodes in distributed multiprocessors.

In vector space models document retrieval can be seen as searching a very high-dimensional document space to find the area(s) where most of the relevant documents to a user query are located. We have been developing a genetic algorithm in document retrieval through the modification of query term weights. The steps of the genetic query modification processes can be described as follows.

(1) Generate query variants:

As described above, the genetic algorithms are applied to situations where there is a group of individuals. To apply the techniques, there needs to be a set of query individuals. They are generated in this step. Each query individual vector has the same elements as in the original query (user's query). Each element within a query vector represents a weight corresponding to the related term in the original query. The weights are assigned randomly using a random number generator, with the values of the weights scaled to the range [0,1]. The number of query variants is set in the beginning of the experiments and remains fixed in later processes of each run.

(2) Individual query performance evaluation:

This process evaluates the performance of each individual query variant. First, the distances between query variants and documents are calculated, using the formula mentioned in Section 2. Second, for each query individual, those documents whose distance values to this query individual are less than a given threshold are viewed as relevant to the user request and are retrieved. Finally, the retrieval performance ($P$) of each query individual is assessed using the formula:

$$P = A - B - C$$

where $A$ is the number of retrieved documents which are judged as relevant to the user request; $B$ is the number of retrieved documents which are not relevant; and $C$ is the number of relevant documents that are not retrieved by the query individual. The performance values for the query variants are to be used in the genetic modification. (Note: the $P$ value also can be in the form: $P = A - B$, in case the $C$ value is unknown which is normal in the actual online retrieval process. Note also that $P$ is the internal value used by the system and is unknown to the user.)

(3) The genetic process:

After the evaluation process genetic operators are used to modify the structures of the query variants, with the aim of improving the retrieval effectiveness of the variants.

(a) Selection: In this step those query individuals whose performances are greater than the average performance of all the query variants are selected and retained for further processing. Other query individuals are discarded.

(b) Reproduction: In this process the survivors from the previous step are reproduced. In our algorithm, the reproduction algorithm developed by Baker (1987) is used. The number of offspring for each individual depends on the ratio of its performance to the total performance of all the survivors and the population size. Therefore, the individual survivor with the highest performance will have the most offspring. Also, in this step, the query variants are rearranged randomly in a sequence for later processing. We call the set of query variants after reproduction the *semi-new* generation.

(c) Mutation: In the mutation step, some individuals are selected at random from the *semi-new* generation, and some of their genes (i.e., the term weights) are selected and assigned new values, also in the range [0,1]. The number of individuals to be selected depends on the mutation ratio chosen in the experimentation. The selection of genes and the new value assignments are also done randomly.

(d) Crossover: The crossover process mates pairs of query individuals in the *semi-new* generation and exchanges parts of their term weights. Here, the two-point crossover method is used. For each pair, two positions in the term vectors are selected randomly, and all term weights between the two positions are exchanged for the two individuals. Thus, two new individuals are born. However, it may be the case that not all individuals are chosen to mate with others. The crossover ratio controls the number of mates.

After the crossover operation, a totally new generation is created and the process goes back to step 2 and continues until a preset condition is satisfied, either a fixed number of generations has been processed or no more relevant documents are retrieved.

The genetic retrieval algorithm is distinguished from other research using relevance feedback: (1) Several query individual variants exist and explore the document space simultaneously. (2) To modify the term weights, it is not required to calculate and analyze the document term weights, at least in the current study. Query term weight modification is done by the genetic mechanisms through the interchange of term weights between query individuals and the introduction of new weights by the mutation operation. (3) Only term weights are modified and no new terms are introduced and no terms are deleted. The importance of a term is shown by the value of its weight.

## 4. Preprocessing of Documents and Queries

As in most information retrieval systems, the documents and topics in the TREC project need to be preprocessed before they can be used by our algorithm.

(1) Document processing:

Document retrieval in our system is based on keyword match. In current system, a keyword[1] is a single word. For the documents, to facilitate the retrieval we need to create inverted files. The processes for creating the inverted files can be described as follows.

(a) Keyword extraction:

Keywords are extracted for each document. The stopwords (total 2,529 stopwords on the list) are deleted, and keywords are stemmed using the Porter stemming algorithm which was implemented by Fox (Frakes, 1992). The document number from which a keyword is extracted is also stored with that keyword, for it will be used to create inverted files and later to facilitate the retrieval of the full text of the document. Also an "address file" is generated which indicates for each document the offset of its location related to the original file within which the document is stored.

(b) Creation of inverted files:

An inverted file is created for each database to facilitate the retrieval process. The file is organized as: a keyword followed by a list of document numbers from which the keyword extracted. Three-level indexed files are generated for the inverted file to reduce the search time. Each index file include keywords, their offsets in the inverted file and the offset in the immediately higher level file (for the second and third levels).

(2) Query processing:

Queries are generated from the topics provided on the TREC project. For each topic a single query vector (the *original query*) is generated, which consists of a list of keywords from the title and the concepts of the topic. For some queries information about the nationality is also included if it is necessary to satisfy the requests from the narrative descriptions of the related topics. The stemming algorithm is also applied to the terms in the query. For the training queries we did not add any terms from other descriptive items on the topics. But for some ad hoc queries, we added several keywords from the narrative because we thought that those keywords would be useful to identify relevant documents. The routing queries are those query individuals from the last generation of the training queries.

Ten query individual vectors were generated for each *original query*. On the training queries the initial query term weights for the query individuals were assigned randomly, and then modified by the genetic algorithm. The final query individual vectors from the training set were used as routing queries with no weights being modified.

---

[1]Keyword, keyterm and term will be used interchangeably in this paper.

The assignment of the term weights to the ad hoc queries is different. Only one query vector (called *query individual one*) is generated for each ad hoc query. The term weights in *query individual one* were assigned either by using the weights of the same terms in the final generation of the training topics if they existed, or by the researchers by referring to the weights of the relative terms in the final generation of the training topics or to their importance related to the topic. Moreover, in the ad hoc queries, a term weight of the *query individual one* could be assigned different values depending on which database the query is used to against documents. Based on the *query individual one*, another nine query individuals were generated with the weights of each term in the nine individuals being normally distributed around the corresponding term weights on the *query individual one*.

One interesting question in query processing is how to handle the situation where a concept with a NOT requirement is presented in a user's request (i.e., the topics in TREC). It is hard to deal with in some retrieval systems. However, in the TREC topics, there are several cases which include NOT in the concepts and the narrative items. Our solution to this problem is to assign negative weight to a keyword if it is described by the NOT concept or narrative item. Since our system is based on a distance measure, a negative assignment can cause the documents which include this keyword to have longer distance from the query than those without the keyword.

## 5. System Configuration

Some features of our system on the TREC project are described in this section.

(1) Window size for document retrieval

In much feedback retrieval research, there is a fixed window size which decides the number of documents to be retrieved. The value usually was set between 5 and 20 (e.g., Salton, 1971). Aalbersberg (1992) used window size one in the feedback retrieval process and showed that the precision values for four standard databases, on average, are higher than those achieved with size greater than one. However, Aalbersberg also suggested that variable relevance feedback (Ide, 1971) is worth implementing in term weighting IR systems.

In our original experiments on small databases a threshold value was generated for each query in the beginning of an experiment, as the criterion for determining the retrieval window. If the distance measure between a document and a query individual was less than the threshold value, the document was viewed as relevant to the query and was retrieved. Thus we had a variable window size. However, in dealing with the large databases we added another factor to decide the window size. As in the previous method, a threshold value is generated first as each original query is read into the system. Documents whose distance is less than the threshold will be regarded as relevant to the query and will be printed for evaluation. However, if more than forty documents satisfy the condition, only the top forty are retrieved for evaluation. If there is a tie, the selection is random among these tied. This number seems reasonable given the time constraints and focus of the study.

(2) User involvement and relevance judgment

Since our system uses relevance feedback (on the training queries and the ad hoc queries), users (evaluators) are needed to evaluate the retrieved documents. The evaluators are our researchers in the TREC project. The document evaluation process runs like this. On each iteration, for each query individual the retrieved documents are printed with both their full text (or abstracts or summaries if available) and document numbers. An evaluator examines the retrieved documents based on his/her reading of the text (or abstract or summaries) and understanding of the topic, and marks the document numbers which are judged relevant to the topic.

The results of the judgment from the evaluators are fed back to the system. The system then applies the genetic modification process (as described in Section 3) based on the feedback information.

(3) The feedback process

The feedback process can be described as follows. The system uses the relevance judgment to calculate the performance ($P$) of each query individual based on the function described in Section 3.2. However, the parameters in the formula were changed as follows. First, there are no known relevant documents for each query, the value $C$ is deleted. Second, in our initial tries on a few training topics we found that for these very large databases many nonrelevant documents are retrieved in the first generation. This would cause the individuals which retrieved few relevant and many nonrelevant documents to get lower $P$ values than those which retrieved no documents or only a few nonrelevant documents. So instead of using equal weight for the values of $A$ and $B$, we gave more weight to $A$. The formula became: $P = 10*A - B$. This increases the chance of survival for those individuals which retrieve a few relevant documents. The weight 10 was chosen for the parameter $A$ because in several initial tries (weight equal to 1, 5 and 10), 10 brought more relevant documents in the modified generation than the other weights. However, values higher than 10 were not used to avoid the query variants to converging too quickly so that the final query might be a local optimum.

After computing the performance for each variant, genetic operations are applied to generate a set of new query individuals. Those new query individuals are then used by the system to retrieve documents. However, those documents which have been retrieved in previous generations are not presented in the new generation, even if they satisfy the selection condition. This strategy is similar to the residual collection method used by Chang et. al. (1971).

The feedback process continues until either no more relevant documents are retrieved for any query variants or the user (the evaluator) decides to stop the process, mainly due to the time constraints.

## 6. Statistical Data

(1) Computing time:

Time is given for two types of processing.

(a) Time for document processing:

Document processing time includes keyword extraction, sorting and merge, and building inverted files. Two systems were used, a SUN -670 and SUN SPARC/IPC(s), depending on availability. Table 1 provides the time used on several stages in document processing.

(b) Time for document retrieval per generation

The time required to retrieve documents for each iteration (generation) depends on several factors: the number of terms in a query, the generation, and the computing system used. Table 2 gives the average retrieval time for three generations, based on topics 1-50 and the first dataset (disk one).

(2) Storage space for data structures:

Several types of data structure were created to facilitate retrieval. They are inverted files, indexed files and address files which consist of document numbers and their locations in the databases. The storage space used for these files (disk one only) is shown in Table 3.

(3) Machine capability:

The capabilities of our systems are:
SUN-670: 32 Megabytes RAM and 40 MHz CPU clock rate;
SUN SPARC/IPC: 24 Megabytes RAM and 25 MHz CPU clock rate.

(4) Manpower:

There are in total four persons participating in the project, two Ph.D. students and two faculty. One Ph.D. student worked full time on this project, and the other part time, one faculty member half time and the other part time.

Table 1. Time for document processing (Unit: min.)

| | DOE | AP | | ZIFF | | WSJ | | FR | |
|---|---|---|---|---|---|---|---|---|---|
| | 1'st Dataset | 1'st Dataset | 2'nd Dataset | 1'st Dataset | 2'nd Dataset | 1'st Dataset | 2'nd Dataset | 1'st Dataset | 2'nd Dataset |
| Keyword extraction | 129 | 148 | 169 | 130 | 43 | 182 | 125 | 19 | 16 |
| Sorting and merge | 293 | 281 | NA | 248 | 72 | 404 | 398 | 18 | 11 |
| Create inverted files | 29 | NA | 27 | 28 | 11 | NA | 34 | 2 | 3 |

* NA: Not Available

Table 2. Average retrieval time for topics 1-50 on the first dataset
(Unit: minutes)

| Generation 0 | Generation 1 | Generation 2 |
|---|---|---|
| 3:56 | 2:19 | 2:05 |

Table 3. Total amount of storage for inverted and indexed files -- disk one only

(Unit: Megabytes)

| | DOE | AP | ZIFF | WSJ |
|---|---|---|---|---|
| Inverted files | 162.3 | 199.8 | 143.7 | 223.4 |
| Indexed files | 3.0 | 2.2 | 2.4 | 2.1 |
| Address files | 4.3 | 1.7 | 1.7 | 2.6 |

## 7. Results

This section describes several results of using the genetic algorithm in the TREC document collection. Examples provided are from training queries (topic 1 to 50) on the DOE database.

(1) Query convergence

In large document collections like the TREC databases, the genetic algorithm caused the query variants to converge within 3 to 6 generations in most cases. For an example, Table 4 shows the term weights of query individuals in the first generation, 0, and last generation, 5, on topic 3. For most of the query terms the weights on the query individuals converged to a single value in the final generation. Although a few variations existed, they were caused by the mutation operation. Table 5 shows a similar situation for topic 12, where the final generation is 4.

An interesting phenomenon is how the query term weights changed. The genetic operators select the query individuals which have higher performance values than the average performance of all the individuals and exchange parts of their term weights by using mutation and crossover. Although the two operations are random, the results are interesting. The

experiments show that from generation to generation the average weight of each term in the query individuals gradually moves to the value in the final converged generation, although small variations may exist. As an example, Table 6 shows the changes of the average term weights on topic 3, from generation 0 to generation 5. The values on generation 5 are almost identical with those in Table 4.

(2) Effects of query term weight modification

The goal of query convergence using the genetic algorithm is to find the query individual with highest performance, that is, retrieving more relevant documents than its predecessors. Evidence from the experiments has shown the GA works as expected. In most cases, new relevant documents were brought in to the user in each generation, until convergence at the final generation. Table 7 through Table 11 show the numbers of new relevant documents retrieved in each generation for the five databases.

Observing the results, one interesting phenomenon arises; that is, for a topic the algorithm may retrieve different numbers of relevant documents on distinct databases. It seems reasonable since the five databases may concentrate on different areas. Moreover, the retrieval patterns for WSJ and AP databases, which should be interesting in the same topic, looks similar with each other.

Table 4 Query individuals on Topic 3 (11 terms)

Generation = 0

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.18 | 0.31 | 0.53 | 0.95 | 0.17 | 0.70 | 0.23 | 0.49 | 0.12 | 0.08 | 0.39 |
| 1 | 0.28 | 0.37 | 0.98 | 0.54 | 0.77 | 0.65 | 0.77 | 0.78 | 0.82 | 0.15 | 0.63 |
| 2 | 0.31 | 0.35 | 0.92 | 0.52 | 0.40 | 0.61 | 0.79 | 0.93 | 0.87 | 0.87 | 0.67 |
| 3 | 0.76 | 0.58 | 0.39 | 0.36 | 0.20 | 0.83 | 0.42 | 0.46 | 0.98 | 0.13 | 0.21 |
| 4 | 0.96 | 0.74 | 0.41 | 0.78 | 0.76 | 0.96 | 0.03 | 0.32 | 0.76 | 0.24 | 0.59 |
| 5 | 0.04 | 0.96 | 0.32 | 0.06 | 0.44 | 0.92 | 0.57 | 0.12 | 0.57 | 0.25 | 0.50 |
| 6 | 0.24 | 0.48 | 0.41 | 0.87 | 0.43 | 0.36 | 0.38 | 0.04 | 0.16 | 0.52 | 0.70 |
| 7 | 0.10 | 0.40 | 0.77 | 0.24 | 0.34 | 0.23 | 0.30 | 0.30 | 0.89 | 0.04 | 0.65 |
| 8 | 0.40 | 0.68 | 0.73 | 0.94 | 0.23 | 0.84 | 0.97 | 0.78 | 0.43 | 0.67 | 0.81 |
| 9 | 0.16 | 0.28 | 0.14 | 0.86 | 0.75 | 0.21 | 0.14 | 0.29 | 0.80 | 0.22 | 0.56 |

Generation = 5

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.28 | 0.37 | 0.98 | 0.54 | 0.77 | 0.76 | 0.03 | 0.45 | 0.69 | 0.24 | 0.78 |
| 1 | 0.28 | 0.22 | 0.98 | 0.54 | 0.77 | 0.56 | 0.03 | 0.32 | 0.76 | 0.24 | 0.58 |
| 2 | 0.28 | 0.22 | 0.98 | 0.54 | 0.77 | 0.66 | 0.03 | 0.45 | 0.69 | 0.24 | 0.78 |
| 3 | 0.28 | 0.22 | 0.98 | 0.54 | 0.77 | 0.66 | 0.03 | 0.45 | 0.69 | 0.24 | 0.78 |
| 4 | 0.28 | 0.22 | 0.98 | 0.54 | 0.77 | 0.66 | 0.03 | 0.45 | 0.69 | 0.24 | 0.78 |
| 5 | 0.28 | 0.22 | 0.98 | 0.54 | 0.77 | 0.66 | 0.03 | 0.45 | 0.69 | 0.24 | 0.78 |
| 6 | 0.28 | 0.42 | 0.98 | 0.54 | 0.77 | 0.66 | 0.13 | 0.45 | 0.69 | 0.24 | 0.78 |
| 7 | 0.28 | 0.22 | 0.98 | 0.54 | 0.77 | 0.66 | 0.03 | 0.32 | 0.76 | 0.24 | 0.58 |
| 8 | 0.28 | 0.37 | 0.98 | 0.54 | 0.77 | 0.67 | 0.13 | 0.32 | 0.76 | 0.24 | 0.55 |
| 9 | 0.28 | 0.37 | 0.98 | 0.54 | 0.77 | 0.66 | 0.03 | 0.32 | 0.76 | 0.24 | 0.58 |

Table 5 Query individuals on Topic 12 (12 terms)

Generation = 0

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.18 | 0.31 | 0.53 | 0.95 | 0.17 | 0.70 | 0.23 | 0.49 | 0.12 | 0.08 | 0.39 | 0.28 |
| 1 | 0.37 | 0.98 | 0.54 | 0.77 | 0.65 | 0.77 | 0.78 | 0.82 | 0.15 | 0.63 | 0.31 | 0.35 |
| 2 | 0.92 | 0.52 | 0.40 | 0.61 | 0.79 | 0.93 | 0.87 | 0.87 | 0.67 | 0.76 | 0.58 | 0.39 |
| 3 | 0.36 | 0.20 | 0.83 | 0.42 | 0.46 | 0.98 | 0.13 | 0.21 | 0.96 | 0.74 | 0.41 | 0.78 |
| 4 | 0.76 | 0.96 | 0.03 | 0.32 | 0.76 | 0.24 | 0.59 | 0.04 | 0.96 | 0.32 | 0.06 | 0.44 |
| 5 | 0.92 | 0.57 | 0.12 | 0.57 | 0.25 | 0.50 | 0.24 | 0.48 | 0.41 | 0.87 | 0.43 | 0.36 |
| 6 | 0.38 | 0.04 | 0.16 | 0.52 | 0.70 | 0.10 | 0.40 | 0.77 | 0.24 | 0.34 | 0.23 | 0.30 |
| 7 | 0.30 | 0.89 | 0.04 | 0.65 | 0.40 | 0.68 | 0.73 | 0.94 | 0.23 | 0.84 | 0.97 | 0.78 |
| 8 | 0.43 | 0.67 | 0.81 | 0.16 | 0.28 | 0.14 | 0.86 | 0.75 | 0.21 | 0.14 | 0.29 | 0.80 |
| 9 | 0.22 | 0.56 | 0.72 | 0.20 | 0.99 | 0.25 | 0.43 | 0.76 | 0.86 | 0.89 | 0.98 | 0.40 |

Generation = 4

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.92 | 0.57 | 0.12 | 0.67 | 0.25 | 0.50 | 0.28 | 0.28 | 0.96 | 0.84 | 0.41 | 0.60 |
| 1 | 0.92 | 0.57 | 0.12 | 0.57 | 0.25 | 0.50 | 0.07 | 0.15 | 0.96 | 0.74 | 0.41 | 0.40 |
| 2 | 0.92 | 0.57 | 0.12 | 0.47 | 0.25 | 0.50 | 0.09 | 0.35 | 0.96 | 0.98 | 0.43 | 0.36 |
| 3 | 0.92 | 0.57 | 0.12 | 0.57 | 0.25 | 0.50 | 0.09 | 0.15 | 0.96 | 0.77 | 0.41 | 0.47 |
| 4 | 0.92 | 0.57 | 0.12 | 0.57 | 0.25 | 0.50 | 0.09 | 0.21 | 0.96 | 0.74 | 0.41 | 0.47 |
| 5 | 0.92 | 0.57 | 0.12 | 0.57 | 0.25 | 0.50 | 0.09 | 0.28 | 0.96 | 0.84 | 0.41 | 0.47 |
| 6 | 0.92 | 0.57 | 0.12 | 0.72 | 0.25 | 0.50 | 0.26 | 0.31 | 0.96 | 0.84 | 0.41 | 0.47 |
| 7 | 0.92 | 0.57 | 0.12 | 0.52 | 0.25 | 0.50 | 0.09 | 0.12 | 0.96 | 0.74 | 0.41 | 0.52 |
| 8 | 0.92 | 0.57 | 0.12 | 0.67 | 0.25 | 0.50 | 0.26 | 0.15 | 0.96 | 0.74 | 0.41 | 0.52 |
| 9 | 0.92 | 0.57 | 0.12 | 0.67 | 0.25 | 0.50 | 0.26 | 0.15 | 0.96 | 0.74 | 0.41 | 0.52 |

Table 6 Average query term weights on each generation for topic 3

| Gen. | t1 | t2 | t3 | t4 | t5 | t6 | t7 | t8 | t9 | t10 | t11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | .343 | .515 | .560 | .612 | .449 | .631 | .460 | .451 | .640 | .317 | .571 |
| 1 | .514 | .493 | .747 | .688 | .585 | .772 | .592 | .672 | .734 | .426 | .671 |
| 2 | .396 | .386 | .881 | .552 | .510 | .787 | .315 | .516 | .763 | .348 | .684 |
| 3 | .301 | .349 | .938 | .526 | .511 | .811 | .262 | .444 | .785 | .213 | .700 |
| 4 | .286 | .342 | .968 | .556 | .696 | .746 | .060 | .346 | .746 | .240 | .649 |
| 5 | .280 | .285 | .980 | .540 | .770 | .661 | .050 | .398 | .718 | .240 | .700 |

Table 7. # of relevant document retrieved - DOE database

| Topic | Generation | | | | | |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 |
| 01 | 0 | | | | | |
| 02 | 0 | | | | | |
| 03 | 5 | 8 | 5 | 9 | 2 | |
| 04 | 0 | | | | | |
| 05 | 0 | | | | | |
| 06 | 0 | | | | | |
| 07 | 0 | | | | | |
| 08 | 0 | | | | | |
| 09 | 0 | | | | | |
| 10 | 2 | 0 | 1 | 1 | | |
| 11 | 4 | 3 | | | | |
| 12 | 29 | 15 | 1 | 2 | | |
| 13 | 23 | | | | | |
| 14 | 0 | | | | | |
| 15 | 0 | | | | | |
| 16 | 0 | | | | | |
| 17 | 0 | | | | | |
| 18 | 0 | | | | | |
| 19 | 0 | | | | | |
| 20 | 0 | | | | | |
| 21 | 27 | 5 | | | | |
| 22 | 0 | | | | | |
| 23 | 0 | | | | | |
| 24 | 36 | 15 | 5 | 2 | | |
| 25 | 10 | 0 | 1 | 10 | 1 | |
| 26 | 6 | 0 | 5 | | | |
| 27 | 58 | 22 | 15 | 4 | 22 | |
| 28 | 1 | | | | | |
| 29 | 0 | | | | | |
| 30 | 0 | | | | | |
| 31 | 0 | | | | | |
| 32 | 0 | | | | | |
| 33 | 10 | 3 | 10 | | | |
| 34 | 0 | | | | | |
| 35 | 0 | | | | | |
| 36 | 0 | | | | | |
| 37 | 0 | | | | | |
| 38 | 3 | | | | | |
| 39 | 14 | | | | | |
| 40 | 2 | 2 | | | | |
| 41 | 0 | | | | | |
| 42 | 3 | 1 | 1 | | | |
| 43 | 4 | | | | | |
| 44 | 0 | | | | | |
| 45 | 3 | 2 | 0 | 1 | | |
| 46 | 0 | 1 | | | | |
| 47 | 0 | | | | | |
| 48 | 4 | | | | | |
| 49 | 9 | | | | | |
| 50 | 0 | | | | | |

Table 8. # of relevant document retrieved - FR database

| Topic | Generation | | | | | |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 |
| 01 | 0 | | | | | |
| 02 | 0 | 1 | | | | |
| 03 | 0 | | | | | |
| 04 | 0 | | | | | |
| 05 | 1 | | | | | |
| 06 | 0 | | | | | |
| 07 | 0 | | | | | |
| 08 | 0 | | | | | |
| 09 | 0 | | | | | |
| 10 | 0 | 1 | | | | |
| 11 | 1 | 1 | | | | |
| 12 | 2 | | | | | |
| 13 | 0 | | | | | |
| 14 | 0 | | | | | |
| 15 | 0 | | | | | |
| 16 | 0 | | | | | |
| 17 | 19 | 18 | 6 | 1 | | |
| 18 | 0 | | | | | |
| 19 | 0 | | | | | |
| 20 | 0 | | | | | |
| 21 | 0 | | | | | |
| 22 | 0 | 1 | 2 | | | |
| 23 | 0 | | | | | |
| 24 | 0 | | | | | |
| 25 | 0 | | | | | |
| 26 | 0 | | | | | |
| 27 | 0 | | | | | |
| 28 | 0 | | | | | |
| 29 | 0 | | | | | |
| 30 | 0 | | | | | |
| 31 | 0 | | | | | |
| 32 | 0 | | | | | |
| 33 | 0 | | | | | |
| 34 | 0 | | | | | |
| 35 | 0 | | | | | |
| 36 | 0 | | | | | |
| 37 | 0 | | | | | |
| 38 | 0 | | | | | |
| 39 | 0 | | | | | |
| 40 | 0 | | | | | |
| 41 | 0 | | | | | |
| 42 | 0 | | | | | |
| 43 | 0 | | | | | |
| 44 | 0 | | | | | |
| 45 | 0 | | | | | |
| 46 | 0 | | | | | |
| 47 | 0 | | | | | |
| 48 | 0 | | | | | |
| 49 | 0 | | | | | |
| 50 | 0 | | | | | |

Table 9. # of relevant document retrieved - AP database

| Topic | Generation | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 01 | 12 | 6 | 19 | 5 | | | |
| 02 | 12 | 11 | 2 | | | | |
| 03 | 17 | | | | | | |
| 04 | 5 | 7 | 7 | | | | |
| 05 | 6 | 0 | 2 | 1 | | | |
| 06 | 11 | 14 | 14 | 8 | 3 | 7 | 26* |
| 07 | 5 | 4 | 0 | 2 | | | |
| 08 | 0 | 3 | | | | | |
| 09 | 14 | 2 | 1 | | | | |
| 10 | 30 | 18 | 11 | 3 | 1 | | |
| 11 | 16 | 23 | 8 | 20 | 14 | 2 | 1 |
| 12 | 45 | 38 | 3 | 1 | | | |
| 13 | 67 | 6 | | | | | |
| 14 | 0 | | | | | | |
| 15 | 12 | | | | | | |
| 16 | 0 | | | | | | |
| 17 | 3 | | | | | | |
| 18 | 0 | 4 | 8 | 10 | 21 | 2 | 5* |
| 19 | 28 | 35 | 62 | 74 | 88 | 29 | 48* |
| 20 | 0 | 1 | 4 | 3 | 2 | | |
| 21 | 3 | 1 | | | | | |
| 22 | 65 | 15 | 24 | 7 | 3 | 4 | 5* |
| 23 | 1 | 2 | | | | | |
| 24 | 16 | 14 | 10 | 2 | | | |
| 25 | 0 | | | | | | |
| 26 | 0 | 1 | 2 | | | | |
| 27 | 0 | | | | | | |
| 28 | 3 | 1 | | | | | |
| 29 | 0 | | | | | | |
| 30 | 0 | | | | | | |
| 31 | 0 | | | | | | |
| 32 | 0 | | | | | | |
| 32 | 0 | | | | | | |
| 33 | 0 | 1 | 1 | | | | |
| 34 | 0 | | | | | | |
| 35 | 0 | 1 | | | | | |
| 36 | 0 | | | | | | |
| 37 | 2 | | | | | | |
| 38 | 0 | | | | | | |
| 39 | 0 | | | | | | |
| 40 | 0 | | | | | | |
| 41 | 0 | | | | | | |
| 42 | 2 | 1 | | | | | |
| 43 | 0 | | | | | | |
| 44 | 7 | 5 | | | | | |
| 45 | 0 | | | | | | |
| 46 | 0 | 19 | | | | | |
| 47 | 8 | | | | | | |
| 48 | 1 | 8 | | | | | |
| 49 | 12 | | | | | | |
| 50 | 0 | | | | | | |

* means more relevant documents were retrieved after this generation, but the document number has been added into this generation.

Table 10. # of relevant document retrieved - WSJ database

| Topic | Generation | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 01 | 15 | 10 | 4 | 8 | 1 | | |
| 02 | 24 | 9 | 1 | | | | |
| 03 | 30 | 5 | | | | | |
| 04 | 11 | 9 | 7 | 8 | 2 | | |
| 05 | 3 | | | | | | |
| 06 | 26 | 2 | 5 | 5 | 5 | 7 | 1 |
| 07 | 4 | 5 | | | | | |
| 08 | 15 | 11 | 8 | 1 | | | |
| 09 | 22 | 4 | 5 | 2 | | | |
| 10 | 43 | 24 | 6 | 5 | 3 | | |
| 11 | 10 | 3 | 2 | | | | |
| 12 | 26 | 2 | | | | | |
| 13 | 29 | | | | | | |
| 14 | 2 | | | | | | |
| 15 | 45 | | | | | | |
| 16 | 0 | | | | | | |
| 17 | 2 | | | | | | |
| 18 | 31 | 16 | 8 | 12 | 8 | 2 | 22* |
| 19 | 50 | 50 | 30 | 5 | 31 | 23 | 32* |
| 20 | 1 | 6 | 10 | 13 | 7 | 3 | 1 |
| 21 | 18 | 7 | 2 | 3 | | | |
| 22 | 21 | 9 | | | | | |
| 23 | 2 | 3 | 7 | 1 | | | |
| 24 | 28 | 25 | 13 | 2 | 7 | 9 | |
| 25 | 1 | 1 | | | | | |
| 26 | 6 | 3 | | | | | |
| 27 | 1 | 1 | 1 | | | | |
| 28 | 6 | | | | | | |
| 29 | 1 | | | | | | |
| 30 | 20 | | | | | | |
| 31 | 3 | 4 | | | | | |
| 32 | 0 | | | | | | |
| 33 | 6 | | | | | | |
| 34 | 3 | | | | | | |
| 35 | 2 | 2 | | | | | |
| 36 | 1 | | | | | | |
| 37 | 0 | | | | | | |
| 38 | 1 | | | | | | |
| 39 | 0 | | | | | | |
| 40 | 1 | | | | | | |
| 41 | 0 | | | | | | |
| 42 | 0 | | | | | | |
| 43 | 0 | | | | | | |
| 44 | 0 | | | | | | |
| 45 | 0 | | | | | | |
| 46 | 1 | 14 | | | | | |
| 47 | 22 | | | | | | |
| 48 | 0 | | | | | | |
| 49 | 23 | 20 | 19 | 1 | | | |
| 50 | 0 | | | | | | |

Table 11. # of relevant document retrieved - ZIFF database

| Topic | Generation | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 01 | 3 | 1 | 2 | | | | |
| 02 | 10 | 1 | 0 | 1 | | | |
| 03 | 24 | 10 | 7 | 6 | | | |
| 04 | 0 | | | | | | |
| 05 | 2 | | | | | | |
| 06 | 0 | | | | | | |
| 07 | 1 | | | | | | |
| 08 | 0 | | | | | | |
| 09 | 0 | | | | | | |
| 10 | 0 | | | | | | |
| 11 | 3 | | | | | | |
| 12 | 0 | | | | | | |
| 13 | 0 | | | | | | |
| 14 | 0 | | | | | | |
| 15 | 47 | 3 | 2 | | | | |
| 16 | 0 | | | | | | |
| 17 | 0 | | | | | | |
| 18 | 0 | | | | | | |
| 19 | 0 | | | | | | |
| 20 | 18 | 11 | 15 | | | | |
| 21 | 2 | 1 | | | | | |
| 22 | 4 | | | | | | |
| 23 | 0 | | | | | | |
| 24 | 0 | 4 | | | | | |
| 25 | 0 | | | | | | |
| 26 | 73 | 75 | 10 | 3 | 5 | 5 | |
| 27 | 28 | 27 | 17 | 1 | | | |
| 28 | 34 | 29 | 8 | | | | |
| 29 | 9 | 4 | 1 | | | | |
| 30 | 8 | 0 | 2 | 1 | | | |
| 31 | 11 | 5 | 1 | | | | |
| 32 | 18 | 4 | | | | | |
| 33 | 13 | 20 | 13 | 15 | 7 | 6 | 9* |
| 34 | 24 | 29 | 7 | 2 | | | |
| 35 | 2 | | | | | | |
| 36 | 3 | | | | | | |
| 37 | 59 | 38 | 1 | | | | |
| 38 | 6 | 4 | 3 | 7 | 14 | 2 | |
| 39 | 6 | | | | | | |
| 40 | 3 | | | | | | |
| 41 | 15 | | | | | | |
| 42 | 12 | | | | | | |
| 43 | 2 | | | | | | |
| 44 | 43 | | | | | | |
| 45 | 10 | 6 | 1 | | | | |
| 46 | 14 | 20 | | | | | |
| 47 | 26 | 13 | 14 | 2 | | | |
| 48 | 1 | 0 | 5 | | | | |
| 49 | 72 | 14 | 5 | | | | |
| 50 | 2 | | | | | | |

(3) Effects of the genetic operators - mutation and crossover

Previous examples have shown the overall effect of the GA operations, where the genetic modification processes on the query term weights have improved retrieval results. However, in most situations, the function of the two important operators - mutation which assigns a new weight to a randomly selected term, and crossover which exchanges individual term weights between two randomly selected pairs - is difficult to display. However, in experiments on the DOE database we found one example, topic 13, which shows how mutation and crossover can create a new query individual retrieving, in this example, all of the relevant documents which are retrieved part by one of their parent and part by the other parent, together with the other new query individual retrieving none.

Table 12 depicts this situation. The top part of the table shows the term weights of the query individuals. The left side are the two original query individuals - the parents, and the right side are the children after mutation and crossover. The rest of the table shows the documents retrieved by the parent 1, parent 2 and the children. We can see that one of the children has inherited all of the proper term weights from the parents, and this child retrieved all of the relevant documents retrieved by both of the parents. Note that although two of the term weights are changed by the mutation, it does not affect the combination effect because the difference between the changed weights and the original ones from one of the parents is not significant. Of cause, the effects of crossover and mutation are not always this clear cut.

(4) Parallel search using multiple query individuals

The genetic algorithm using multiple individuals in the document retrieval process provides the capability of parallel search with different query individuals searching the document space simultaneously. Each individual, with the same terms but different weights, may retrieve different documents which are located in different areas in the document space. This makes the genetic search distinct from other searching methods, where only one query is used.

The effects of parallel search can be explained by showing the documents retrieved from several query individuals. Here we give the results from topic 24. Table 13 displays the term weights of query individuals in the first generation, and Table 14 shows the relevant documents retrieved by four different query individuals from Table 13. We can see that for these four queries, there is no overlap among the relevant documents retrieved. Thus each query individual emphasizes different concepts by assigning high weights to them. Documents which are more closely related to those concepts would be retrieved by different query individuals.

The situation also can be viewed as having the query individuals handle the AND and OR operation simultaneously, which is impossible for methods using only one query unless they use the Boolean model. However, many researchers reject the Boolean model because it is difficult for users to apply the AND/OR operations correctly (e.g., Bookstein, 1985).

Table 12 The combined effect of crossover and mutation, topic 13

| | t1 | t2 | t3 | t4 | t5 |
|---|---|---|---|---|---|
| parent 1 | .77 | .65 | .77 | .78 | .82 |
| parent 2 | .46 | .98 | .13 | .21 | .96 |

| | t1 | t2 | t3 | t4 | t5 |
|---|---|---|---|---|---|
| child 1 | .72 | .65 | .77 | .59 | .96 |
| child 2 | .51 | .98 | .13 | .39 | .82 |

**Note:** underline means caused by crossover
double underline means caused by mutation

| Documents retrieved by parent 1 | Documents retrieved by parent 2 | Documents retrieved by child 1 | Documents retrieved by child 2 |
|---|---|---|---|
| DOE1-05-0747 | DOE1-24-0861 | DOE1-05-0747 | none |
| DOE1-11-0383 | DOE1-26-0555 | DOE1-11-0383 | |
| DOE1-44-0234 | DOE1-36-0791 | DOE1-44-0234 | |
| DOE1-49-0051 | DOE1-50-0948 | DOE1-49-0051 | |
| DOE1-69-0957 | DOE1-79-0679 | DOE1-69-0957 | |
| DOE1-81-0200 | DOE2-27-0093 | DOE1-81-0200 | |
| DOE2-20-0798 | DOE2-69-1023 | DOE2-20-0798 | |
| DOE2-27-0146 | DOE2-70-0787 | DOE2-27-0146 | |
| DOE2-32-1039 | DOE2-71-0280 | DOE2-32-1039 | |
| DOE2-37-0525 | DOE2-80-0365 | DOE2-37-0525 | |
| DOE2-48-0100 | | DOE2-48-0100 | |
| DOE2-49-0342 | | DOE2-49-0342 | |
| DOE2-71-0973 | | DOE2-71-0973 | |
| | | DOE1-24-0861 | |
| | | DOE1-26-0555 | |
| | | DOE1-36-0791 | |
| | | DOE1-50-0948 | |
| | | DOE1-79-0679 | |
| | | DOE2-27-0093 | |
| | | DOE2-69-1023 | |
| | | DOE2-70-0787 | |
| | | DOE2-71-0280 | |
| | | DOE2-80-0365 | |

The parallel search could be continued if the individuals retrieving different relevant documents have the same power, that is, the differences in the number of relevant documents brought by each of them are small. The results in Table 15 and Table 16 come from the same topic in Table 13 and Table 14, but from the second generation (generation 1) after feedback and genetic modification to the first generation. It can be seen that due to the genetic operations, parts of several query individuals have begun converging. At this point the queries group into three clusters of similar queries, {0,3,7,8,9}, {1,5,6} and {2,4}-- Table 15. Within each group several term weights are identical. However, parallel search still continues. Differences in the other weights result in query individuals that retrieve completely different sets of documents. For example, contrast query 7 with the others in its cluster. Other examples from this study show that this effect may be observed with differences in only one or two term weights.

Tables 17 and 18 show that the parallel search from this topic went on to the final generation, 3, where for each term almost all of the weights had converged to a single value. But query 7 has two different values at terms 4 and 21 which cause it to retrieve totally different relevant documents than the other query individuals. However, this is not always the case. In most situations, the genetic algorithm will force the query variants to converge to a single query; that is, all of the query individuals in the final generation retrieve the same relevant documents.

The example on topic 24 also shows, as in previous cases, that new relevant documents are brought to the user in each generation, as indicated by the '*' following the document numbers.

Table 13. Term weights of query individuals of topic 24, the first generation (generation 0)

| | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ | $t_8$ | $t_9$ | $t_{10}$ | $t_{11}$ | $t_{12}$ | $t_{13}$ | $t_{14}$ | $t_{15}$ | $t_{16}$ | $t_{17}$ | $t_{18}$ | $t_{19}$ | $t_{20}$ | $t_{21}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 18 | .31 | .53 | .95 | .17 | .70 | .23 | .49 | .12 | .08 | .39 | .28 | .37 | .98 | .54 | .77 | .65 | .77 | .78 | .82 | .15 |
| 1 | .63 | .31 | .35 | .92 | .52 | .40 | .61 | .79 | .93 | .87 | .87 | .67 | .76 | .58 | .39 | .36 | .20 | .83 | .42 | .46 | .98 |
| 2 | .13 | .21 | .96 | .74 | .41 | .78 | .76 | .96 | .03 | .32 | .76 | .24 | .59 | .04 | .96 | .32 | .06 | .44 | .92 | .57 | .12 |
| 3 | .57 | .25 | .50 | .24 | .48 | .41 | .87 | .43 | .36 | .38 | .04 | .16 | .52 | .70 | .10 | .40 | .77 | .24 | .34 | .23 | .30 |
| 4 | .30 | .89 | .04 | .65 | .40 | .68 | .73 | .94 | .23 | .84 | .97 | .78 | .43 | .67 | .81 | .16 | .28 | .14 | .86 | .75 | .21 |
| 5 | .14 | .29 | .80 | .22 | .56 | .72 | .20 | .99 | .25 | .43 | .76 | .86 | .89 | .98 | .40 | .43 | .13 | .46 | .24 | .99 | .65 |
| 6 | .60 | .24 | .45 | .79 | .08 | .48 | .15 | .25 | .94 | .61 | .99 | .48 | .80 | .74 | .38 | .48 | .53 | .10 | .59 | .35 | .14 |
| 7 | .78 | .71 | .45 | .70 | .10 | .96 | .55 | .74 | .58 | .64 | .78 | .19 | .30 | .28 | .68 | .29 | .57 | .42 | .31 | .44 | .57 |
| 8 | .49 | .61 | .42 | .13 | .26 | .04 | .98 | .11 | .38 | .65 | .35 | .55 | .36 | .57 | .48 | .16 | .62 | .17 | .55 | .29 | .87 |
| 9 | .84 | .84 | .90 | .59 | .54 | .17 | .65 | .69 | .26 | .11 | .81 | .19 | .42 | .35 | .84 | .14 | .26 | .18 | .48 | .38 | .50 |

Table 14 Relevant documents retrieved by query individuals for topic 24, the first generation (generation 0)

| | Query individuals | | | |
|---|---|---|---|---|
| Document number | 0 | 3 | 6 | 8 |
| DOE1-57-0764 | v | | | |
| DOE2-04-0727 | v | | | |
| DOE1-75-0356 | v | | | |
| DOE1-56-0376 | v | | | |
| DOE2-14-0677 | v | | | |
| DOE1-07-1283 | v | | | |
| DOE1-14-1165 | v | | | |
| DOE1-35-0677 | v | | | |
| DOE1-35-0739 | v | | | |
| DOE1-53-1065 | v | | | |
| DOE1-77-0170 | v | | | |
| DOE1-85-0552 | v | | | |
| DOE2-15-0328 | v | | | |
| DOE2-31-1277 | v | | | |
| DOE2-61-0632 | | v | | |
| DOE2-28-0819 | | v | | |
| DOE1-55-0792 | | v | | |
| DOE1-70-0125 | | v | | |
| DOE1-01-0620 | | v | | |
| DOE1-11-0738 | | | v | |
| DOE2-73-0983 | | | v | |
| DOE2-71-0057 | | | v | |
| DOE2-73-0254 | | | v | |
| DOE1-85-0035 | | | v | |
| DOE1-06-1157 | | | v | |
| DOE1-22-1160 | | | v | |
| DOE1-09-1204 | | | v | |
| DOE1-01-0602 | | | v | |
| DOE1-73-0028 | | | v | |
| DOE1-19-0847 | | | | v |
| DOE1-21-1059 | | | | v |
| DOE2-83-0561 | | | | v |
| DOE1-09-1154 | | | | v |
| DOE1-30-0306 | | | | v |
| DOE1-33-0952 | | | | v |
| DOE2-25-0240 | | | | v |
| DOE1-07-1305 | | | | v |
| DOE1-70-0170 | | | | v |
| DOE1-73-0068 | | | | v |

(5) Precision and recall

The average performance of the current genetic algorithm can be shown using precision and recall values. Here we use data provided by NIST to show the results from the ad hoc queries and the routing queries. Note that in our experiment the ad hoc queries were used only on the second dataset (disk two). Table 19 and Table 20 display the average precision at the 11 recall levels from both the ad hoc queries and the routing queries, respectively.

## 8. Discussion

The experimental results on the TREC document collection have shown that the genetic approach in query modification can be applied in large and varied databases. Evidence has indicated that the genetic query modification leads to the convergence of the query term weights. The results have also shown that additional relevant documents can be retrieved by modifying the query term weights using the genetic approach.

We have also shown, through specific examples, the effects of crossover and mutation, and how variations in query term weights affect the retrieval of relevant documents. We believe that the genetic algorithm technique introduces new information into the retrieval process, allowing the system to do a better job of query modification.

Evidence has shown that even without the use of genetic techniques our system employs parallel search, although the genetic modification brings additional relevant documents. For the system designer interested in Boolean techniques this permits exploration of several Boolean variants simultaneously, and also facilitates use of Boolean combinations of the individual query variants.

Table 15. Term weights of query individuals of topic 24, the second generation (generation 1)

| | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ | $t_8$ | $t_9$ | $t_{10}$ | $t_{11}$ | $t_{12}$ | $t_{13}$ | $t_{14}$ | $t_{15}$ | $t_{16}$ | $t_{17}$ | $t_{18}$ | $t_{19}$ | $t_{20}$ | $t_{21}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | .18 | .31 | .53 | .95 | .17 | .70 | .23 | .49 | .12 | .08 | .39 | .28 | .37 | .96 | .38 | .48 | .69 | .77 | .78 | .82 | .15 |
| 1 | .60 | .24 | .45 | .79 | .08 | .48 | .15 | .25 | .94 | .61 | .99 | .48 | .80 | .76 | .54 | .77 | .49 | .10 | .59 | .35 | .14 |
| 2 | .49 | .61 | .42 | .13 | .26 | .04 | .98 | .11 | .38 | .65 | .35 | .55 | .36 | .57 | .48 | .16 | .74 | .57 | .55 | .29 | .87 |
| 3 | .18 | .31 | .53 | .95 | .28 | .70 | .23 | .49 | .12 | .08 | .39 | .28 | .37 | .98 | .54 | .77 | .53 | .37 | .78 | .82 | .15 |
| 4 | .49 | .61 | .42 | .13 | .26 | .04 | .67 | .25 | .94 | .61 | .99 | .48 | .80 | .75 | .48 | .16 | .62 | .17 | .55 | .29 | .87 |
| 5 | .60 | .24 | .45 | .79 | .08 | .48 | .47 | .11 | .38 | .65 | .35 | .55 | .36 | .55 | .38 | .48 | .53 | .10 | .59 | .35 | .14 |
| 6 | .60 | .24 | .45 | .79 | .08 | .48 | .15 | .25 | .94 | .61 | .79 | .28 | .37 | .98 | .54 | .77 | .65 | .77 | .78 | .69 | .14 |
| 7 | .18 | .31 | .53 | .95 | .17 | .70 | .23 | .49 | .12 | .08 | .59 | .48 | .80 | .74 | .38 | .48 | .53 | .10 | .59 | .48 | .98 |
| 8 | .18 | .31 | .53 | .95 | .17 | .70 | .23 | .49 | .12 | .08 | .39 | .28 | .37 | .98 | .54 | .77 | .65 | .77 | .78 | .82 | .15 |
| 9 | .18 | .31 | .53 | .95 | .17 | .70 | .23 | .49 | .12 | .08 | .39 | .28 | .37 | .98 | .54 | .77 | .65 | .77 | .78 | .82 | .15 |

Table 16. Relevant documents retrieved by query individuals for topic 24, the second generation ( * new retrieved documents)

| | Query individuals | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Document number | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| DOE1-57-0764 | v | | | v | | | | | v | v |
| DOE2-04-0727 | v | | | v | | | | | v | v |
| DOE1-56-0376 | v | | | v | | | | | v | v |
| DOE1-07-1283 | v | | | | | | | | v | v |
| DOE1-14-1165 | v | | | | | | | | v | v |
| DOE1-35-0677 | v | | | | | | | | v | v |
| DOE1-35-0739 | v | | | | | | | | v | v |
| DOE1-53-1065 | v | | | | | | | | v | v |
| DOE1-77-0170 | v | | | | | | | | v | v |
| DOE1-85-0552 | v | | | | | | | | v | v |
| DOE2-15-0328 | v | | | | | | | | v | v |
| DOE2-31-1277 | v | | | | | | | | v | v |
| DOE2-73-0254 | | v | | | | | v | | | |
| DOE1-75-0356 | | v | | v | | | v | | v | v |
| DOE1-06-1164* | | v | | | v | | | | | |
| DOE1-11-0738 | | v | | | v | | | | | |
| DOE2-73-0983 | | v | | | | | | | | |
| DOE2-71-0057 | | v | | | | | | | | |
| DOE2-28-0819 | | | v | | v | | | | | |
| DOE1-19-0847 | | | v | | | | | | | |
| DOE1-21-1059 | | | v | | | | | | | |
| DOE2-83-0561 | | | v | | | | | | | |
| DOE1-09-1154 | | | v | | | | | | | |
| DOE1-30-0306 | | | v | | | | | | | |
| DOE1-33-0952 | | | v | | | | | | | |
| DOE2-25-0240 | | | v | | | | | | | |
| DOE1-55-0888* | | | | | v | | | | | |
| DOE1-83-0442* | | | | | v | | | | | |
| DOE2-42-0268* | | | | | v | | | | | |
| DOE2-72-0035* | | | | | v | | | | | |
| DOE1-77-0125* | | | | | | v | | | | |
| DOE1-40-1152* | | | | | | | | v | | |
| DOE1-11-0490* | | | | | | | | v | | |
| DOE1-15-1164* | | | | | | | | v | | |
| DOE1-35-0747* | | | | | | | | v | | |
| DOE1-46-0311* | | | | | | | | v | | |
| DOE1-73-0024* | | | | | | | | v | | |
| DOE2-39-0087* | | | | | | | | v | | |
| DOE2-71-0068* | | | | | | | | v | | |
| DOE2-71-0075* | | | | | | | | v | | |
| DOE1-43-0907* | | | | | | | | v | | |
| DOE1-76-1035* | | | | | | | | v | | |
| DOE1-82-0152* | | | | | | | | | v | v |
| DOE2-14-0677 | | | | | | | | | v | v |

Table 17. Query individuals on Topic 24 (generation 3)

| | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.18 | 0.31 | 0.53 | 0.95 | 0.17 | 0.70 | 0.23 | 0.49 | 0.12 | 0.08 | 0.39 | 0.28 | 0.37 | 0.98 | 0.54 | 0.77 | 0.65 | 0.77 | 0.78 | 0.82 | 0.15 |
| 1 | 0.18 | 0.31 | 0.53 | 0.95 | 0.17 | 0.70 | 0.23 | 0.49 | 0.12 | 0.08 | 0.39 | 0.28 | 0.37 | 0.98 | 0.54 | 0.77 | 0.65 | 0.77 | 0.78 | 0.82 | 0.15 |
| 2 | 0.18 | 0.31 | 0.53 | 0.95 | 0.17 | 0.70 | 0.33 | 0.59 | 0.12 | 0.08 | 0.39 | 0.28 | 0.37 | 0.98 | 0.54 | 0.77 | 0.75 | 0.77 | 0.78 | 0.82 | 0.15 |
| 3 | 0.18 | 0.31 | 0.53 | 0.95 | 0.28 | 0.70 | 0.23 | 0.49 | 0.12 | 0.08 | 0.39 | 0.28 | 0.37 | 0.98 | 0.54 | 0.77 | 0.55 | 0.77 | 0.78 | 0.82 | 0.15 |
| 4 | 0.18 | 0.31 | 0.53 | 0.95 | 0.17 | 0.70 | 0.33 | 0.59 | 0.12 | 0.08 | 0.39 | 0.28 | 0.37 | 0.98 | 0.54 | 0.77 | 0.65 | 0.77 | 0.78 | 0.82 | 0.15 |
| 5 | 0.18 | 0.31 | 0.53 | 0.95 | 0.17 | 0.70 | 0.23 | 0.49 | 0.12 | 0.08 | 0.39 | 0.28 | 0.37 | 0.98 | 0.54 | 0.77 | 0.65 | 0.77 | 0.78 | 0.82 | 0.15 |
| 6 | 0.18 | 0.31 | 0.53 | 0.95 | 0.17 | 0.70 | 0.33 | 0.59 | 0.12 | 0.08 | 0.39 | 0.28 | 0.37 | 0.98 | 0.54 | 0.77 | 0.65 | 0.77 | 0.78 | 0.82 | 0.15 |
| 7 | 0.18 | 0.31 | 0.53 | 0.95 | 0.17 | 0.70 | 0.13 | 0.39 | 0.12 | 0.08 | 0.39 | 0.28 | 0.37 | 0.98 | 0.54 | 0.77 | 0.65 | 0.77 | 0.78 | 0.82 | 0.98 |
| 8 | 0.18 | 0.31 | 0.53 | 0.95 | 0.17 | 0.70 | 0.23 | 0.49 | 0.12 | 0.08 | 0.39 | 0.28 | 0.37 | 0.98 | 0.54 | 0.77 | 0.65 | 0.77 | 0.78 | 0.82 | 0.15 |
| 9 | 0.18 | 0.31 | 0.53 | 0.95 | 0.17 | 0.70 | 0.23 | 0.49 | 0.12 | 0.08 | 0.39 | 0.28 | 0.37 | 0.98 | 0.54 | 0.77 | 0.65 | 0.77 | 0.78 | 0.82 | 0.15 |

Table 18. Relevant documents retrieved by query individuals for topic 24, (generation 3) ( * new retrieved documents)

| Document number | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| DOE1-57-0764 | v | v | v | v | v | v | v | | v | v |
| DOE1-75-0356 | v | v | v | v | v | v | v | | v | v |
| DOE2-07-0957* | v | v | v | v | v | v | v | | v | v |
| DOE2-55-0603* | v | v | v | v | v | v | v | | v | v |
| DOE1-82-0152 | v | v | v | v | v | v | v | | v | v |
| DOE1-82-0447 | v | v | v | v | v | v | v | | v | v |
| DOE2-14-0677 | v | v | v | | v | v | v | | v | v |
| DOE1-07-1283 | v | v | v | | v | v | v | | v | v |
| DOE1-53-1065 | v | v | v | | v | v | v | | v | v |
| DOE1-59-1152 | v | v | | v | v | v | v | | v | v |
| DOE1-82-0434* | v | v | v | | v | v | v | | v | v |
| DOE2-63-0711* | | | v | | | | | | | |
| DOE1-07-1313* | | | | v | | | | | | |
| DOE1-12-0905* | | | | | | | | v | | |
| DOE1-23-0511 | | | | | | | | v | | |
| DOE2-33-1315* | | | | | | | | v | | |
| DOE1-48-0009 | | | | | | | | v | | |
| DOE1-16-1155* | | | | | | | | v | | |
| DOE2-40-1200* | | | | | | | | v | | |
| DOE1-11-0490 | | | | | | | | v | | |
| DOE1-15-1164 | | | | | | | | v | | |
| DOE1-35-0747 | | | | | | | | v | | |
| DOE1-43-0913* | | | | | | | | v | | |
| DOE1-46-0311 | | | | | | | | v | | |
| DOE1-73-0024 | | | | | | | | v | | |
| DOE1-93-0736* | | | | | | | | v | | |
| DOE2-71-0075 | | | | | | | | v | | |
| DOE1-33-0034* | | | | | | | | v | | |
| DOE1-84-1200* | | | | | | | | v | | |
| DOE1-85-0564 | | | | | | | | v | | |

Table 19. Precision and recall for ad hoc queries

Top ranked evaluation

Run number: upitt3 all
Num_queries: 50
Total number of documents over all queries
Retrieved: 5383
Relevant: 5923
Rel_ret: 1249
Recall - Precision Averages:
at 0.00 0.5622
at 0.10 0.2723
at 0.20 0.2025
at 0.30 0.1382
at 0.40 0.0708
at 0.50 0.0309
at 0.60 0.0279
at 0.70 0.0155
at 0.80 0.0136
at 0.90 0.0000
at 1.00 0.0000
Average precision for all points
11-pt Avg: 0.1213
Average precision for 3 intermediate points (0.20, 0.50, 0.80)
3-pt Avg: 0.0824

Recall:

at 5 docs: 0.0192
at 15 docs: 0.0509
at 30 docs: 0.0849
at 100 docs: 0.1753
at 200 docs: 0.2206

Precision:

At 5 docs: 0.3280
At 15 docs: 0.2867
At 30 docs: 0.2533
At 100 docs: 0.1844
At 200 docs: 0.1249

Table 20. Precision and recall for routing queries

Top ranked evaluation

Run number: upitt3 all
Num_queries: 50
Total number of documents over all queries
Retrieved: 7611
Relevant: 14216
Rel_ret: 1277
Recall - Precision Averages:
at 0.00 0.5285
at 0.10 0.2057
at 0.20 0.0649
at 0.30 0.0085
at 0.40 0.0000
at 0.50 0.0000
at 0.60 0.0000
at 0.70 0.0000
at 0.80 0.0000
at 0.90 0.0000
at 1.00 0.0000
Average precision for all points
11-pt Avg: 0.0734
Average precision for 3 intermediate points (0.20, 0.50, 0.80)
3-pt Avg: 0.0216

Recall:

at 5 docs: 0.0090
at 15 docs: 0.0215
at 30 docs: 0.0362
at 100 docs: 0.0842
at 200 docs: 0.1007

Precision:

At 5 docs: 0.3520
At 15 docs: 0.3013
At 30 docs: 0.2727
At 100 docs: 0.2102
At 200 docs: 0.1277

## 9. Failure Analysis

Comparing our results with those provided by NIST, the precision values at eleven recall points indicate that our system performs better than the median level on about half of the topics for the ad hoc queries (Figure 1). In several queries (54, 60, 79, 81, 84, 91, and 100) we sent only a few documents (fewer than 10), and for queries 59, 61, 68, 80 and 99 we sent no documents, because the threshold values limited the number of documents retrieved. Thus the precision values for those queries are zero or very low. The same situation happened for some routing queries.

The precision values for the routing queries in our system are lower than the median level in most cases (Figure 2). Beside the threshold inhibition mentioned above, we observed that due to the query convergence in the final generation of the training topic, most query variants retrieved the same documents. Some query individuals in the intermediate generations which retrieved different relevant documents than the last generation may not have survived. We think this caused the situation where fewer relevant documents were retrieved on the routing queries.

Problems arose with specific queries due to our pre-processing of the documents. Several circumstances were not considered in designing our system. For example, some special keywords, such as *AT&T* and *M* which was used in some documents to represent *million*, were not processed, but were significant in some topics. Another factor is that in the AP, WSJ and ZIFF databases more than one text with different topics comprised a single document. Since we did not separate them, the keyword match could cause a document to be retrieved because keywords from different text parts matched the query, though the document itself is not relevant.

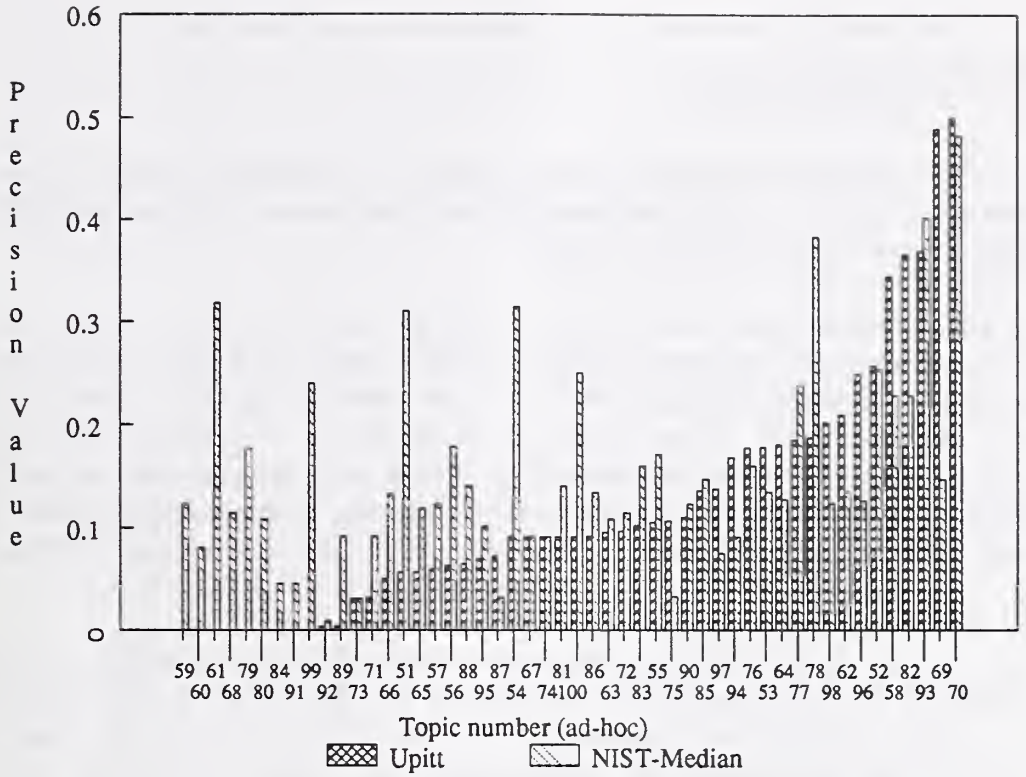



Figure 1. Precision values on 50 ad hoc queries

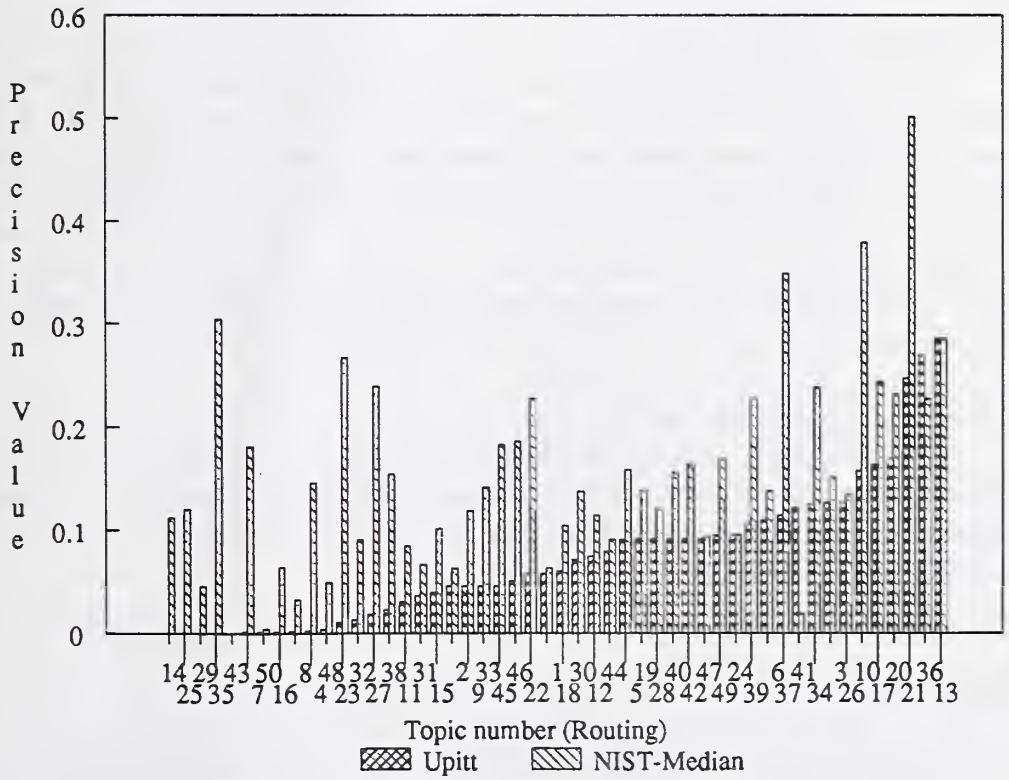



Figure 2. Precision values on 32 routing queries

## 10. Further Studies

In dealing with these large databases, our algorithm shows many interesting phenomena which we could not deal with due to the short time period of TREC, and which could form the agenda for further studies.

(1) The document term weights

Although we use the vector space model mentioned in Section 3, unweighted keyterms are assigned to describe the documents. We have mentioned the two factors based on which we made the decision. Moreover, the databases consist of a variety of subjects which may make the general term weights less reliable than in most experimental databases, which deal with only a narrow area. While our Cranfield study supports this decision, the effect of weighted and unweighted document keywords in large databases should be further studied.

(2) Window size for document retrieval

Although we established the maximum size of the retrieved document set as 40, it would be interesting to set the window size smaller than 40 at the beginning of the process and gradually increase the size after the query individuals move toward convergence. Various thresholds and cutoff points are often arbitrarily set. Further experimentation is planned to investigate the effect of changes in these values.

(3) Document classification

In this experiment we found many cases in which different query individuals retrieved different documents. This may arise because the database contains relevant documents that belong to different classes, that can be retrieved only by using different queries. This fact would never be discovered by systems that rely on a single query. Our present algorithm aims at convergence to a single query form. We need a mechanism which will detect the development of distinct classes, and automatically separate the query into two or more distinct queries. Similar ideas, not based on genetic algorithms, have been discussed in document classification (Worona, 1971) and in query splitting (Borodin, et. al., 1971).

(4) Term addition and deletion

In our feedback process, no terms were added to or removed from the original query. This may cause lower recall. However, the determination of which terms should be added or removed is not an easy task (e.g., Harman, 1988, 1992). It would be interesting to see how this problem can be handled in the genetic process.

## Reference

Aalbersberg, I.J. (1992). Incremental relevance feedback. *Proceedings of the 15th Annual International SIGIR Conference*, pp 11-22. Copenhagen, Denmark (June, 1992).

Baker, J.E. (1987). Reducing bias and inefficiency in the selection algorithm. In J.J. Grefenstette (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*. pp. 14-21. NJ: Lawrence Erlbaum Associates, Publishers.

Bookstein, A. (1985). Probability and fuzzy-set applications to information retrieval. In: M. E. Williams (Ed.), *Annual Review of Information Science and Technology*, Vol. 20, pp. 117-151.

Borodin, A., Kerr, L. and Lewis, F. (1971). Query splitting in relevance feedback systems. In: G. Salton (Ed.), *The Smart Retrieval System: experiments in automatic document processing*. NJ: Prentice Hall.

Chang, Y.K., Cirillo, C. and Razon, J. (1971). Evaluation of feedback retrieval using modified freezing, residual collection, and test and control groups. In: G. Salton (Ed.) (1971). *The SMART Retrieval System: experiments in automatic document processing*. New Jersey: Prentice-Hall, Inc.

Frakes, W.B. (1992). Stemming Algorithms. In W.B. Frakes and R. Baeza-Yates, (Eds.) *Information Retrieval: data structures and algorithms*, pp. 131-160. NJ: Prentice Hall.

Frieder, O. and Siegelmann, H. (1991). On the allocation of documents in multiprocessor information retrieval systems. *Proceedings of the Fourteenth Annual International ACM/SIGIR Conference on Research & Development in Information Retrieval*, pp. 230-239. Illinois: Chicago.

Goldberg, D. E. (1989). *Genetic Algorithms: in Search, Optimization, and Machine Learning*. New York: Addison-Wesley Publishing Company, Inc.

Gordon, M. D. (1988). The necessity for adaptation in modified Boolean document retrieval systems. *Information Processing & Management,* 24(3), pp. 339-347.

Harman, D. (1988). Towards interactive query expansion. *Proceedings of the Eleventh Annual International ACM/SIGIR Conference*, pp. 321-331.

Harman, D. (1992). Relevance feedback revisited. *Proceedings of the 15th Annual International SIGIR Conference*, pp 1-10. Copenhagen, Denmark (June, 1992).

Holland, J. H. (1975) *Adaptation in Natural and Artificial Systems*. Ann Arbor: The University of Michigan Press.

Ide, E. (1971). New experiments in relevance feedback. In: G. Salton (Ed.) (1971). *The SMART Retrieval System: experiments in automatic document processing*, Chapter 16. New Jersey: Prentice-Hall, Inc.

Salton, G. (Ed.). (1971). *The SMART Retrieval System: Experiments in Automatic Document Processing*. NJ: Prentice-Hall, Inc.

Salton, G. (1986). Recent trends on automatic information retrieval. *Proceedings of ACM/SIGIR Conference on Research and Development in Information Retrieval*, pp. 1-10. Pisa, Italy (September, 1986).

Salton, G. and Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5), pp. 513-523.

Worona, S. (1971). Query clustering in a large document space. In: G. Salton (Ed.) (1971). *The SMART Retrieval System: experiments in automatic document processing*, Chapter 12. New Jersey: Prentice-Hall, Inc.

Yang, J.J. and Korfhage, R.R. (1992a). Adaptive information retrieval systems in vector space model. *Proceedings of Symposium on Document Analysis and Information Retrieval*, pp. 134-150. Las Vegas, Nevada (March, 1992).

Yang, J.J. and Korfhage, R.R. (1992b). Query modification using genetic algorithms in vector space models. *Research Reports, LIS045/IS92001*, Department of Information Science, University of Pittsburgh.

# Automatic Retrieval With Locality Information Using SMART

Chris Buckley*, Gerard Salton, and James Allan

## Abstract

The Smart project at Cornell University, using a completely automatic approach for both routing and ad-hoc experiments, performed extremely well in the first Text Retrieval Conference. The basic ad-hoc approach uses local/global matching to achieve its results. A global match ensures that each retrieved document uses the same vocabulary as the query; a local match then attempts to guarantee some local part of the document (eg, a paragraph or sentence) focuses on the query topic. Runs were made with and without simple adjacency phrases. A simple relevance feedback algorithm is used for routing experiments; the original query is expanded by terms occurring in relevant documents, with term weights being based upon occurrence in the relevant documents. In addition, a set of system design issues and tradeoffs are examined.

## Introduction

For over 30 years, the Smart project at Cornell has been interested in the analysis, search, and retrieval of heterogeneous text databases, where the vocabulary is allowed to vary widely, and the subject matter is unrestricted. Such databases may include newspaper articles, newswire dispatches, textbooks, dictionaries and encyclopedias, manuals, magazine articles, and so on. The usual text analysis and text indexing approaches that are based on the use of thesauruses and other vocabulary control devices are difficult to apply in unrestricted text environments, because the word meanings are not stable in such circumstances and the interpretation varies depending on context. The applicability of more complex text analysis systems that are based on the construction of knowledge bases covering the detailed structure of particular subject areas, together with inference rules designed to derive relationships between the relevant concepts, is even more questionable in such cases. Complete theories of knowledge representation do not exist, and it is unclear what concepts, concept relationships and inference rules may be needed to understand particular texts.[5]

Accordingly, a text analysis and retrieval component must necessarily be based primarily on a study of the available texts themselves. Fortunately very large text databases are now available in machine-readable form, and a substantial amount of information is automatically derivable about the occurrence properties of words and expressions in natural-language texts, and about the contexts in which the words are used. This information can help in determining whether two or more texts are semantically homogeneous, that is, whether they cover similar subject areas. When that is the case, such semantically homogeneous texts can be linked, thereby generating an automatic structured text (hypertext) representation; alternatively, in a retrieval setting a text can be retrieved when another semantically homogeneous text is submitted as a query.

*Department of Computer Science, Cornell University, Ithaca, NY 14853-7501. This study was supported in part by the National Science Foundation under grant IRI 89-15847.

## Automatic Indexing

In the Smart context, the vector-processing model of retrieval is used to transform both the available information requests as well as the stored documents into vector form of the type

$$D_i = (w_{i1}, w_{i2}, \ldots, w_{it})$$

where $D_i$ represents a document (or query) text and $w_{ik}$ is a term weight of term $T_k$ attached to document $D_i$. A weight of zero is used for terms that are absent from a particular document, and positive weights characterize terms actually assigned. The assumption is that $t$ terms in all are available for the representation of the information.

In choosing a term weighting system, low weights should be assigned to high-frequency terms that occur in many documents of a collection, and high weights to terms that are important in particular documents but unimportant in the remainder of the collection. The weight of terms that occur rarely in a collection is relatively unimportant, because such terms contribute little to the needed similarity computation between different texts.

A well-known term weighting system following that prescription assigns weights $w_{ik}$ to term $T_k$ in document $D_i$ in proportion to the frequency of occurrence of a term in $D_i$, and in inverse proportion to the number of documents to which the term is assigned.[6,9] Such a weighting system is known as a $tf * idf$ (term frequency times inverse document frequency) weighting system. In practice the document length, and hence the number of non-zero term weights assigned to as document, varies widely. To give each text item an equal chance of being retrieved, it is convenient to use a length normalization factor as part of the term weighting formula. A high-quality term weighting formula for $w_{ik}$, the weight of term $T_k$ in document $D_i$ is

$$w_{ik} = \frac{f_{ik} * \log(N/n_k)}{\sqrt{\sum_{k=1}^{t} (f_{ik} * \log(N/n_k))^2}} \qquad (1)$$

where $f_{ik}$ is the occurrence frequency of $T_k$ in $D_i$, $N$ is the collection size, and $n_k$ the number of documents with term $T_k$ assigned. The factor $\log(N/n_k)$ is an inverse collection frequency factor which decreases as terms are used widely in a collection, and the denominator in expression (1) is used for weight normalization.

The terms $T_k$ included in a given vector can in principle represent any entities assigned to a document for content identification. In the Smart context, such terms are derived by a text transformation of the following kind:[2]

1. recognize individual text words
2. use stop list to eliminate unwanted function words
3. perform suffix removal to generate word stems
4. optionally use term grouping methods based on a statistical word co-occurrence, or word adjacency, computation to form term phrases (alternatively syntactic analysis computations can also be used)
5. assign term weights to all remaining word stems and/or phrase stems to form the term vector for all information items.

Once term vectors are available for all information items, all subsequent processing is based on term vector manipulations.

The fact that the indexing of both documents and queries is completely automatic means that the results obtained are reasonably collection independent and should be valid across a wide range of collections. No human expertise in the subject matter is required for either the initial collection creation, or the actual query formulation.

## Text Similarity Computation

When the texts are represented by term vector of the form $D_i = (w_{i1}, w_{i2}, \ldots, w_{it})$ and $D_j = (w_{j1}, w_{j2}, \ldots, w_{jt})$ for documents $D_i$ and $D_j$, a similarity ($S$) computation between two items can conveniently be obtained as the inner product between corresponding weighted term vector as follows:

$$S(D_i, D_j) = \sum_{k=1}^{t} (w_{ik} * w_{jk}) \tag{2}$$

Thus, the similarity between two texts (whether query or document) depends on the weights of coinciding terms in the two vectors.

Information retrieval and text linking systems based on the use of global text similarity measures such as that of expression (2) will be successful when the common terms in the two vectors are in fact used in semantically similar ways. In many cases it may however happen that highly-weighted terms that contribute substantially to the text similarity are semantically distinct. For example, a sound may be an audible phenomenon, or a body of water.

In determining the meaning of individual words, we take advice from Wittgenstein and others who suggest that text understanding must be based on a study of how text words are used in the language ("word use" theory of text meaning).[11] In a mechanized text manipulation environment, "word use" may be interpreted as the contexts in which the words are used in the texts in which they occur. The assumption then is that identical words used in identical contexts (that is, in substantially similar phrases, sentences and paragraphs) are in fact semantically homogeneous. Contrariwise, similar words such as "sound" are expected to occur in different local environments when they represent different entities such as bodies of water and audible phenomena.

To detect similarity of local term environments, we carry out text similarity measurements such as those suggested by expression (2), but applied to small text units such as text sentences and text paragraphs. Two texts are then accepted as related only when a sufficiently high global text similarity exists, as well as sufficient local text similarities in the form of similarities between sentences and/or paragraphs in the texts under study.[7,8]

A complete text retrieval system based on text similarity computations is then generated in the following way:

1. formation of term vectors for the text items
2. computation of text similarities, and elimination of text pairs with insufficient global text similarities
3. computation of local text similarities for the remaining texts
4. retrieval of text items with sufficiently large global and local similarities
5. use of user relevance judgments for rephrasing of search requests using relevance feedback.

This process is expected to perform effectively in producing both high precision as well as high recall.

## System Description

The Cornell TREC experiments use the SMART Information Retrieval System, Version 11 and were run on a dedicated Sun Sparcs 2 with 64 Mbytes of memory and 5 Gbytes of local disk.

SMART Version 11 is the latest in a long line of experimental information retrieval systems, dating back over 30 years, developed under the guidance of G. Salton. Version 11 is a reasonably complete re-write of earlier versions, and was designed and coded by C. Buckley. The new version is approximately 44,000 lines of C code and documentation.

SMART Version 11 offers a basic framework for investigations into the vector space and related models of information retrieval. Documents are fully automatically indexed, with each document representation being a weighted vector of concepts, with the weight indicating the importance of a concept to that particular document. The document representatives are physically stored on disk as an inverted file. Natural language queries will go through the same indexing process. The query representative vector is then compared with the indexed document representatives to arrive at a similarity. The documents are then fully ranked by similarity.

## Specific Methodology Used for TREC Study

There are two major sets of Cornell TREC experimental runs. The first set is the official TREC set with ad-hoc runs using the local/global matching procedure described above in steps 1-4. There are two automatic runs in this set; one using only single term indexing and the second using both single term and two term phrases. There is also an official routing run, using a simple relevance feedback technique to form a revised query based on relevance judgements from the training set.

The other set of runs provide an examination of some of the tradeoffs (disk space, memory, time, and effectiveness) encountered within a single information retrieval system. There are many decisions that need to be made when designing a system; the goal in this set of runs is to explore the consequences of some fundamental choices including

1. Degree of stemming
2. Size of stopword list
3. Inverse document freq weighting
4. Phrases
5. Query optimization

Both sets of runs use completely automatic indexing of queries and documents. Queries and documents are treated as flat text; some sections (like DOCID) might be omitted, but all indexable text is treated the same (unfortunately, even if preceded by a NOT!). This ignores the structure (in both form and semantic meaning) of the queries which could be very useful. SMART has the capability to treat different parts of query or document in different appropriate manners. However, using this structure would have tremendously complicated the second set of runs by adding another large set of variables to the experiments. Now that the choices investigated in the second set of runs have been made, future runs can use the structure of documents and queries.

## Official Runs (Ad-hoc Queries)

The overall procedure used for both of the official runs (one with single terms only, the other including phrases) is given in figure 1.

```
1. Automatically index document collection D1+D2 using
    tf * idf weights, cosine normalized (ntc) creating inverted file.
2. Automatically index query collection Q2 using
    tf * idf weights, cosine normalized (ntc)
3. For each Q in Q2
     3.1  Compute sim of Q to each of documents in D1+D2
          keeping track of the top 500 documents.
     3.2  Re-index Q, breaking it down into sentences.
          Each sentence is reweighted with tf * idf weights (ntn)
          and formed into a vector.
     3.3  For each D in the top 500 global sim documents
            3.3.1  Re-index D, breaking it down into sentences.
                   Each sentence is reweighted with tf * idf weights (ntn)
                   and formed into a vector.
            3.3.2  Do a pairwise comparison of every sentence
                   vector of Q against every sentence vector
                   of D.  If some sentence match satisfies the
                   local criteria, add a large constant to the
                   global sim of Q vs D already computed.
     3.4  Return the top 200 documents out of the set of 500
          documents.  Given the method above, first will be
          the documents satisfying the local criteria, sorted
          by global sim, and then the documents not
          satisfying the local criteria, sorted by global sim.
```

The local criteria varied in the two official runs.

## Single Term Global/Local Matching

Indexing the document collection for the single term run (step 1) took 5.4 CPU hours, creating an inverted file of 690 Mbytes.

The actual retrieval took 1465 CPU seconds for all 50 queries and considerably longer in elapsed time (a large amount of time being spent waiting for disk io). For each query, the text of 500 documents had to be read in, broken down into sentences, indexed, and weighted with idf weights. Then every indexed sentence in the query had to be compared against every indexed sentence in the document.

The local matching criteria are based on the detection of matching substructures in both the query texts and the texts of retrieved documents. The criterion used for the sample runs required the presence of at least one pair of matching text sentences with a pairwise sentence similarity of at least 100.0. This was chosen to be high enough so that very few matches of only one term would satisfy the threshold, but low enough so that several medium weighted terms could match and reach the threshold.

### Phrase Global/Local Matching

The phrases being used were two-term SMART adjacency phrases. Phrases were adjacent non-stopwords, term components stemmed, that occurred at least 25 times in the learning (D1) document set. The term components were put into alphabetical order, thus the text phrases "information retrieval" and "retrieving information" both mapped to the same phrase concept. The phrases were treated as a separate concept type (ctype) within an indexed vector, and had their own dictionary and inverted file separate from those of the single terms. The components of phrases remained in the single term ctype.

Determination of phrases took 5.8 hours, finding 4,700,000 phrases occurring in D1 at least once. Of those phrases 158,000 occurred at least 25 times. These phrases were then put into a dictionary and used as controlled vocabulary for phrases when doing the indexing of D1+D2. The single term indexing remained exactly as it was in the single term run (terms occurring in phrases were not removed from the vector).

Both the terms and phrases were given a "natural" $tf * idf$ weight (ie, the idf weight was based on the collection frequency of the phrase itself rather than being a function of the idf values of the single term components). The cosine normalization was handled in the following way. All terms in the vectors had their weight divided by the cosine length of the single term sub-vector instead of the vector as a whole. Thus the weights of single terms in the final vector were exactly the same as if phrases were not being used. At retrieval time, the effect of a phrase match was divided by 2. (The same effect could have been obtained by dividing the indexed weight of a phrase by sqrt(2).)

Indexing the document collection with phrases took 10.6 hours, creating an inverted file of 840 Mbytes. The actual retrieval took 2405 CPU seconds for all 50 queries and again considerably longer in elapsed time.

A more complicated local match criteria was used for the phrase run. The basic threshold was reduced from 100.0 (used in the single term run) to 75.0, but an additional restriction was placed on the match to ensure that no one term would contribute more than 65 percent of the computed pairwise sentence similarity for any sentence pair. This effectively eliminates sentence matches due to the presence of a single highly weighted term. The 65% was determined empirically from tests using the learning query/document sets; percentages ranging from 55% to 85% performed equally well.

The more complicated local match, in conjunction with the phrases, produced a very significant improvement in the phrase run as opposed to the single-term run. The 11-point average over 50 queries for the single-term run was 0.1738, while the phrase run did very well at 0.2032.

### Offical Routing Queries

Standard SMART relevance feedback techniques were used to automatically construct routing queries to be run on the test set of documents (D2).

Each routing query was composed of terms from the original indexed query plus the "best" 30 terms from the documents in the learning set that were relevant to that query. The weight of each routing query term was a linear combination of the tf x idf weight in the original query, the $tfxidf$ weight in each of the relevant documents, and the $tfxidf$ weight in a single non-relevant document. E. Ide's [4,10] feedback formula was used:

$$Q_{new} = Q_{old} + \sum_{rel}(w_q) - W_{q-in-nonrel}$$

The idf component in the query and document weights was based on occurrences of the term in the learning set of documents only.

The routing query was then run against each of the documents in the test set. Those documents were indexed in the standard SMART fashion, with each term receiving a *tf.xidf* weight, cosine normalization. Again, the idf document weight was determined by the occurrences of the term in the learning set of documents only. Thus no collection information from the test set of documents was used.

It took 306 seconds to construct the full feedback query set (most of the time spent deciding which terms should be added to each query). It took 1.9 hours to index D2, forming an inverted file, and then 293 seconds to run the 50 reformulated queries against the inverted file.

Effectiveness of this simple method was reasonable but not spectacular. The 11-point average over 50 queries was 0.1924.

### Tradeoff runs

This set of runs provide an examination of some of the tradeoffs (disk space, memory, time, and effectiveness) encountered within a single information retrieval system. There are many decisions that need to be made when designing a system; the goal in this set of runs is to explore the consequences of some fundamental choices including stopwords, stemming, phrases, and term weighting.

Conceptually, the standard SMART indexing and retrieval algorithms are given below.

```
INDEXING
    For each document/query text
        1.1 Break the text into tokens
        1.2 Determine if token is a common word (stopword) to be
            discarded.
        1.3 Stem all remaining tokens to their root forms.
        1.4 Assign concept numbers to each root, forming a
            ''vector'' of concepts.
        1.5 Weight each term in the vector.
        1.6 Store vector in an inverted file
RETRIEVAL
    For each term in the query
        2.1 Get the inverted list of documents containing that term.
        2.2 For each document/weight on inverted list
            2.3 Add Qi * Di to the partial similarity computed
                for this term so far (Qi is this terms query
                weight and Di the document weight)
            2.4 Add document to current list of top documents
                if similarity is high enough.
    Return list of top documents to the user.
```

Tradeoff runs

STANDARD

1. ntc.ntc (single terms) Full 2 pass indexing
2. ntc.ntc (single terms) alternate indexing method making document vectors

STOPWORD

3. ntc.ntc automatic stopword (added 69 terms occurring in 10\% of coll)
4. ntc.ntc automatic stopword (added 350 terms occurring in 5\% of coll)
5. ntc.ntc automatic stopword (added 1286 terms occurring in 2\% of coll)

STEMMING

6. ntc.ntc only plural stemming
7. ntc.ntc no stems

LOCAL/GLOBAL

*8. ntc.ntc local/global (single terms)
9. ntc.ntc local/global (single terms, same thresholds as 2nd official run)

QUERY OPTIMIZATION

10. ntc.ntc query efficiency optimization (15 docs guaranteed good)

PHRASES

11. ntc.ntc phrase dictionary. (> 25 times in D1, 158,000 out of 4.7 million)
*12. ntc.ntc local/global (phrases)

OTHER WEIGHTS

13. nnc.ntc (single terms)
14. lnc.ltc (single terms)
15. lnc.ltc (phrases)

| | Doc Indexing Time (hours) | Query Indexing Time (seconds) | Inverted File Size (Mbytes) | Other File Size (Mbytes) | Retrieval Speed 50 queries (seconds) | Retrieval-Effectiveness (averaged over 47 queries) 11-pt | NumRel Total | Recall/prec at 200 |
|---|---|---|---|---|---|---|---|---|
| 1. | 4.5/4.9 | 2.3(13.6) | 667 | 100 | 358 | 1813 | 3114 | 2614/3313 |
| 2. | 4.7/0.7 | 2.7 | "" | 790 | "" | "" | "" | "" |
| 3. | 4.3/4.6 | 3.2(13.1) | 624 | 100 | 306 | 1828 | 3101 | 2587/3299 |
| 4. | 4.0/4.3 | 3.0(12.8) | 528 | 100 | 166 | 1750 | 2978 | 2524/3168 |
| 5. | 3.7/3.9 | 2.8 | 381 | 100 | 78 | 1538 | 2658 | 2237/2828 |
| 6. | 4.3/5.0 | 2.7 | 724 | 98 | 251 | 1745 | 3148 | 2605/3349 |
| 7. | 4.2/4.7 | 1.6 | 752 | 98 | 235 | 1709 | 3101 | 2545/3299 |
| *8. | 4.7/0.7 | 2.7 | 667 | 790 | 1465 | 1783 | 3150 | 2636/3351 |
| 9. | "" | "" | "" | "" | "" | 1982 | 3400 | 2856/3617 |
| 10. | "" | "" | "" | "" | 97 | 1693 | 2983 | 2476/3173 |
| 11. | 7.5/8.0 | 3.8 | 892 | 104 | 415 | 1903 | 3298 | 2814/3509 |
| *12 | 9.7/0.9 | 2.7 | 892 | 1040 | 2405 | 2080 | 3555 | 3076/3782 |
| 13. | 4.5 | (88.5) | 667 | 89 | 262** | 1818 | 3203 | 2614/3407 |
| 14. | 4.5 | 2.7 | "" | "" | ? | 2249 | 3746 | 3272/3985 |
| 15. | 8.1 | | 892 | 104 | 396 | 2424 | 3886 | 3394/4134 |

* indicates official TREC run, **: timing on machine with 128 Mbyte memory
Query timing numbers in parenthesis indicate CPU time using dictionary on disk

The weighting scheme used in 1.5 determines whether the entire indexing approach can be done in one pass or requires two. The standard SMART weight which we recommend when nothing is known about the collection is a straight $tf * idf$ cosine normalized weight ($ntc$). Unfortunately, the "idf" value cannot be computed without knowing the document frequency of the term. Thus an accurate idf requires a two pass algorithm; the first finding the collection frequency of all terms and the second actually assigning the tf*idf weight. Alternative one pass weighting schemes are discussed at the end of the tradeoff discussion. Until then, all indexing runs discussed will be two pass runs, indexing the documents with an $ntc$ weight.

### Intermediate Document Vectors.

A two pass approach that uses minimal space involves doing steps 1.1 through 1.4 above on pass one, but instead of weighting and storing the actual vectors, just keep track of the collection frequency of each term. Then pass 2 repeats steps 1-4, but then can go ahead and compute the weight in 1.5 and store the vector. RUN 1 in Table 1 gives the timing figures for this approach: 4.5 hours for pass 1 and 4.9 hours for pass 2 (pass 2 takes longer because it needs to construct the inverted index).

An alternative to this approach is to store unweighted document vectors (as document vectors in not in inverted index form) for pass 1. Then pass 2 can ignore steps 1.1-1.4, and go directly to the weighting and inverted index construction. As RUN 2 shows, this is much quicker (4.7 hours for pass 1 and 0.7 hours for pass 2), but at a cost of doubling the amount of disk space needed.

Obviously the choice of these two approaches depends on whether indexing time or disk space is important to the database administrator.

### Stopwords

No retrieval system wants to store inverted indices for all words in the text (at least for retrieval purposes). Words like "the", "of", and "a" are not useful for distinguishing relevant documents and take up an extremely large amount of space since they occur in nearly every document. The question is how many stopwords to ignore. SMART has a standard collection independent list of 571 words that seem to convey little information about relevance. But individual collections often contain an additional number of words that give little information for the particular subject matter covered by that collection.

Three runs RUNS 3-5 were made on TREC, adding the most frequently occurring words occurring in TREC to the standard SMART stopword list. RUN 3 added the 69 terms occurring in more than 10% of the collection; RUN 4 added 350 terms occurring in more than 5% of the collection; and RUN 5 added the 1286 terms occurring in more than 2% of the collection. The space savings are substantial, ranging from 7% to 43% of the inverted file size, with a corresponding savings in indexing time. Retrieval time is even more affected, as many of the very long inverted lists for common words no longer have to be dealt with. RUN 4 saves 54% and RUN 5 79%.

The penalty that needs to be paid for these savings is the retrieval effectiveness. There's no penalty for RUN 3, and the reduced effectiveness in RUN 4 is insignificant, but RUN 5 loses about 15%. Except if you need maximal effectiveness, RUN 4 would seem to be worthwhile in practice.

One other potential problem with removing the most common words of the collection is user mystification. Users can understand that words like "the" don't help retrieval, but may be surprised when sentences like

> The head and president of an American computer system company based in Washington said she expected to make a million systems by the end of the year.

contain no indexable words at all! All words are among the standard SMART stopwords or occur in more than 10% of the TREC documents.

### Stemming

Stemming is one of those areas where the tradeoffs can be somewhat subtle[2.3]. The standard SMART approach uses full stemming where most suffixes are removed RUN 1, RUN 6 removes only plurals and RUN 7 does no stemming at all. An oft-cited drawback of not doing full stemming is the increase in the dictionary size; however that is reasonably insignificant. Far more important is the increase in inverted file size due to multiple forms of the same word occurring in a document. Plural indexing increased the inverted file by 8.5% and using no stems increased it by 12.7%

Indexing speed is given as an advantage of not doing full stemming, but again, that's reasonably insignificant. If full stemming is efficient, then the cost is almost completely counter-balanced by the cost of creating a larger inverted index. Retrieval speed is not normally mentioned as a disadvantage of full stemming, but seems to be a considerable factor. RUN 6 (plurals) is 30% faster than RUN 1 (full).

Retrieval effectiveness is often giving as an advantage of full stemming over just plural removal, but the results here agree with other recent results; the differences between the two are insignificant. No stemming at all is noticeably worse, but not by an extraordinary amount (6%).

### Local/global

The basic local/global algorithm is described in a previous section of this report. Our current implementation is designed to increase flexibility at the cost of retrieval time: for every query, we had to go out and index and weight 500 documents from scratch. That means at retrieval time, we can do any sort of indexing, weighting, and restrictions we like. In an operational system, however, it's expected that the local restriction operation would be determined in advanced, and would use preindexed sentence vectors. Thus indexing time and space would increase, but retrieval speed would go to a reasonable level (it currently takes 4 times as long for retrieval with local/global matching).

RUN 8 gives the timing and effectiveness figures for the first Cornell official run. Effectiveness for that run was disappointing; about the same as RUN 1 without local matching. However, between the time we submitted our first official run and the time of our second official run, we were able to get a better local restriction method working. Using the restriction method of the second official run (described in the main portion of the writeup), but on the single term collection (our second official run used phrases), we get a 10% improvement (RUN 9).

### Query Optimization.

In the stopword section above, the tradeoff between retrieval speed and retrieval effectiveness was examined by completely removing long inverted file stopword lists from the collection. This tradeoff can be examined directly at retrieval time by considering schemes to avoid looking at the longest inverted lists for query terms unless forced to.

The basic method used here, (described in more detail in [1]) is to sort the query by decreasing query weight (thus hopefully putting query terms with long lists and therefore low idf at the end), go through the query term-by-term, and stop when it is guaranteed that a certain number of "good" documents will appear in the final list of 200 top documents. Here, "good" means retrieved in the top 200 if all query terms are used.

If $X$ "good" documents are to be guaranteed out of 200 documents, the decision procedure invoked before each term $k$ of the query has been done is testing

$$(S(D_{200}) + \sum_{i=k}^{t}(q_i^2)) < S(D_X)$$

where $S(D_{200})$ is the similarity of the $200_{th}$ ranked document and $S(D_X)$ is the similarity of the $X_{th}$ ranked document. If this condition is true, then do not consider the rest of the terms in this query. This makes the assumption that the weight of a term in a document will be less than the weight of that term in the query. It's almost always true with *ntc* weights (except for very short documents).

RUN 10 in Table 1 gives one point on the curve; other points are in the table below.

| Guarantee | CPU Time | Elapsed Time | Rel Ret | 11-pt | Recall/prec at 200 |
|---|---|---|---|---|---|
| 1 | 71 | 141 | 2828 | 1575 | 2330/3009 |
| 5 | 83 | 146 | 2911 | 1630 | 2391/3097 |
| 10 | 88 | 154 | 2944 | 1659 | 2429/3132 |
| 15 | 97 | 178 | 2983 | 1693 | 2476/3173 |
| 25 | 106 | 218 | 3017 | 1721 | 2505/3210 |
| 50 | 123 | 255 | 3034 | 1751 | 2543/3228 |
| 75 | 144 | 291 | 3069 | 1765 | 2574/3265 |
| 100 | 167 | 347 | 3082 | 1782 | 2583/3279 |
| 150 | 224 | 459 | 3085 | 1778 | 2595/3282 |
| 200(Full) | 374 | 724 | 3144 | 1813 | 2614/3313 |

As X decreases, retrieval effectiveness and CPU time decrease pretty smoothly, with retrieval effectiveness remaining reasonable for quite a long time. Exactly which point is suitable for any particular application is determined by the relative priorities of efficiency and effectiveness.

### Phrase runs

The basic adjacency phrase approach used by SMART is described in the official run portion of the paper. We've looked at other methods of producing phrases: but our other implementations were too slow to be of use with TREC. Adjacency phrases have the advantage of being fast, simple, and producing reasonable results. They have the disadvantage that some sort of filtering operation has to be performed to come up with a good phrase list. There are just too many pairs of terms to index all of them. For these runs, we used the criteria that the phrase had to occur more than 25 times in D1, the learning document set.

We had hoped that phrases would help substantially in TREC since as the collection grows, the need to be more specific in the query grows, and phrases should be a good way of increasing precision. We got improvement, but it remained in the range of 5-8%, about what it is on the very small conventional test collections of the past. Perhaps other phrase approaches can do better.

Phrases are indexed with *ntc* weights, but the cosine normalization of the entire vector is done over the length of the single term subvector only. This means that the single terms end up with exactly the same weight as they would if the entire collection was indexed with only single terms. Thus, phrases only increase similarity. This seems to be quite important for some collections, although not crucial for our phrase selection on TREC.

In our runs, the global phrase run, RUN 11, does about 5% better in retrieval effectiveness than RUN 1, at a cost of increased indexing time, indexing space, and retrieval time. Similarly, the local/global phrase run, RUN 12, is about 5% better than the local/global single term run with the same parameters, RUN 9.

### Alternative Weighting Schemes

The two passes needed for idf weights in documents are a definite burden. In earlier experiments with other collections we found that not using idf in documents (while still using it in queries) was very reasonable, just a bit less effective than using idf.

When tried on TREC, tf-cosine normalized (*nnc*) document weights even proved to be marginally better than the *ntc* document weights. The one pass *nnc* weights, RUN 13, took less than half the total indexing time of RUN 1. There is no question that this is a major advantage of *nnc*. The possible advantage that *ntc* weights might have is in feedback, where normally query weights and document weights are combined to form new query weights.

A variant of term frequency weighting that's been in SMART for a couple of years is the *l* scheme (eg, *ltc* instead of *ntc*). *l* stands for log; (1.0 + ln (tf)) is used instead of tf, the number of times a term occurs in a document. The goal is to downweight the importance of the tf factor in collections which have very long documents. That fits TREC very well.

RUN 14 and RUN 15 describe using *lnc* document weights and *ltc* query weights for single terms and phrases respectively. They work remarkably well, about 20% better than the corresponding *nnc.ntc* runs. That's an enormous improvement.

Actually, half of the improvement is somewhat questionable. About 10%, out of the total 20%, is due to the *ltc* query weights, and the other half is due to the *lnc* document weights. The document weight improvement is reasonable; there's 770,000 documents of all sizes indexed with *lnc* weights. I feel the strong improvement due to *ltc* query weights is almost certainly an artifact of the TREC queries, and possibly even an artifact of the second query set (queries 51-100). 50 queries is a small enough number so that random effects can be important.

An average user supplied query will not have the distribution of terms that the TREC queries have. For that reason, *ltc* query weights can not be generally recommended. In tests on small collections, *ltc* performs about the same as *ntc*. It shouldn't hurt to use *ltc*, but don't bet the farm on it for TREC 2!

## Failure Analysis

There seems to be little consistent that can be said about the performance of Smart in the ad-hoc experiments. Smart does comparatively better (when compared with the median values) on queries with a lot of relevant documents, as opposed to those with few relevant documents, where it often does substantially worse. But it is hard to tell whether that is a feature of the system or the queries. For some queries, the Smart performance is very poor, because the query structure is ignored. That is especially true of queries using NOT clauses. The NOT is ignored and the following words are treated as positive relevance indications.

In general, the local match requirement does not have as big of an effect on queries as it has on other collections. There are definite successes; for example, query 69 on "Attempts to Revive the SALT II Treaty". The local requirement rejects all documents that deal with industrial salts insstead of a peace treaty. But in TREC 1 at least, there are few queries in which ambiguous words played an important part.

```
                Query 69 - Global Match Only
Num    Rel? Sim Title
349983   Y 0.48 REVIEW & OUTLOOK (Editorial): Breaking With SALT II</HL>
339345   Y 0.45 Letters to the Editor: Salt Ceilings Serve U.S. Interests</HL>
204128     0.36 [disposal of waste salt]
187056     0.33 [Superconducting compositions of the general formula]
370883     0.33 Diamond Crystal Peppered by Rivals Admits It's Licked --- Maker
582868     0.32 Salt Rationed in Many Parts of China</HEAD>
358376   Y 0.32 REVIEW & OUTLOOK (Editorial): No-Sweat SALT</HL>
232619     0.31 [process for recovering metals and metallic salts]
132873     0.30 [Salt deposits have economic significance]
206721     0.30 [interaction between intact salt and crushed salt]


                Query 69 - Local/Global Match
Num    Rel? Sim Title
349983   Y 10.48 REVIEW & OUTLOOK (Editorial): Breaking With SALT II</HL>
339345   Y 10.45 Letters to the Editor: Salt Ceilings Serve U.S. Interests</HL>
358376   Y 10.32 REVIEW & OUTLOOK (Editorial): No-Sweat SALT</HL>
342288   Y 10.27 [House banned funds for deployment of weapons]
 90352     10.27 [Arms control purposes include strengthening]
530499   Y 10.27 Arms Control Restrictions Figure In Pentagon Budget Battle</HE
167476   Y 10.25 [controlling the nuclear arms race]
353163   Y 10.23 REVIEW & OUTLOOK (Editorial): Is SALT Harmful?</HL>
534139   Y 10.23 House OKs Pentagon Spending Bill</HEAD> <NOTE>Eds: To update i
166087     10.23 [examines all the major arms control treaties]
340954   Y 10.22 House Passes Bill Slashing \$33 Billion From Reagan's Military
```

The routing run performs quite badly on a number of queries because of query expansion. The query length after terms from relevant documents are added is about twice the query length of the original query. A large portion of the added terms are general terms that have nothing to do with the query topic, but still get a high weight because of the number of relevant documents they occur in. Two approaches to try in the future are improving the query expansion process, and using local/global matching to ensure a retrieved document has something in common with the original query as well as the expanded query.

## Automatic versus Manual

One of the great unresolved debates of information retrieval is whether automatic approaches using no direct human expertise are better or worse than manual approaches, where human expertise is directly involved in fashioning a query. Long term, of course, the answer is obvious. By definition, the automatic approach is barred from using manual techniques, while the manual approach can use the best automatic approach and then add a bit of human knowledge on top of that. But absent such piggy-backing, the results from TREC 1 suggest for the time being the approaches are roughly equal. The best automatic runs and the best manual runs end up about the same.

As expected, the manual runs seem to do relatively better on precision and the automatic runs better on recall, but the effects are so small as to be insignificant. It is clear that the local match

requirement of the Cornell local/global approach, which is almost purely a precision enhancing device, still does not increase precision to a point that is as good as a manual search.

## Conclusion

The Cornell approach of using completely automatic methods to index and retrieve works extremely well. Requiring that some small part of a document (sentence) highly matches the query well gains about 10% in effectiveness. Using simple adjacency phrases in addition to single terms improves effectiveness by 5% to 8%.

There are a host of tradeoffs to be considered when designing and creating an information retrieval collection. Many of them produce surprising gains in efficiency, at only a minor cost in effectiveness.

## References


1. C. Buckley and A. Lewitt, Optimization of Inverted Vector Searches, Proc. Eighth Int. ACM/SIGIR Conference on Research and Development in Information Retrieval, Association for Computing Machinery, New York, 1985, 97-110

2. D. Harman, A Failure Analysis on the Limitation of Suffixing in an Online Environment, Proc. Tenth Int. ACM/SIGIR Conference on Research and Development in Information Retrieval, Association for Computing Machinery, New York, 1987

3. D. Harman, Towards Interactive Query Expansion, Proc. Eleventh Int. ACM/SIGIR Conference on Research and Development in Information Retrieval, Association for Computing Machinery, New York, 1988, 321-331

4. E. Ide, "New Experiments in Relevance Feedback", in The SMART Retrieval System - Experiments in Automatic Document Processing, ed G. Salton. Prentice Hall Englewood Cliffs, NJ. 1971, Chapter 16.

5. G. Salton, Developments in Automatic Text Retrieval. Science, 253. 30 August 1991, 974-980.

6. G. Salton, Automatic Text Processing – The Transformation, Analysis and Retrieval of Information by Computer, Addison-Wesley Publishing Co., Reading, MA 1989.

7. G. Salton and C. Buckley, Global Text Matching for Information Retrieval, Science 253:5023, 30 August 1991, 1012-1015.

8. G. Salton and C. Buckley. Automatic Text Structuring and Retrieval – Experiments in Automatic Encyclopedia Searching, Proc. Fourteenth Int. ACM/SIGIR Conference on Research and Development in Information Retrieval, Association for Computing Machinery, New York, 1991, 21-30.

9. G. Salton and C. Buckley, Term-weighting Approaches in Automatic Text Retrieval. Information Processing & Management, Vol 24 No 5, 1988, 513-523

10. G. Salton and C. Buckley, Improving Retrieval Performance by Relevance Feedback. JASIS, Vol 41 No 4, 1990, 288-297

11. L. Wittgenstein, Philosophical Investigations. Basil Blackwell & Co. Ltd., Oxford, England, 1953.

# Probabilistic Retrieval in the TIPSTER Collections: An Application of Staged Logistic Regression

*Wm. S. Cooper*
*Fredric C. Gey*
*Aitao Chen*

S.L.I.S., University of California
Berkeley, CA 94720

ABSTRACT: In this experiment the TIPSTER test collections were used as a vehicle for evaluating an approach to probabilistic retrieval for full-text documents. The methodology in question, called 'staged logistic regression,' involves two or more stages of logistic regression analysis of a learning sample of relevance judgements. The aim is to produce effective initial probability rankings of documents, without undue computational complexity at run time, by applying regression equations derived with the help of standard statistical software packages. In addition, the experiment explored the feasibility of using equations derived from training data for one document collection in a different document collection for which no training data happens to be available, and of calculating document relevance probabilities accurately enough so that they can be displayed as part of the output seen by the user. The regression equations were implemented as retrieval rules in an experimental prototype system obtained by modifying the SMART retrieval system.

## Introduction

The Berkeley group's interest in participating in the NIST/TREC Conference was stimulated by the opportunity it offered to gain experience with a methodology called 'Staged Logistic Regression.' This technique (hereinafter abbreviated 'SLR') is a systematic approach to retrieval system design based on probabilistic and statistical principles. It has been under study at Berkeley as a possible means of achieving effective probabilistic retrieval including acceptably accurate estimates of relevance probability without undue computational complexity.

In order to test out the SLR methodology on the TIPSTER data base in as straightforward a fashion as possible, attention was restricted to the problem of how to form the initial document ranking -- that is, the output ranking first offered by the system to the user in response to the user's original query. This problem should not be confused with the subsequent task of exploiting any relevance judgements that may be obtainable from the user once he or she has started down the output ranking and is actively examining documents. The latter problem -- the matter of how to exploit intra-search 'relevance

feedback' -- has been the subject of some fruitful probabilistic investigations, but the question of how to create the initial ranking seems equally significant.

**Objectives of the Experiment**

A principal goal of the experiment was to investigate the retrieval effectiveness of the SLR methodology in large collections. It was hoped especially that something could be learned of the soundness and retrieval power inherent in the statistical logic that underlies the SLR method. In view of this emphasis on logical foundations, and also because of the limited time and resources at the researchers' disposal, only a few simple and commonly employed frequency statistics were used as retrieval clues. No attempt was made to exploit more elaborate types of linguistic or locational evidence that would have required the incorporation of a parser, a conflator, a disambiguator, a thesaurus, a phrase identifier, etc. It is not that the latter kinds of evidence could not be exploited effectively under the SLR approach. Rather, in this particular experiment the idea was to see how far one could get on the basis of careful statistical logic alone.

Because of this 'mainly logic' approach, the results of the experiment must be interpreted with special care. It is not the absolute retrieval effectiveness of the system that is of most interest, but rather its retrieval effectiveness *relative to the amount of evidence used.* If an SLR-based system can achieve with less clues the same level of effectiveness that under other design approaches requires more clues, the objective of creating a powerful underlying retrieval logic will have been demonstrated. Because regression methods are hospitable to the use of almost any kind of predictive evidence that can be expressed in statistical form, there is little doubt that the performance of the system could have been improved through the use of additional clue-types. In considering the present experiment, however, the reader is asked to bear in mind the philosophy of building a generalizable logical platform capable of extracting a maximum of retrieval power from whatever clues *do* happen to be available.

Another objective of the SLR methodology is to keep the computational aspects of the retrieval reasonably simple and efficient. Some probabilistic schemes call for elaborate programming and are not impressively efficient at run-time. As a reasonable desideratum, a truly practical probabilistic method should be no more trouble to program, and not run substantially slower than, say, a vector-processing IR system using comparable types of evidence.

Still another goal of SLR is to partially replace traditional IR research procedures with more convenient and powerful standard statistical methods. For many years the customary experimental research paradigm in IR has been to conduct retrieval trials and apply specially invented IR effectiveness measures such as precision and recall to compare the trial results. It is reasonable to hope that much of this trial-and-error experimentation could be replaced by more efficient statistical regression analyses that use standard statistical software packages and standard measures of goodness-of-fit, thus bringing IR research more into the realm of mainstream statistical analysis.

The SLR design methodology requires the use of 'training data' ('learning trials', etc.) in the form of human relevance judgements for a sample of query-document pairs representative of the collection in which the proposed IR system is to operate. It is sometimes objected that methods requiring training data are useless in situations where a

system must be designed for a collection for which no training data is available, and as a practical matter none can be gathered. This objection has considerable force but it overlooks the possibility of extrapolating the results of a regression analysis in one collection for which training data exists into another for which it does not. As the experiment developed, it cast some light on the question of whether such an extrapolation can be effected without unacceptable loss of retrieval power.

A final objective of the SLR methodology is to produce estimates of relevance probability that are reliable enough to present to the system users as part of the ranked output they receive. Some IR research would appear to be premised on the notion that the output ordering of the collection is all that matters -- that the only purpose of generating retrieval status values ('similarity coefficients', 'ranking scores', etc.) is to achieve as effective a ranking as possible. We agree that imposing an effective order of presentation on the documents is the most essential single role of the retrieval status values, but feel that in addition the numeric scores are themselves a potentially important part of the output. Their significance lies in their ability to provide the user at each point in the search with information about whether it is likely to be worth while to continue the search down the ranking. Clearly, such scores will be most helpful if presented in a form that most users find readily interpretable, and interpretable moreover in a sense that bears as directly as possible on the decision of whether they should stop searching. Probability-of-relevance estimates would appear to fit this prescription admirably.

For reasons that will become apparent, this final objective was not attained in the present experiment. However, experiments in small collections indicate that SLR is capable of producing well-calibrated probability estimates, and doing so remains one of the general objectives of the methodology.

**The SLR Methodology**

The theoretical foundations of the SLR approach are presented in a recent paper by Cooper, Dabney, & Gey (1992). A synthesis and extension of earlier approaches to probabilistic retrieval, the SLR method combines the commonplace theoretical stratagem of invoking statistical simplifying assumptions with the empirical technique of applying statistical regression analysis to a learning sample. The use of statistical simplifying assumptions in IR has been explored by Maron & Kuhns (1960), Robertson & Sparck Jones (1976), Yu & Salton (1976), van Rijsbergen (1979) and others (surveyed by Maron (1984), Bookstein (1985)). Examples of the use of regression analysis are to be found for example in the work of Fox (1983), Fuhr (1989), and Fuhr & Buckley (1991).

A distinguishing characteristic of SLR is that it breaks the analysis of the retrieval process down into two or more distinct steps or stages. For the present experiment a simple two-stage procedure was adopted. In the first stage a learning sample was used to develop a regression equation that combines elementary retrieval clues into composite clues. In the second stage, the same empirical data is used to derive another regression equation that combines these composite clues into an estimate of the desired estimate of relevance probability for each query-document pair. Thus the evidence bearing on the retrieval decision is organized first into sets of simple properties of particular descriptors, and then into combinations of such sets as determined by the particular descriptors common to the query and document under consideration.

This split level approach allows for a natural separation of the available retrieval clues into two kinds. Statistical inferences based on properties of particular terms may be drawn first, while other kinds of evidence not confined to particular terms (e.g. document length, citedness, etc.) are saved for the second stage of statistical inference. An important virtue of this split-level approach is that the second-stage regression tends to correct for biases introduced by the statistical simplifying assumptions used to consolidate the results of the first stage.

### A 'Bare-Bones' SLR Methodology

Because the sole focus of interest in the experiment was the logic of the SLR approach, it seemed appropriate to keep all other design complications to a minimum. Except for the capacity to perform the two-level probabilistic computations requisite for SLR, therefore, the experimental system was kept as simple and automatic as possible. Thus no phrase discovery, part-of-speech tagging, disambiguation, or other linguistically sophisticated operations were incorporated, nor was a thesaurus included for the conflation of synonyms or other purposes, nor was the descriptor vocabulary structured in any way. There was no clustering, no knowledge base, no set of implicative rules, no network, nor anything else 'AI-like.' All indexing was performed extractively without benefit of human intervention. No use was made of the manually assigned descriptors in the document collections that had them.

The experimental retrieval system was implemented by modifying the SMART system (Version 10), a suite of IR programs generously provided to the IR research community by researchers at Cornell University. Since the new model to be implemented was probabilistic, all features of SMART motivated by the vector space retrieval model were left unused or replaced by corresponding probabilistic operations. The SMART stop list was used as-is except for the addition of a few common words from the query vocabulary thought unlikely to be content-indicative (e.g. 'document', 'contains'). The SMART system's stemmer was used without modification to strip suffixes and endings off of all query and document words that survived the stop list. The search algorithm used a slightly extended form of SMART's inverted file. The retrieval speed and general programming complexity of the SMART system were left substantially unaffected by this conversion to SLR-based retrieval, which meant that the objective of run-time efficiency and complexity roughly comparable to that of a vector processing system had been achieved.

### The Design Equations

At the heart of the design are four statistical equations. Taken together they are capable of supplying an estimate, for any query-document pair of interest, of the probability that the document in question is relevant to the query in question. We shall take them up in their natural order of application.

Some preliminary vocabulary: A 'logodds' may be regarded simply as a probability re-expressed on a special scale. The odds $O(E)$ of an event $E$ is by definition $\frac{P(E)}{P(\overline{E})}$. The conditional odds $O(E_1 \mid E_2)$ is $\frac{P(E_1 \mid E_2)}{P(\overline{E_1} \mid E_2)}$. The 'logodds' of $E$, sometimes also

called a 'logit', is $\log O(E)$, or in the case of conditional odds, $\log O(E_1 \mid E_2)$. Natural logarithms will be used throughout the sequel.

The first equation estimates the logodds that a document is relevant to a query, given just one composite clue relating query to document. For the TREC experiment a composite clue was defined to be a set of six frequency properties of a particular stem match, where a 'stem match' is understood to be the event that some word stem has been found to occur at least once in both the query and document. Consider a composite clue $A_i$ consisting of the six elementary properties $X_1, \ldots, X_6$ that describe some stem match. Let $R$ denote the possible event that the document under consideration is in fact relevant to the query under consideration. The first equation, derived by logistic regression from a learning sample for the Wall Street Journal (WSJ) collection, is

$$\log O(R \mid A_i) = \log O(R \mid X_1, X_2, X_3, X_4, X_5, X_6)$$

$$\approx -7.08 + .38\, X_1 + .04\, X_2 + .77\, X_3 - .07\, X_4 + 1.05\, X_5 + .23\, X_6 \qquad (1)$$

where

$X_1$ = the log of the absolute frequency of occurrence of the stem in the query; i.e. a simple count of its occurrences in the query, logged.

$X_2$ = the log of the relative frequency of occurrence of the stem in the query; i.e. of $X_1$ divided by the query length, with query length defined as the total number of occurrences of all word stems in the query.

$X_3$ = the log of the absolute frequency of occurrence of the stem in the document.

$X_4$ = the log of the relative frequency of occurrence of the stem in the document.

$X_5$ = the log of the inverse document frequency of the stem in the collection; i.e. the proportion of documents containing at least one occurrence of the stem, inverted and logged.

$X_6$ = the log of the global relative frequency of the stem in the collection; i.e. the fraction of word occurrences in the entire collection that are occurrences of the stem in question, logged.

The equation says roughly that if a TIPSTER query and a WSJ document were chosen at random, and a certain stem were found to occur in them both with frequency statistics $X_1, \ldots, X_6$, then in the absence of other knowledge it would be appropriate to apply this formula to estimate the logodds that the document is relevant to the query.

Next, suppose a query and document have $N$ terms in common, leading via Eq. (1) to $N$ different logodds estimates, one for each of the term matches $A_1$ through $A_N$. The second equation allows these estimates to be combined into a preliminary estimate of the logodds that the document is relevant to the query. It has the form

$$\log O(R \mid A_1, \ldots, A_N) = \log O(R) + \sum_{i=1}^{N} [\log O(R \mid A_i) - \log O(R)] \qquad (2)$$

The only quantity in the right side that cannot be estimated using Eq. (1) is the prior logodds $\log O(R)$. The value -6.725 was used for this parameter.

Eq. (2) follows from the 'Assumption of Linked Dependence.' Considering for simplicity only the special case of two properties, this assumption can be expressed as the equality

$$\frac{P(A_1, A_2 \mid R)}{P(A_1, A_2 \mid \bar{R})} = \frac{P(A_1 \mid R)}{P(A_1 \mid \bar{R})} \frac{P(A_2 \mid R)}{P(A_2 \mid \bar{R})}$$

Intuitively this asserts that the degree of dependency that exists between properties $A_1$ and $A_2$ in the set of relevance-related query-document pairs is linked in a certain way with the degree of dependency that exists among the nonrelevance-related pairs. It is a weaker assumption than the independence postulates commonly encountered in the literature on probabilistic IR. For discussion and a derivation of Eq. (2) from the Linked Dependence Assumption see Cooper (1991) and Cooper, Dabney & Gey (1992 op. cit.).

The role of the third equation, as developed for this experiment at least, is to correct for deficiencies in the second equation. There are two major sources of distortion to contend with. One is that the validity of Eq. (2) depends on the Linked Dependence Assumption, a simplifying assumption that is at best only approximately true. Especially when the number $N$ of term matches is large, Eq. (2) (if uncorrected) is capable of grossly overestimating the logodds of relevance. The other source of distortion is that Eq. (2) as it stands fails to take into account the fact that longer documents will tend to produce more term matches than shorter ones simply by chance. If nothing were done to correct it, longer documents could receive much higher relevance probability estimates than shorter ones merely by virtue of their length.

This latter failing is related to a subtle criticism that can be raised against the policy of using only term matches, never mismatches, as the composite clues. Actually, a term match (i.e. term present in both query and document) is only one of four conditions that might obtain for a term vis-a-vis a given query and document. The others are: ii) term present in query but absent from document; iii) term present in document but absent from query; and iv) term absent from both query and document. Had clues of other types been recognized, accorded their own regression equations, and allowed to make their own contribution to $Z$, it might not have been necessary to make any correction for document length. But because we have taken the computationally convenient shortcut of ignoring all evidence of types ii) - iv) at the first stage of the analysis, we must compensate somehow at the second stage for the distortion caused by that oversimplification.

The corrective equation, as developed for the WSJ collection through another application of logistic regression to the learning sample, is

$$\log O(R \mid A_1, \ldots, A_N) \approx -6.08 + 3.63 \log \max(Z, 1) - 1.45 \log L \qquad (3)$$

where

$Z$ = the value of the summation expression in the right side of Eq. (2);

max( $Z$, 1 ) is the larger of Z and 1; and

$L$ = the length of the document under consideration, expressed as the total number of stem occurrences in the document counting separate occurrences of the same stem separately.

This estimate of $\log O(R \mid A_1, \ldots, A_N)$ is understood to modify and supercede the estimate produced by Eq. (2).

Since logodds are monotonically related to probabilities, a retrieval system could in principle probability-rank the documents of the collection for the user simply by presenting them in descending order of the logodds values assigned to them by Eq. (3). However, since probabilities are easier for users to interpret than logodds, the conditional logodds estimates of form $\log O(R \mid A_1, \ldots, A_N)$ produced by Eq. (3) were translated into ordinary conditional probability estimates with the help of a fourth equation, the identity

$$P(R \mid A_1, \ldots, A_N) = \frac{1}{1 + e^{-\log O(R \mid A_1, \ldots, A_N)}} \tag{4}$$

The probability estimates so obtained for the TIPSTER data appear in the 'score' column of the rankings submitted by the Berkeley group.

## Computational Arrangements

The computations required for storage and retrieval under the SLR methodology follow roughly the order of the four equations. Documents are indexed as follows. First, an incoming document is put through preparatory operations that include the removal of markup and other unwanted information, the deletion of stop words, and the stemming of the remaining words. Then for each stem, all stem statistics that can be calculated before the query is known -- specifically, $X_3$, $X_4$, $X_5$, and $X_6$ -- are collected. Finally, these statistics are used to compute the stem's indexing weight in the document using the formula

$$\textit{Doc stem weight} = -7.08 + .77\, X_3 - .07\, X_4 + 1.05\, X_5 + .23\, X_6 - (-6.725)$$

This formula is just the right side of Eq. (1) without the terms for $X_1$ and $X_2$, and with the value of the prior logodds $\log O(R)$ subtracted off in preparation for the application of Eq. (2). The result is stored in the inverted file as the weight of the term for this document.

Query indexing is similar. After an incoming query has been stop-listed and stemmed, for each stem the two statistics $X_1$ and $X_2$ that are dependent upon query properties are calculated. The stem is then assigned as its weight in the query the value

$$\textit{Query stem weight} = .38\, X_1 + .04\, X_2$$

This formula comprises the rest of the right side of Eq. (1).

To effect retrieval, the query is compared against each document with which it shares at least one stem. For each stem contained in both query and document, the query stem weight and the document stem weight are added, resulting in an estimate of $\log O(R \mid A_i) - \log O(R)$ for the stem. Summing these estimates for individual stems over all the match stems yields the value of the summation expression in the right side of Eq. (2). This computation is analogous to the calculation of a vector product in the vector space model, except that corresponding query and document weights are added instead of multiplied.

Calling this sum $Z$, Eq. (3) is applied to obtain the estimate of the logodds that the document is relevant to the query. Using Eq. (4), this logodds is translated into a probability, and the documents are sorted for the user in descending order of their estimated probabilities.

**Constructing the Sample**

The SLR method calls for the use of a learning sample of relevance judgements that can be assumed accurate. Among other requirements the sample must (a) be sufficiently large and include a sufficient number of queries; and (b) have a definite, complete statistical structure of some kind (e.g. that of a random or a stratified sample) that can serve as a basis for making statistical inferences. Arguably the TIPSTER data supplied for TREC experimentation fulfilled condition (a) for one or two of the five collections (WSJ and perhaps ZIFF). However, so far as could be ascertained from information made available to participants, the second condition was not satisfied for any of the five collections. The samples of relevance data that were provided had no apparent statistical structure of the kind upon which statistical reasoning is ordinarily based. The omission is understandable but unfortunate from the point of view of probabilistic retrieval.

For the present experiment the lack of a known statistical sample structure was highly problematical and almost fatal. It meant that the method of interest could not be rigorously applied to the available data. The theory on which SLR is based consists of a careful chain of statistical reasoning. If one of the links is weak, scientifically sound estimates of relevance probability cannot be expected.

Rather than abandon the experiment entirely, however, the following stopgap measure was adopted. An arbitrary assumption would be made about the structure of the sample, and the analysis would be carried out on the basis of this fictitious assumption as though it were known to be true. It was thought that this desperation measure would at least allow the investigators to gain experience in applying the method to a large data set (albeit on a hypothetical basis), and to illustrate the methodology for other interested parties. Also, it was thought that the fictitious assumption could be chosen in such a way that the experimental evidence gathered about the comparative worth of various retrieval clues would probably have at least some value. Finally, it was hoped that the output orderings might be reasonably effective in spite of the likely miscalibration of the retrieval status values as probability estimates.

While this policy allowed participation in the venture to continue, the artificiality of the constructed sample diminished to some unknown extent the retrieval effectiveness of the prototype system. This should be remembered when interpreting the results of the experiment: they do not fairly represent the SLR methodology's capabilities. Had the investigators been in a position to design their own sampling procedure, the results would presumably have been better. In the same spirit it should be kept in mind that the level of accuracy of the probability estimates themselves cannot be taken as indicative of the reliability of SLR.

The arbitrary assumption that was settled upon for the sample construction was that for the WSJ collection the universe of all query-document pairs is separable into two sets: one a small set rich in relevance-related pairs, the other a large set containing no relevance-related pairs. The supplied judgements of relevance and irrelevance for the WSJ

were to be treated as though they were a random sample of one out of every ten members of the first of these two sets. This implies among other things the supposition that the professional searchers had found 10% of the relevance-related pairs that could have been found had the entire collection been examined for every query. The 10% figure was an unsupported guess.

This working assumption made it possible to construct a two-part (stratified) sample for the WSJ collection. The first portion or stratum of the sample consisted of the set of 2194 query-document pairs for which human judgements of relevance and nonrelevance were available. In the subsequent regression analysis, each of these was weighted by a factor of 10, reflecting the assumption stated above. The second part of the constructed sample consisted of a set of 1687 query-document pairs drawn essentially randomly from among all query-document pairs that shared at least one stem in common. (The latter sample was obtained by considering all query-document pairs constructible from the 52 training queries and the documents in the WSJ training collection, ordering them by query number and within query number by document number, choosing every 2444th pair from this ordering, and discarding any pairs so chosen for which there was no overlap between query and document. The result may be considered a random sample, though technically it was slightly preferable to a random sample isofar as it was stratified by query.) The pairs in this second set were assumed to lie outside the first set and to be nonrelevance-related (though not verified this assumption was probably approximately true). Each pair was accorded a weight of 2400 in the subsequent regression analysis.

It may be worth reiterating that the necessity of specifying the number of relevance-related pairs by sheer guesswork, together with the general artificiality of the constructed sample, dashed all hope of producing well-calibrated estimates to present to the users.

**The Regression Analysis**

As the first stage of the regression analysis, for each query-document pair in the sample the set of all stems shared in common by the query and the document was assembled. This resulted in an expanded sample of 30,234 query-document-stem triples. For each triple, the values of the six statistics $X_1, \ldots, X_6$ were calculated and recorded along with the relevance judgement associated with the query and document in question. A weighted logistic regression analysis was performed on this data, with $X_1, \ldots, X_6$ serving as the independent variables and the binary relevance judgements as the dependent variable. The result was the set of coefficients in Eq. (1). In other words, Eq. (1) is the regression equation that was found to be the logistic equation of maximum likelihood (the 'best fit') for the data in the sample.

How were the six retrieval clue-types chosen? Actually the aforedescribed regression analysis was performed repeatedly for various types and combinations of independent variables before these six were settled upon. Of those tried out, these turned out in combination to exhibit the most predictive power -- that is, to offer the best prospects of yielding useful estimates of logodds of relevance.

'Predictive power' was judged according to several interrelated criteria. One was the extent to which a model based on a clue- combination under investigation was found to fit the data, as measured by the -2 Log Likelihood statistic or one of its minor variants

such as the Akaike Information Criterion. Another was the extent to which the ordering imposed on the triples by the logodds estimates assigned to them by the model resembled the ideal ordering in which all relevant triples precede all nonrelevant triples. This was measured statistically using various rank correlation coefficients including Kendall's Tau-a and the Goodman-Kruskal Gamma. Still another property that was taken into account was the shape of a variable's graph when the observed logodds was plotted against the variable values arranged along the X-axis in deciles. In such a graph, a shape that roughly resembles a straight line is considered desirable. In some cases it was found that pretransforming a variable by taking its logarithm helped to produce such a straight line. This was in fact one of the motivations for logging the variables. Another motive for doing so was the observation that it seemed to improve the general fit of the model as measured by the other indicators.

Fortunately, the various criteria used to optimize the choice of variables were rarely in qualitative conflict. But there was insufficient time to explore all variables of potential interest, or to investigate more elaborate possibilities such as interaction terms. All regression analyses were run using the SAS statistical package (Version 6.06) on an IBM 3090 mainframe computer with accelerated math capabilities. The SAS package automatically supplies most of the diagnostic statistics mentioned above. Detailed discussions of logistic model building can be found in works on logistic regression (Hosmer & Lemeshow 1989; Collet 1991).

It is worth noting that all the choices among possible predictor variables were made, and their weights in the equation determined, as part of the regression analysis. There was no recourse to traditional retrieval trials based on precision, recall, and the like. The regression procedures were found to be more convenient and efficient than experimentation of the usual kind in which there is no way other than trial-and-error to converge on optimal numeric coefficients. Ideally one might think of combining the two techniques -- that is, of using traditional methods to confirm the findings of the regression studies. However, time pressures prevented the pursuit of that luxury.

Next, Eq. (1) was applied to each triple in the sample to calculate the estimate of the logodds of relevance $\log O(R \mid A_i)$ for the query and document in question, given the characteristics of the match stem. Then in accordance with Eq. (2) the estimated prior logodds of relevance $\log O(R) = -6.725$ was subtracted from each of these estimates, and the differences summed within each query-document pair to obtain the value of $Z$ for the pair.

For the second-stage regression, the value of $Z$ calculated as just described, together with the length $L$ of the document, were recorded for each query-document pair in the sample. From these the value of $\frac{\log Z}{L^{0.4}}$ was calculated for each pair. (To ensure that the logarithm would always be well defined, $Z$ was first replaced by $\max(Z, 1)$ .) Using this ratio as the independent variable and the binary relevance judgements as the dependent variable, a second weighted logistic regression was run. This produced two coefficients -- an intercept and a slope -- specifying a regression equation. Simple manipulations were used to transform it into the form displayed earlier as Eq. (3).

Here are some of the considerations that led to the use of $\frac{\log Z}{L^{0.4}}$ as the variable on

which to regress. If one could trust the Linked Dependence Assumption completely, and if nonmatch as well as match events had been taken into explicit account in the computation of $Z$, one might have tried letting $Z$ stand as-is as the desired logodds estimate. But because such is not the situation, this would fail to correct for dependency distortion and would give longer documents an unfair advantage. Thus it seemed advisable to perform a corrective linear transformation on $Z$, and moreover to normalize $Z$ first by dividing it by some simple function of document length. It was found by trial and error that dividing $Z$ by $L$ raised to a power of around 0.4 seemed to remove most of the visible bias toward either very short documents or very long documents in the five collections. Logging the entire expression was found to improve the fit to the sample data.

It would have been appropriate to include a correction for query length analogous to the one developed for document length. However, for lack of time the necessary analysis could not be carried out.

**Extrapolation to Other Collections**

The regression analysis was confined to the WSJ data because relevance judgements were not available for most of the other collections in sufficient quantities, or for enough of the training queries, to make regression feasible. This circumstance brought with it the problem of how to extrapolate the WSJ retrieval rules to the remaining four collections.

Speaking generally, the extension of retrieval formulae to other collections is a significant problem throughout the IR field. One would like to know how to transfer design parameters from one collection, for which there is enough relevance data, to another collection for which there may be too little data or none. If the transfer could be accomplished without too much loss of predictive power, the almost exclusive use by IR experimenters of special 'test collections' could be justified more easily. We welcomed the dearth of TIPSTER relevance data for some of the collections as an opportunity to explore this problem. To simplify the challenge and confront it in its starkest form, we elected to ignore entirely even such data as were available for collections other than WSJ.

The method used for the extrapolation was based on the well known statistical concept of standardization of variables. The standardized value of a variable in a population is obtained by subtracting from its observed value the variable's mean value in the population, then dividing this difference by its standard deviation in the population. The new standardized values have a mean of zero and a standard deviation of one. The working assumption that was made was that a regression equation such as Eq. (1) can be carried over and applied in another collection provided all variables involved have first been standardized in both collections.

Although no variable values were actually standardized, the coefficients in Eq. (1) were recalculated for each of the other four collections in such a way as to create the same effect. The values for the population means and standard deviations used in the recalculation of the coefficients were taken from random samples of triples taken from the five collections. The samples were comparable to those in the 'random' subsample of WSJ query-document triples described earlier.

The algebraic details of the transformation process will not be presented here, but the resulting modifications of the right side of the earlier WSJ form of Eq. (1) may be of

interest. They are

$$\textit{For AP:} \quad -7.21 + .40\,X_1 + .04\,X_2 + .88\,X_3 - .10\,X_4 + 1.09\,X_5 + .25\,X_6$$

$$\textit{For DOE:} \quad -7.51 + .44\,X_1 + .05\,X_2 + 1.18\,X_3 - .12\,X_4 + .94\,X_5 + .24\,X_6$$

$$\textit{For FR:} \quad -6.83 + .44\,X_1 + .04\,X_2 + .57\,X_3 - .06\,X_4 + 1.13\,X_5 + .24\,X_6$$

$$\textit{For ZIFF:} \quad -6.95 + .38\,X_1 + .04\,X_2 + .68\,X_3 - .07\,X_4 + 1.03\,X_5 + .21\,X_6$$

As an illustration of what the transformation of coefficients has accomplished, one sees that the coefficient for $X_3$, the absolute frequency of the match term in the document, is largest for the DOE collection and smallest for the FR collection. This serves to compensate for the fact that the average document length is smallest in DOE and largest in FR.

No modifications were made of the coefficients in the second-stage regression equation, Eq. (3), when applying it in other collections. Because adjustments had already been made for inter-collection differences in the first-stage equation, the investigators were not convinced that further adjustments would be profitable at the second stage. Indeed we are not entirely confident that the adjustments are a good idea even for all the variables at the first stage, but thought it worth the experiment.

**Effectiveness Scores**

In the TREC evaluations the 11-point average recall calculated for ad hoc retrieval by the Berkeley system was 0.151; the average number of relevant documents retrieved in the top-ranked 100 documents for each request was 40.8; and the average number of relevant documents retrieved in the top-ranked 200 documents for each request was 67.9. The comparable three figures for the medians of all systems submitting results for ad hoc retrieval were .157, 39.5, and 62.5 respectively. Comparing the Berkeley system's scores against the median scores it can be seen that for the number of relevant documents retrieved among the top 200, the Berkeley system exceeded the median by 5.4 documents, an amount that can be shown (in a paired comparison t-test over the mean of the differences for the 50 topics) to be statistically significant at the 0.05 level. The other two scores do not differ from the corresponding median scores by what are customarily regarded as statistically significant amounts. Thus by one of the three measures the SLR experimental system was significantly more effective than the median system, and by the other two it was not significantly different from them.

To put these results in perspective it should be remembered that unlike other systems included in the comparison group, the SLR system was primitive in every respect except for its statistical logic. It involved no human reformulation of topics or other manual intervention, it used no relevance feedback, and it employed no special linguistic or other devices such as parsing systems, thesauri, disambiguation, phrase identification, or global/local combinations of evidence. It even forewent the use of training data from four of the five collections. Yet by a statistically careful use of simple frequency information alone, it was able to hold its own against typical systems that used much more elaborate forms of evidence.

Since the SLR methodology is hospitable to the introduction of additional clue-types, and indeed might be expected to wring a maximum amount of leverage out of them, the prospect for future, less primitive, SLR systems that combine many types of evidence seems promising.

### Future Possibilities

Though the prototype system described here used only a few simple statistical clues, the SLR approach is general and in principle flexible enough to accommodate most of the clue-types that researchers have been interested in as predictors of relevance. Broadly speaking, retrieval evidence having to do with particular index terms lends itself to exploitation in the form of variables in the first-stage regression equation, while other kinds -- properties of the entire query, entire document, or their relationship -- can be accommodated as variables in the second stage.

As an example of possible new evidence at the first stage, suppose by virtue of parsing, suffix analysis, or dictionary lookup some information is available about the parts of speech of the match stems in the query and document. Then an additional categorical variable might be introduced into the first-level regression analysis to represent the match stem's part of speech in the document, on the hunch that some parts of speech (e.g. nouns) should be more heavily weighted than others. The general two-level form of the analysis would remain the same. A further possibility would be to introduce a variable to represent the event that the part of speech of the stem as it occurs in the query is the same as its part of speech in the context in which it occurs in the document.

Further clues could be introduced at the second stage. In the present experiment the only retrieval evidence introduced at the second stage that was not already present in the first was the document length $L$, which was intended more as an antidote to a bias in $Z$ than as an independent predictor of relevance in its own right. But nothing prevents any helpful relationship between the query and document from being brought to bear. As an example, suppose a measure of the mutual closeness of the query's match stems in the document is to be introduced on the hypothesis that the closer together the query stems tend to occur in the document, the likelier it is (other things being equal) that the document is relevant (Keen 1992). Such a measure of proximity could be added as a new variable in the second-level equation, with no other change being needed in the underlying statistical framework.

### Conclusions

1. The TREC results indicate that the SLR methodology is capable of achieving a respectable degree of retrieval effectiveness even when the retrieval evidence is confined to a few simple frequency clues. ('Respectable' in this context means competitive with the median performance of other systems most of which use more elaborate evidence.) Since nothing prevents the incorporation of additional clue types into future SLR systems, and the regression procedure should help to combine them with existing clues in an optimal way, the outlook for the retrieval effectiveness of the SLR approach seems promising.
2. The prototype SLR system demonstrates that a probabilistic initial ranking can be achieved with a run-time efficiency approximately equivalent to that of a vector

processing system utilizing similar kinds of evidence.

3. Standard statistical program packages can be used for an SLR analysis even in large collections, circumventing much of the need for retrieval trials of conventional design and allowing the choice of variables and the determination of optimal numerical parameters to be carried out more conveniently.

4. The TREC results suggest that use of the SLR approach need not necessarily be ruled out because of a lack of training data for the particular document collection to which it must be applied. A respectable level of effectiveness can (under at least some conditions) be achieved through the extrapolation into the new collection of regression equations derived from a collection for which training data already exists.

5. The present experiment failed to demonstrate that the SLR method is capable of producing, in large collections, probability of relevance estimates sufficiently well-calibrated to be presented to the users as part of the output display. It is suspected that this failure is associated with the peculiar limitations of the training data supplied for this initial TREC conference. If so, use of the more extensive training data to be made available for future conferences may be sufficient to resolve this problem.

**Acknowledgements**

The theory of staged logistic regression, developed in collaboration with Dan Dabney of U.C.L.A, was originally stimulated by discussions with James Allen and Gerard Salton of Cornell University and Stephen Robertson of City University, London. The computer science department at Cornell University provided a hospitable environment for the early stages of the theoretical development. Ray Larson at U.C. Berkeley contributed experienced advice on the conversion of SMART and on general systems problems. We are indebted to Chris Buckley for supporting our efforts to make SMART regressive, and to all past contributors to SMART for making this valuable research tool available. The work stations used for the experiment were DEC 5000's supplied by the Sequoia 2000 project at the University of California, a project principally funded by the Digital Equipment Corporation. A DARPA grant supported the programming effort.

**References**

Bookstein, A. Probability and fuzzy set applications to information retrieval. In M. Williams (ed.), *Annual Review of Information Science and Technology,* 20, White Plains, NY: Knowledge Industry Publications. 1985.

Collett, D. *Modelling Binary Data.* London: Chapman & Hall; 1991.

Cooper, W. S. Exploiting the maximum entropy principle to increase retrieval effectiveness. *Journal of the American Society for Information Science,* 34(1): 31-39; 1983.

Cooper, W. S. Inconsistencies and Misnomers in Probabilistic IR. *Proceedings Fourteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.* Chicago: 57-62. October 1991.

Cooper, W. S.; Dabney, D.; Gey, F. Probabilistic retrieval based on staged logistic regression. *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.* Copenhagen: 198-210; June 1992.

Cooper, W. S.; Huizinga, P. The maximum entropy principle and its application to the design of probabilistic retrieval systems. *Information Technology: Research and Development,* 1(2): 99-112; 1982.

Fox, Edward A., *Extending the Boolean and Vector Space Models of Information Retrieval with P-Norm Queries and Multiple Concept Types,* Ph.D. Dissertation, Computer Science, Cornell University, 1983.

Fuhr, N. Optimal polynomial retrieval functions based on the probability ranking principle. *ACM Transactions on Information Systems* 7(3): 183-204; 1989.

Fuhr, N.; Buckley, C. A probabilistic learning approach for document indexing. *ACM Transactions on Information Systems,* 9(3): 223-248; 1991.

Harper, D. J.; Van Rijsbergen, C. J. An evaluation of feedback in document retrieval using co-occurrence data. *Journal of Documentation,* 34(3): 189-216; 1978.

Hosmer, D. W.; Lemeshow, S. *Applied Logistic Regression.* New York: Wiley; 1989.

Kantor, P. Maximum entropy and the optimal design of automated information retrieval systems. *Information Technology: Research and Development,* 3(2): 88-94; 1984.

Keen, E. M. Term position ranking: Some new test results. *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.* Copenhagen: 66-76; June 1992.

Lee, J. J.; Kantor, P. A study of probabilistic information retrieval systems in the case of inconsistent expert judgement. *Journal of the American Society for Information Science,* 42(3), 1990.

Maron, M. E. Probabilistic Retrieval Models. In B. Dervin and M. Voigt (Eds.), *Progress in Communication Sciences,* Vol. V, Ablex, 1984, pp. 145-176.

Maron, M. E.; Kuhns, J. L. On relevance, probabilistic indexing, and information retrieval. *Journal of the Association for Computing Machinery,* 7(3): 216-244; 1960.

Robertson, S. E; Bovey, J. D. *Statistical problems in the application of probabilistic models to information retrieval.* British Library Research and Development Department, Report No. 5739, November 1982.

Robertson, S. E.; Sparck Jones, K. Relevance weighting of search terms. *Journal of the American Society for Information Science,* 27(3): 129-146; 1976.

van Rijsbergen, C. J. A theoretical basis for the use of co-occurrence data in information retrieval. *Journal of Documentation,* 33(2): 106-119; 1977.

van Rijsbergen, D. J. *Information Retrieval* (2nd ed.). London: Butterworth & Co. Ltd; 1979.

Yu, C.T.; Buckley, C.; Lam, H.; Salton, G. A generalized term dependence model in information retrieval. *Information Technology: Research and Development,* 2: 129-154; 1983.

Yu, C.T.; Salton, G. Precision Weighting -- An Effective Automatic Indexing Method. *Journal of the Association for Computing Machinery,* 23(1): 76-88; 1976.

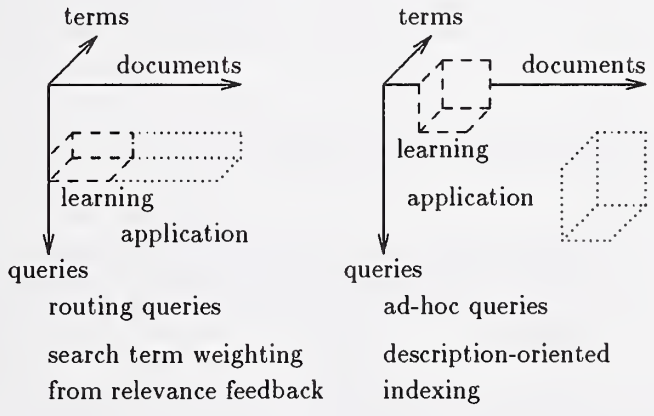
# Optimizing Document Indexing and Search Term Weighting Based on Probabilistic Models

Norbert Fuhr* Chris Buckley†

**Abstract**

We describe the application of probabilistic indexing and retrieval methods to the TREC material. For document indexing, we apply a description-oriented approach which uses relevance feedback information from previous queries run on the same collection. This method is also very flexible w.r.t. the underlying document representation. In our experiments, we consider single words and phrases and use polynomial functions for mapping the statistical parameters of these terms onto probabilistic indexing weights. Based on these weights, a linear (utility-theoretic) retrieval function is applied when no relevance feedback data is available for the specific query. Otherwise, the retrieval-with-probabilistic-indexing model can be used. The experimental results show excellent performance in both cases, but also indicate possible improvements.

## 1 Learning in IR




routing queries — ad-hoc queries

search term weighting from relevance feedback — description-oriented indexing

Abbildung 1: Learning approaches in IR

Figure 1 shows two major learning approaches that are used in IR, both of which are applicable to the tasks to be performed within TREC. For the routing queries, we have relevance feedback data for some documents w.r.t. a specific query, and then the system has to rank further documents for the same query. As indicated by the third dimension, our knowledge is restricted to the terms we have

*University of Dortmund, Informatik VI, P.O. Box 500500, W-4600 Dortmund 50, Germany, fuhr@ls6.informatik.uni-dortmund.de

†Department of Computer Science, Upson Hall, Cornell University, Ithaca, NY 14853, USA, chrisb@cs.cornell.edu

seen in the learning sample. If the new documents would contain totally different terms, the learning sample would be of no use for coping with these documents. For dealing with the routing queries, mostly search term weighting methods are applied, which are well established in IR.
For the ad-hoc queries, the task is more difficult: Given relevance information for some query-document pairs, this information has to be exploited in order to rank new documents w.r.t. new queries; furthermore, most of these new query-document pairs will involve new terms. As a method for dealing with this type of task, description-oriented indexing has been developed ([Fuhr & Buckley 91]). The major concept of this approach is abstraction. For the routing queries, we abstract from specific documents by regarding the presence or absence of terms. In description-oriented indexing, we have to abstract in addition from specific queries and terms. This can be done by regarding features of these objects instead of the objects itself. Similar to pattern recognition methods, documents, queries and terms are described by sets of features here. This way, new documents, queries and terms can be mapped onto sets of features which we have already seen in the learning sample.
In principle, description-oriented learning could be combined with most IR models. Only the probabilistic model, however, relates directly to retrieval quality. The probability ranking principle described in [Robertson 77] states that optimum retrieval performance is achieved when documents are ranked according to descending values of their probability of relevance. So our probabilistic approach uses the learning data in order to optimize retrieval quality for the test sample. This statement also holds for our work with the routing queries, where we apply the retrieval-with-probabilistic-indexing (RPI) model for estimating the query term weights.
In the following section, we describe the description-oriented document indexing method which yields probabilistic weights for terms w.r.t. documents. In order to use these weights in retrieval, two methods are applied here. When no relevance information for the specific query is available, a utility-theoretic retrieval function can be used, where the utility weights for the terms from the query are derived by some heuristics. With relevance feedback data, however, the RPI model can be applied. Experiments with the ad-hoc queries are described in section 3.1, followed by the presentation of our work with the routing queries in section 3.2.

# 2 Probabilistic document indexing

## 2.1 General approach

We first give a brief outline of the description-oriented indexing approach, which is presented in full detail in [Fuhr & Buckley 91]. Based on the binary independence indexing model ([Maron & Kuhns 60], [Fuhr 89]), one can define probabilistic document indexing weights as follows: let $d_m$ denote a document, $t_i$ a term and $R$ the fact that a query-document pair is judged relevant, then $P(R|t_i, d_m)$ denotes the probability that document $d_m$ will be judged relevant w.r.t. an arbitrary query that contains term $t_i$. These weights can hardly be estimated directly, since there will not be enough relevance information available for a specific document. Instead, the description-oriented indexing approach is applied, where the indexing task is divided into two steps, namely a description step and a decision step.
In the description step, term-document pairs $(t_i, d_m)$ are mapped onto so-called *relevance descriptions* $\vec{x}(t_i, d_m)$. The elements $x_i$ of the relevance description contain values of features of $t_i, d_m$ and their relationship, like e.g.

| | |
|---|---|
| $tf$ | within-document frequency (wdf) of $t_i$, |
| *logidf* | = log(inverse document frequency), |
| *lognumterms* | = log(number of different terms in $d_m$), |
| *imaxtf* | = 1/(maximum wdf of a term in $d_m$). |

In the experiments described in the following, we only use these simple parameters. However, it should be emphasized that the concept of relevance description is a means for coping with rather complex forms of document representations, e.g. when advanced natural language processing techniques are applied.

In the decision step, probabilistic indexing weights of the form $P(R|\vec{x}(t_i, d_m))$ are estimated. Instead of the probability $P(R|t_i, d_m)$ which relates to a specific document and a specific term, $P(R|\vec{x}(t_i, d_m))$ is the probability that an arbitrary term-document pair having relevance description $\vec{x}$ will be involved in a relevant query-document relationship. This probability is estimated by a so-called *indexing function* $u(\vec{x})$. Different regression methods or probabilistic classification algorithms can serve as indexing function. Here, we use polynomial regression. For this purpose, a polynomial structure $\vec{v}(\vec{x}) = (v_1, \ldots, v_L)$ has to be defined as

$$\vec{v}(\vec{x}) = (1, x_1, x_2, \ldots, x_N, x_1^2, x_1 x_2, \ldots)^T$$

where $N$ is the number of dimensions of $\vec{x}$. The class of polynomials is given by the components $x_i^l \cdot x_j^m \cdot x_k^n \cdot \ldots \quad (i, j, k, \ldots \in [1, N]; \quad l, m, n, \ldots \geq 0)$ which are to be included in the polynomial. The indexing function now yields $u(\vec{x}) = \vec{a}^T \cdot \vec{v}(\vec{x})$, where $\vec{a} = (a_1, \ldots, a_L)^T$ is the coefficient vector which has to be estimated in a separate training phase preceding the application of the indexing function. So $P(R|\vec{x})$ is approximated by the polynomial

$$u(\vec{x}) = a_1 + a_2 \cdot x_1 + a_3 \cdot x_2 + \ldots + a_{N+1} \cdot x_N + a_{N+2} \cdot x_1^2 + a_{N+3} \cdot x_1 \cdot x_2 + \ldots$$

In the training phase, a learning sample is derived from relevance feedback data in the following way: for every query-document pair $(q_k, d_m)$, a learning sample element $(\vec{x}(t_i, d_m), y)$ is generated for each term $t_i$ common to $q_k$ and $d_m$ with $y = 1$, if $d_m$ is relevant w.r.t. $q_k$, and $y = 0$ otherwise. Given this set of learning sample elements, a linear regression method is applied in order to minimize the squared error $(y - u(\vec{x}))^2$ (see [Fuhr & Buckley 91] for the details). The result of the regression method is the coefficient vector $\vec{a}$ which defines the indexing function.

In the application phase, for each term occurring in a document, a relevance description is constructed first and then the indexing function gives the indexing weight of the term in the document.

## 2.2 Experiments

We developed two types of document indexing, one with single terms only and one with both single words and phrases.

For the single words, we initially used a linear indexing function, which gave us the function:

$$\begin{aligned} u(x(t_i, d_m)) = \; & -0.0019844 & & & + \\ & 0.0000296 & \cdot & \; tf & + \\ & 0.0001079 & \cdot & \; imaxtf & + \\ & 0.0003519 & \cdot & \; logidf & + \\ & 0.0003068 & \cdot & \; lognumterms. & \end{aligned}$$

Retrieval experiments with this type of document indexing, however, revealed that the few extremly long documents (Federal Register documents with up to 2.6 MB) were always retrieved in the top

ranks. This effect was due to the positive coefficient of *lognumterms*, which gave indexing weights larger than 1 for the terms in these documents, thus yielding high retrieval status value for these documents w.r.t. most queries.

Obviously a linear indexing function is not well suited for coping with such extreme distributions of the elements of the relevance description. Besides the effect on the few very long documents, other documents are also affected by this distribution, for the following reason: Many terms in the long documents will be assigned indexing weights larger than 1. For the computation of the squared error that is to be minimized, the quadratic difference $(y - u(\vec{x}))^2$ between the indexing weight $u(\vec{x})$ and the actual relevance decision $y$ (0 or 1) in the learning sample is considered. Even if $y = 1$, there is still the difference $(u(\vec{x}) - 1)^2$, for $u(\vec{x}) > 1$. In order to reduce this error and thus the overall error, a larger error for other indexing weights is taken into account. There are three possible strategies for coping with this problem:

- Some experiments performed with other material have shown that overall indexing quality can be improved by excluding outliers from the learning sample ([Pfeifer 91]). As outliers, not only those pairs with weights lying on the "correct side" of the interval $[0, 1]$ (i.e. either $y = 0$ and $u(\vec{x}) < 0$ or $y = 1$ and $u(\vec{x}) > 1$) should be regarded. Also the exclusion of those pairs with weights lying on the "wrong side" (i.e. either $y = 0$ and $u(\vec{x}) > 1$ or $y = 1$ and $u(\vec{x}) < 0$) yields better results.
- Using logistic functions instead of linear functions overcomes the problem of indexing weights outside the interval $[0, 1]$. However, we have some experimental evidence ([Pfeifer 90]) that even in this case the removal of outliers from the learning sample (where the outliers are defined w.r.t. a linear indexing function) improves the indexing quality.
- For the experiments described here, we switched from linear to polynomial functions, which also gave fairly good results.

The function finally used for indexing with single terms only is

$$
\begin{aligned}
u(\vec{x}) = \quad & 0.00042293 & & & & & & & & & + \\
& 0.00150083 & \cdot & \mathit{tf} & & & \cdot & \mathit{logidf} & \cdot & \mathit{imaxtf} & + \\
& -0.00150665 & \cdot & \mathit{tf} & & & \cdot & \mathit{imaxtf} & & & + \\
& 0.00010465 & \cdot & \mathit{logidf} & & & & & & & + \\
& -0.00122627 & \cdot & \mathit{lognumterms} & \cdot & \mathit{imaxtf}. & & & & &
\end{aligned}
$$

For indexing with phrases, we first had to derive the set of phrases to be considered. We took all adjacent non-stopwords that occurred at least 25 times in the $D_1$ (training) document set. Then an indexing function common for single words and phrases was developed by introducing additional binary factors *is_single* (*is_phrase*) having the value 1 if the term is a single word (a phrase) and 0 otherwise. The parameters $tf$ and *logidf* were defined for phrases in the same way as for single words, and in the computation of *imaxtf* and *lognumterms* for a document both phrases and single words were considered. This procedure produced the indexing function

$$
\begin{aligned}
u(\vec{x}) = \quad & 0.00034104 & & & & & & & & & + \\
& 0.00141097 & \cdot & \mathit{is\_single} & \cdot & \mathit{tf} & \cdot & \mathit{logidf} & \cdot & \mathit{imaxtf} & + \\
& -0.00119826 & \cdot & \mathit{is\_single} & \cdot & \mathit{tf} & \cdot & \mathit{imaxtf} & & & + \\
& 0.00014122 & \cdot & \mathit{is\_single} & \cdot & \mathit{logidf} & & & & & + \\
& -0.00120413 & \cdot & \mathit{lognumterms} & \cdot & \mathit{imaxtf} & & & & & + \\
& 0.00820515 & \cdot & \mathit{is\_phrase} & \cdot & \mathit{tf} & \cdot & \mathit{logidf} & \cdot & \mathit{imaxtf} & + \\
& -0.04188466 & \cdot & \mathit{is\_phrase} & \cdot & \mathit{tf} & \cdot & \mathit{imaxtf} & & & + \\
& 0.00114585 & \cdot & \mathit{is\_phrase} & \cdot & \mathit{logidf}. & & & & &
\end{aligned}
$$

For each phrase occurring in a document, indexing weights for the phrase as well as for its two components (as single words) were computed.
In the following, we will refer to the indexing with single words only as "word indexing" and to the indexing using both single words and phrases as "phrase indexing".

# 3 Retrieval

For the probabilistic document indexing weights as described above, there are specific retrieval functions which yield a probabilistic or utility-theoretic ranking of documents w.r.t. a query. These functions differ in the aspect whether or not they consider relevance feedback information for the specific query. For this reason different retrieval functions were applied for the ad-hoc queries and the routing queries.

## 3.1 Ad-hoc queries

For the ad-hoc queries, we used a linar utility-theoretic retrieval function described in [Wong & Yao 89]. Let $q_k^T$ denote the set of terms occurring in the query, $d_m^T$ the set of document terms and $u_{im}$ the indexing weight $u(\vec{x}(t_i, d_m))$. If $c_{ik}$ gives the utility of term $t_i$ for the actual query $q_k$, then the utility of document $d_m$ w.r.t. query $q_k$ can be computed by the retrieval function

$$\varrho(q_k, d_m) = \sum_{t_i \in q_k^T \cap d_m^T} c_{ik} \cdot u_{im}. \tag{1}$$

The only problem here is the estimation of the utility weights $c_{ik}$, for which we do not have a theoretically justified method. As a heuristic approach, we used the SMART $tf \cdot idf$ weights , where $tf$ denotes the number of occurrences of the term $t_i$ in the query $q_k$ here [Salton & Buckley 88]. We assume that there are other choices for this parameter which could significantly improve retrieval quality.

| run | $tf \cdot idf$ | `fuhra1` (words) | `fuhrp1` (phrases) |
|---|---|---|---|
| average precision for recall points: | | | |
| 11-pt Avg. | 0.1750 | 0.1943 | 0.2054 |
| 3-pt Avg. | 0.1159 | 0.1399 | 0.1664 |
| query-wise comparison with median: | | | |
| 11-pt Avg: | | 32:14 | 32:17 |
| Prec. @ 100 docs | | 25:21 | 35:13 |
| Best/worst results: | | | |
| 11-pt Avg: | | 6(2)/1(1) | 4(1)/2 |
| Prec. @ 100 docs | | 2(2)/2 | 8(3)/(2) |
| single words vs. phrases: | | | |
| 11-pt Avg: | | 25:25 | |
| Prec. @ 100 docs | | 21:28 | |

Tabelle 1: Results for adhoc queries

The retrieval function (1) was applied for both kinds of document indexing, where words only were considered for run `fuhra1` and both words and phases in the run `fuhrp1`. Figure 2 shows the recall-precision curves for both runs in comparison to a standard SMART $tf \cdot idf$ run. It can be seen that

both probabilistic indexing methods clearly outperform the $tf \cdot idf$ run, with the exception of the low-recall end for the phrase run (see below). Comparing word indexing with phrase indexing shows mixed results (see also table 1): For the recall-precision averages, phrases perform better again with the exception of the low recall end. More information is given by the recall and precision values at different numbers of documents retrieved (see appendix of this volume). Precision for phrases is worse or about equal to that of words, but recall is always better. This means that phrases perform better for narrow queries (with a small number of relevant documents).

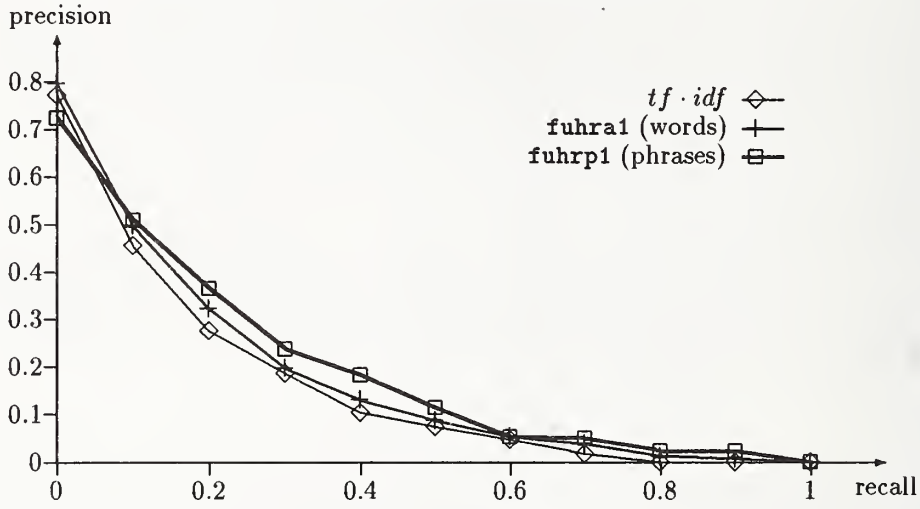



Abbildung 2: Recall-precision curves for probabilistic indexing in comparison to $tf \cdot idf$ run

In general, one would expect that using phrases in addition to single words would never decrease precision. We blame the current definition of the retrieval function (1) for the observed behaviour. This function assumes that all the terms considered are independent, which is obviously not true when we consider a phrase in addition to its two components as terms. Since the size of the error made here depends on the indexing weights of the components, this fact may explain the precision decrease for the highest ranked documents (i.e. at low recall levels). A better strategy would be to ignore the components in case the phrase occurs in the document.

The query-wise comparison with the median (see table 1) shows a large scattering of the results for the single queries. A preliminary analysis has indicated that the relative performance for a specific query depends significantly on the fact whether or not the query statement contains negated terms. For example, in topic 86 we have ... *it is a bank, as opposed to another financial institution such as a savings and loan or credit union* ... and topic 87 reads ... *Civil actions, such as creditor claims filed for recovery of losses, are not relevant* ... . In both cases, our system yields one of the worst results (for word indexing). Since our query indexing procedure extracts only the terms from the query and is not able to recognize negation, the system explicitly searches for documents containing these negated terms. Theoretically, it is obvious that negated terms should be given a negative utility weight. However, the examples cited here show that it is a difficult task to recognize negation automatically.

## 3.2 Routing queries

For the routing queries, the retrieval-with-probabilistic-indexing (RPI) model described in [Fuhr 89] and [Fuhr 92] was applied. This model combines query-specific relevance feedback information with

probabilistic document indexing in order to estimate query term weights. Let $p_{ik}$ denote the average indexing weight of term $t_i$ in the documents judged relevant w.r.t. query $q_k$ and $r_{ik}$ the average indexing weight of $t_i$ in the nonrelevant documents. Now the query term weight is computed by the formula

$$c_{ik} = \frac{p_{ik}(1 - r_{ik})}{r_{ik}(1 - p_{ik})} - 1$$

and the RPI retrieval function yields

$$\varrho(q_k, d_m) = \sum_{t_i \in q_k^T \cap d_m^T} \log(c_{ik} u_{im} + 1). \tag{2}$$

In our experiments, due to the lack of time, we used the standard SMART $tf \cdot idf$ document indexing here [Salton & Buckley 88] (single words only) instead of the probabilistic indexing described above. After an initial retrieval run with both $tf \cdot idf$ weights for queries and documents, relevance feedback information was used for computing the feedback query term weights $c_{ik}$. Only the terms occurring in the query were considered here, so no query expansion took place. Theoretically, the RPI formula can also be applied in case of query expansion. However, the additional terms should be treated differently when estimating their query term weights. This problem has not been investigated yet for the RPI model.

| # queries: 49 | |
|---|---|
| query-wise comparison with median: | |
| 11-pt Avg: | 43:5 |
| Prec. @ 100 docs | 43:5 |
| Best/worst results: | |
| 11-pt Avg: | 12/1 |
| Prec. @ 100 docs | 16/1 |

Tabelle 2: Results for routing queries

Table 2 shows the results for the run `fuhra2` with routing queries. It can be seen that this approach works very well for almost every query. The single worst result is for topic # 50, where we did not retrieve any relevant document; this outcome is due to the fact that there was only one relevant document in the training sample, which is obviously not sufficient for probabilistic parameter estimation.

# A Operational details of runs

## A.1 Overall

All runs were done completely automatically, treating the text portions of both documents and queries as flat text without structure. This made everything much simpler, but was not ideal given the complexity in both form and meaning of the queries. Recall-precision definitely suffered.
All actual indexing (as opposed to weighting) and retrieval was done with the Cornell SMART Version 11.0 system, using the standard SMART procedures (eg, stopwords, stemming, inverted file retrieval).
All runs were made on a Sun Sparc 2 with 64 Mbytes of memory. All times reported are CPU time.

## A.2 Ad-hoc runs

Two automatic ad-hoc runs were done; one in which the documents and queries were indexed with single terms only, and one in which they were indexed with both single terms and adjacency phrases. The overall procedure for both runs was:

1. Index $D_1$ (the learning document set) and $Q_1$ (the learning query set).
2. For each document $d \in D_1$
   2.1 For each $q \in Q_1$
      2.1.1 Determine the relevance value $r$ of $d$ to $q$
      2.1.2 For each term $t$ in common between $q^T$ (set of query terms) and $d^T$ (set of document terms)
         2.1.2.1 Find values of the elements of the relevance description involved in this run and add values plus relevance information to the least squares matrix being constructed
3. Solve the least squares matrix to find the coefficient vector $\vec{a}$
4. Index $D_1 \cup D_2$ (both sets of documents together) with term-freq weights.
5. For each document $d \in D_1 \cup D_2$ (both sets of documents together)
   5.1 For each term $t \in d^T$
      5.1.1 Find values of the relevance description $\vec{x}(t,d)$ involved in run.
      5.1.2 Give $t$ the weight $\vec{a} \cdot \vec{v}(\vec{x}(t,d))$
         where $\vec{a}$ is the value determined in step 3.
   5.2 Add $d$ to the inverted file.
6. Weight $Q_2$ (test query set) with $tf \cdot idf$ weights (ntc variant).
7. Run an inner product inverted file similarity match of $Q_2$ against the inverted file formed in step 5, retrieving the top 200 documents.

In an operational environment, the learning document set and test document set would be the same, so step 1 (or step 4) would be omitted. Once coefficients have been found, they should remain valid unless the character of the collection changes. So new documents can be added to a dynamic collection by just going through step 5 for each new document.
The algorithm above is different from that implemented in the past for this learning approach. Earlier versions iterated over queries (instead of documents) in step 2 and used inverted files for speed (with the document vectors still being needed). However, the size of TREC required a reformulation since document vectors and inverted files could not be kept in memory.
The coefficient determination is valid only if the relevant judgements used are representative of the entire set of reolevance judgements. In this first TREC, the initial judgements are fragmentary and not very representative; it's impossible to tell how much this affects things.

## A.3 Single term automatic ad-hoc run

The single term ad-hoc run used 5 factors (described in 2.1):

- *constant*
- $tf \cdot logidf \cdot imaxtf$
- $tf \cdot imaxtf$
- *logidf*
- $lognumterms \cdot imaxtf$

Determining the coefficients for the factors in the single term run took about 1.7 hours (steps 2 + 3 above). Construction of the inverted file (steps 4+5) containing the factor weighted terms took about 6.4 hours from scratch. Note that this is only about 1.0 hours longer than construction of a normal (for SMART) $tf \cdot idf$ weighted inverted file.
Query indexing and weighting used the normal SMART procedures and took about 1.5 seconds. The fields Topic, Nationality, Narrative, Concepts, Factors, Description were used to index the query, with no distinction made between fields.
Retrieval plus ranking took 383 seconds.

## A.4 Phrase automatic ad-hoc run

The phrases being used were two-term SMART adjacency phrases. Phrases were adjacent non-stopwords, term components stemmed, that occurred at least 25 times in the $D_1$ document set. The term components were put into alphabetical order, thus the text phrases "information retrieval" and "retrieving information" both mapped to the same phrase concept. The phrases were treated as a separate ctype within an indexed vector, and had their own dictionary and inverted file separate from those of the single terms.
Determination of phrases took 5.8 hours, finding 4,700,000 phrases occurring in $D_1$ at least once. Of those phrases 158,000 occurred at least 25 times. These phrases were then put into a dictionary and used as controlled vocabulary for phrases when doing the indexing of $D_1$ (step 1) and $D_1 \cup D_2$ (step 4). The single term indexing remained exactly as it was in the single term run (terms occurring in phrases were not removed from the vector).
The phrase term ad-hoc run used 8 factors (described in 2.2):

- $constant$
- $is\_single \cdot tf \cdot logidf \cdot imaxtf$
- $is\_single \cdot tf \cdot imaxtf$
- $is\_single \cdot logidf$
- $lognumterms \cdot imaxtf$
- $is\_phrase \cdot tf \cdot logidf \cdot imaxtf$
- $is\_phrase \cdot tf \cdot imaxtf$
- $is\_phrase \cdot logidf$

Determining the coefficients for the factors in the phrase run took about 2.4 hours (steps 2 + 3 above). Construction of the inverted file (steps 4+5) containing the factor weighted terms took about 12.6 hours from scratch. Note that this is only about 1.7 hours longer than construction of a normal (for SMART) $tf \cdot idf$ weighted inverted file with phrases.
Query indexing and weighting used the normal SMART procedures and took about 2.7 seconds. The fields Topic, Nationality, Narrative, Concepts, Factors and Description were used to index the query, with no distinction made between fields.
Retrieval plus ranking took 374 CPU seconds. (This was less than the single term run, but for no apparent reason. Perhaps the machine was less loaded.)

## A.5 Automatic routing run

There was one automatic routing run done. It was totally unconnected to the factor weighting approach described above. Basically, it was very easy to implement and run so we decided we might as well submit it. The actual weighting function had to be programmed, but even so less than 3 person-days in total was spent on routing.

The routing experiment format was treated almost exactly as a normal relevance feedback experimental run. The overall procedure was:

1. Index query set $Q_1$ and document set $D_1$ with $tf \cdot idf$ weights
2. For each query $q \in Q_1$
   2.1 For each term $t \in q^T$ (set of query terms)
      2.1.1 Reweight term $t$ using the RPI relevance weighting formula and the (fragmentary) relevance information supplied.
3. Index document set $D_2$ with $tf \cdot idf$ weights. Note that the collection frequency information used in the *idf* weight was derived from occurrences in $D_1$ only (in actual routing the collection frequencies within $D_2$ would not be known)
4. Run the reweighted queries of $Q_1$ (step 2) against the inverted file (step 3), returning the top 200 documents for each query.

This approach differs from true routing in that

A. All documents of $D_2$ were indexed at once instead of individually indexing them and comparing at each query sequentially. Thus the indexing/retrieval times obtained for the above algorithm are pretty meaningless.

B. True routing is really a binary decision most often implemented as a similarity threshold, and retrieving the top 200 documents (in ranked order) wouldn't normally be done. However, this difference from true routing is required for evaluation purposes, and was thus required for TREC.

Note that the approach was completely automatic with the query and documents treated as flat text (no structure). Differing from the ad-hoc runs above, the queries were indexed including the words from all topic sections.

It's unknown what effect the fragmentary relevance information had on the query reformulation. The strength of the effect will depend on whether the top similarity documents according to the $tf \cdot idf$ weights used had been judged and included in the fragmentary judgements.

Step 1 took 3.0 hours, step 2 about 1304 seconds, step 3 about 1.9 hours, and step 4 about 312 seconds.

## References


**Fuhr, N.; Buckley, C.** (1991). A Probabilistic Learning Approach for Document Indexing. *ACM Transactions on Information Systems 9(3)*, pages 223–248.

**Fuhr, N.** (1989). Models for Retrieval with Probabilistic Indexing. *Information Processing and Management 25(1)*, pages 55–72.

**Fuhr, N.** (1992). Integration of Probabilistic Fact and Text Retrieval. In: Belkin, N.; Ingwersen, P.; Pejtersen, M. (eds.): *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 211–222. ACM, New York.

**Maron, M.; Kuhns, J.** (1960). On Relevance, Probabilistic Indexing, and Information Retrieval. *Journal of the ACM 7*, pages 216–244.

**Pfeifer, U.** (1990). *Development of Log-Linear and Linear-Iterative Indexing Functions (in German).* Diploma thesis, TH Darmstadt, FB Informatik, Datenverwaltungssysteme II.

**Pfeifer, U.** (1991). Entwicklung linear iterativer und logistischer Indexierungsfunktionen. In: Fuhr, N. (ed.): *Information Retrieval*, pages 23–37. Springer, Berlin et al.

**Robertson, S.** (1977). The Probability Ranking Principle in IR. *Journal of Documentation 33*, pages 294–304.

**Salton, G.; Buckley, C.** (1988). Term Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management 24(5)*, pages 513–523.

**Wong, S.; Yao, Y.** (1989). A Probability Distribution Model for Information Retrieval. *Information Processing and Management 25(1)*, pages 39–53.

# The University of Massachusetts TIPSTER Project

W. Bruce Croft
Computer Science Department
University of Massachusetts
Amherst, MA. 01003

The TIPSTER project in the Information Retrieval Laboratory of the Computer Science Department, University of Massachusetts, Amherst (which includes MCC and David Lewis of the University of Chicago as subcontractors), is focusing on the following goals:

- Improving the effectiveness of information retrieval techniques for large, full-text databases,
- Improving the effectiveness of routing techniques appropriate for long-term information needs, and
- Demonstrating the effectiveness of these retrieval and routing techniques for Japanese full-text databases.

Our general approach to achieving these goals has been to use improved representations of text and information needs in the framework of a new model of retrieval. Retrieval (and routing) is viewed as a probabilistic inference process which "compares" text representations based on different forms of linguistic and statistical evidence to representations of information needs based on similar evidence from natural language queries and user interaction. New techniques for learning (relevance feedback) and extracting term relationships from text are also being studied. The details and evaluation (with smaller test databases) of the new model, known as the inference net model, can be found in other papers [3, 2, 4].

Some of the specific research issues we are addressing are morphological analysis in English and Japanese, word sense disambiguation in English, the use of phrases and other syntactic structure in English and Japanese, the use of special purpose recognizers in representing documents and queries, analyzing natural language queries to build structured representations of information needs, learning techniques appropriate for routing and structured queries, and probability estimation techniques for indexing.

Comparing the TIPSTER experiments to previous IR experiments done using the standard test collections (e.g. CACM, CISI, NPL, etc.), there are a number of interesting differences:

- The size of the corpus is much larger than previous collections, both in terms of the number of documents and the amount of text. This presents a challenge to the robustness and efficiency of experimental information retrieval systems. Experiments with indexing, for example, can take days instead of minutes.
- The documents in TIPSTER are nearly all full text, rather than abstracts.

- The documents in TIPSTER are heterogeneous in terms of both subject and length. They come the general area of science, technology and economics, but the sources are the Wall Street Journal, Associated Press newswire, Ziff magazines in the high technology area, Department of Energy abstracts, and the Federal Register.
- The queries (known as "topics" in TIPSTER) are longer and have more structure than those found in other test collections.
- The queries have specific and strict criteria specified for documents to be relevant. These criteria (specified in the "narrative" part of the topic) will reduce inconsistency between relevance judges, but are sometimes difficult to handle in the context of an information retrieval system.
- The routing experiments are unlike any carried out before.
- The retrieval and routing experiments with Japanese are also unique.

The first TIPSTER evaluation was limited by a number of factors, the primary one being the lack of relevance judgements for the initial query set. This made it difficult to carry out experiments to select techniques appropriate for large, full-text databases. The results from this evaluation should, therefore, be regarded as preliminary, and indeed raise more questions than they answer.

In the retrieval experiment, 50 new "topics" were used to search the "old" database, which consisted of approximately 1 GByte of text. One of the major subjects of the evaluation was to try different forms of queries produced by processing the topics. Our basic approach to topic processing is to parse them, selecting parts to be indexed, recognizing phrases and "factors" such as locations, dates, companies, etc. Some factors, such as "developing country", which have been specifically identified as important in the topic, will be expanded using a synonym operator. Weights reflecting relative importance are attached to the concepts (words and phrases). Phrase-based concepts are represented by operators defined in the inference net language. These operators use proximity of the words making up the phrase as the major form of evidence for the presence of the concept [1]. The result of topic processing is an inference net representing the information need.

In addition to the automatic query processing, some query versions were generated by simulating simple user interaction with the results of the topic processing. The modifications to the automatically processed topics were limited to changing the weight of concepts, deleting concepts considered unimportant, and adding structure (such as specifying synonymous concepts). The most significant change in the last category was the introduction of "unordered window" operators to simulate paragraph-level retrieval. The equivalent in terms of a user user interface would be to ask users to group concepts that should occur together.

The results of the first evaluation are described here in terms of the average precision in the top 5, 30 and 200 documents in the ranking produced by the inference net retrieval engine (INQUERY). This evaluation method was chosen because only the top 200 documents for each query were judged for relevance. The results were as follows:

| Query Type | Average Precision (50 topics) | | |
|---|---|---|---|
| | 5 docs | 30 docs | 200 docs |
| T+D+C+F+phrase | .64 | .52 | .35 |
| T+D+C+F | .62 (-3.1%) | .52 (0%) | .35 (0%) |
| 1+N | .60 (-6.7%) | .50 (-3.8%) | .34 (-2.8%) |
| T+C+phrase | .66 (+3.1%) | .53 (+1.9%) | .36 (+2.8%) |
| 1+man | .65 (+1.6%) | .56 (+7.7%) | .36 (+2.8%) |
| 1+man+para | .72 (+12.5%) | .61 (+17.3%) | .39 (+10.3%) |

Table 1: TIPSTER Retrieval Results: Query types refer to topic fields used. T is topic, D is description, C is concepts, F is factors, N is narrative, phrase means phrase constructs used, 1 refers to the baseline (the first line), man means manual modification, para means paragraph retrieval.

These results support two main conclusions: the first being that the effectiveness of the retrieval techniques is surprisingly good considering the difficulty of the queries; the second is that paragraph-level retrieval as simulated by manual creation of "unordered window" queries significantly improves effectiveness. Much of the short-term development of the inference net retrieval system will concentrate on techniques to accomplish paragraph-level retrieval automatically. The major question raised by the results concerns the effectiveness of phrases. In previous experiments with medium-sized full-text collections, phrase-based retrieval led to significant effectiveness improvements. This is not evident in the results shown here. A possible explanation for this is the size of the TIPSTER topics, where queries may have more than 50 terms, but it should also be remembered that these results are very preliminary.

The routing experiments used 20 "old" topics to search the "new" database (approximately 1 GByte of text from the same sources as the "old" database, with the exception of DOE abstracts). Since the aim of these experiments was to study techniques for representing and using long-term information needs, we assumed that users would be more involved in query formulation and thus the baseline used was the "1+man+para" queries. The other query types in this experiment used variations of relevance feedback to modify the baseline queries. These modifications consist of adding concepts to the query and reweighting the query concepts based on their frequency of occurrence in the identified relevant documents. For this experiment, we had a small number of relevance judgements based on documents retrieved by another system. The techniques used to select concepts to add to the query were based on local and global application of the EMIM measure of association [5] The number of terms added to a query was limited to 5.

The results show that, once again, the effectiveness levels are quite good (note the 50% precision value at the 200 document level). The relevance feedback techniques were not effective, except at the high precision end of retrieval. The features selected were, on inspection, reasonable, but they do not appear to be the features required by the narrative in order to make a document relevant. No definite conclusions can be made about the

| Query Type | Average Precision (20 topics) | | |
|---|---|---|---|
| | 5 docs | 30 docs | 200 docs |
| man | .66 | .65 | .50 |
| man+weights | .68 (+3.0%) | .63 (-3.1%) | .48 (-4.0%) |
| man+EMIM+weights | .71 (+7.6%) | .61 (-6.2%) | .49(-2.0%) |
| Man+LEMIM+weights | .68 (+3.0%) | .64 (-1.5%) | .50 (0%) |

Table 2: TIPSTER Routing Results: weights are based on frequency in relevant documents, EMIM is a global selection measure, LEMIM is a local (window-based) selection measure.

feedback techniques until experiments with larger sets of relevance judgements are carried out.

The third set of results are related to the retrieval of Japanese text. The goal of these experiments was to compare different approaches to morphological analysis or word segmentation. Japanese text is made up of characters from a number of alphabets (Kanji, Katakana, Hiragana, and English). There are, however, no word separators and therefore a major part of indexing is deciding what to index. We tested two alternatives:

1. An efficient, relatively crude technique where individual Kanji (Chinese) characters and strings of Katakana characters are indexed.

2. A more sophisticated dictionary and grammar-based segmentation algorithm developed at Kyoto University (JUMAN).

There is a significant difference in the indexing times required by these techniques. With a database of 1,100 documents from a Japanese newspaper, the character-based indexing took 4 minutes while the word-based (JUMAN) indexing took 31 minutes. The relative effectiveness of the two text representations was then tested using the average precision in the top 10 documents for 30 queries. The queries were either treated as strings of characters, or were automatically structured using the JUMAN segmenter. In the character-based approach, words found in the query were expressed using the phrase operator to combine Kanji and Katakana characters. The results show that the retrieval performance using Japanese seems to be comparable to similar experiments with English databases, and the relatively simple character-based indexing technique is surprisingly effective compared to more sophisticated word-based techniques. The latter result is interesting, but the experiment must be repeated when the larger TIPSTER Japanese database and query set becomes available.

We are currently carrying out a range of more detailed experiments using the relevance judgements that are now available. The results from these experiments will allow us to tune the techniques being used and to make more definite conclusions about their relative effectiveness. In addition, we will continue to incorporate new approaches into the retrieval and routing software for the upcoming evaluations.

| Query Type | Average Precision in Top Ten (30 queries) | |
|---|---|---|
| | Character-Based Indexing | Word-Based Indexing |
| Characters | .61 | - |
| Words using phrase operator | .63 (+3.3%) | - |
| Words | - | .65 (+6.6%) |

Table 3: TIPSTER Japanese Retrieval Results

## References


[1] W.B. Croft, H. Turtle, D. Lewis, "The Use of Phrases and Structured Queries in Information Retrieval", Proceedings of SIGIR 91, 32-45, (1991).

[2] W.B. Croft and H. Turtle, "Text Retrieval and Inference", in *Text-Based Intelligent Systems*, Paul Jacobs (ed.), Lawrence Erlbaum, New Jersey, 127-156, (1992).

[3] H.R. Turtle and W.B. Croft, "Evaluation of an Inference Network-Based Retrieval Model", *ACM Transactions on Information Systems*, 9(3), 187-222, (1991).

[4] H. Turtle and W.B. Croft, "A Comparison of Retrieval Models", *Computer Journal*, 35(3), 279-290, (1992).

[5] C. J. Van Rijsbergen, *Information Retrieval.* Butterworths, (1979).

# HNC's *MatchPlus* System

Stephen I. Gallant* William R. Caid† Joel Carleton†
Robert Hecht-Nielsen† Kent Pu Qing† David Sudbeck†

HNC Inc

## 1 Introduction

HNC is developing a neural network related approach to document retrieval called *MatchPlus*[1]. Goals of this approach include high precision/recall performance, ease of use, incorporation of machine learning algorithms, and *sensitivity to similarity of use.*

To understand our notion of sensitivity to similarity of use, consider the four words: 'car', 'automobile', 'driving', and 'hippopotamus'. 'Car' and 'automobile' are synonyms and they very often occur together in documents; 'car' and 'driving' are related words (but not synonyms) that sometimes occur together in documents; and 'car' and 'hippopotamus' are essentially unrelated words that seldom occur within the same document. We want the system to be sensitive to such similarity of use, much like a built-in thesaurus, yet without the drawbacks of a thesaurus, such as domain dependence or the need for hand-entry of synonyms. In particular we want a query on 'car' to prefer a document containing 'drive' to one containing 'hippopotamus', and we want the *system itself* to be able to figure this out from the corpus.

The implementation of *MatchPlus* is motivated by neural networks, and designed to interface with neural network learning algorithms. High-dimensional ($\approx$ 300) vectors, called context vectors, represent word stems, documents, and queries in the same vector space. This representation permits one type of neural network learning algorithm to generate stem context vectors that are sensitive to similarity of use, and a more standard neural network algorithm to perform routing and automatic query modification based upon user feedback.

Queries can take the form of terms, full documents, parts of documents, and/or conventional Boolean expressions. Optional weights may also be included.

*124 Mt Auburn St, Suite 200, Cambridge, MA 02138
†5501 Oberlin Drive, San Diego, CA 92121.
[1]Patents pending.

The following sections give a brief overview of our implementation, and look at some preliminary test results. For a previous description of the approach and comments on complexity considerations see [1]; a longer journal article is in preparation.

## 2 The Context Vector Approach

One of the most important aspects of *MatchPlus* is its representation of words (stems), documents, and queries by high ($\approx$ 300) dimensional vectors called *context vectors.* By representing all objects in the *same* high dimensional space we can easily:

1. Form a document context vector as the (weighted) vector sum of the context vectors for those words (stems) contained in the document.
2. Form a query context vector as the (weighted) vector sum of the context vectors for those words (stems) contained in the query.
3. Compute the distance of a query $Q$ to any document. Moreover if document context vectors are normalized, the closest document $d$ (in Euclidean distance) has the context vector $\mathbf{V}^d$ that gives highest dot product with the query context vector $\mathbf{V}^q$:

   $$<\text{closest } d> = \{d | \mathbf{V}^d \cdot \mathbf{V}^Q \text{ is maximized for } d \in D\}$$

   (proof: $\|\mathbf{V}^d - \mathbf{V}^Q\|^2 = \|\mathbf{V}^d\|^2 + \|\mathbf{V}^Q\|^2 - 2(\mathbf{V}^d \cdot \mathbf{V}^Q) = \text{const} - 2(\mathbf{V}^d \cdot \mathbf{V}^Q)$.)
4. Find the closest document to a given document $d$ by treating $\mathbf{V}^d$ as a query vector.
5. Perform relevance feedback. If $d$ is a relevant document for query $Q$, form a new query vector

   $$\widehat{\mathbf{V}^Q} = \mathbf{V}^Q + \alpha \mathbf{V}^d$$

where $\alpha$ is some suitable positive number (eg 3). (See also [6].) Note that search with $\widehat{\mathbf{V}^Q}$ takes the same amount of time as search with $\mathbf{V}^Q$.

## 2.1 Context Vector Representations

Context vector representations (or feature space representations) have a long history in cognitive science. Work by Waltz & Pollack [7] had an especially strong influence on the work reported here. They described a neural network model for word sense disambiguation and developed context vector representations (which they termed micro-feature representations). See Gallant [2] for more background on context vector representations and word sense disambiguation.

We use context vector representations for document retrieval, with most of the representation being learned from an unlabeled corpus. A main constraint for all of this work is to keep computation and storage reasonable, even for very large corpora.

For defining context vectors, we begin by specifying a set of $n$ *features* that are useful for differentiating among terms and contexts. These may be chosen in an ad hoc manner using "common sense", or they may consist of the high frequency terms in a given corpus after removal of *stopwords* (*a, the, and,* $\cdots$). Figure 1 gives some typical examples that might be suitable for general news stories. Experiments suggest that the precise selection of features is not critical to system performance.

| | | | | |
|---|---|---|---|---|
| human | man | woman | machine | politics |
| art | science | play | sex | entertainment |
| walk | lie-down | motion | speak | yell |
| research | fun | sad | exciting | boring |
| friend | family | baby | country | hot |
| cold | hard | soft | sbarp | heavy |
| light | big | small | red | black |
| white | blue | yellow | animal | mammal |
| insect | plant | tree | flower | bush |
| fruit | fragrant | stink | past | present |
| future | high | low | wood | plastic |
| paper | metal | building | house | factory |
| work | early | late | day | night |
| afternoon | morning | sunny | cloudy | rain |
| snow | hot | cold | humid | bright |
| smart | dumb | car | truck | bike |
| write | type | cook | eat | spicy |
| ... | | | | |

Figure 1: Some typical features. For convenience some features that apply to **star** have been moved to the top of the list.

For any word stem $k$ we now define its context vector, $\mathbf{V}^k$, to be an $n$-dimensional vector, and interpret each component of $\mathbf{V}^k$ as follows:

- $\mathbf{V}^k_j \approx$ {strongly} positive if word $k$ is {strongly} associated with feature $j$
- $\mathbf{V}^k_j \approx 0$ if word $k$ is not associated with feature $j$
- $\mathbf{V}^k_j \approx$ {strongly} negative if word $k$ {strongly} contradicts feature $j$.

As an example, $\mathbf{V}^{\text{astronomer}}$ might be

$$\begin{array}{rrrrrl} < +2 & +1 & +1 & -1 & -1 & \\ 0 & +2 & 0 & 0 & 0 & \\ 0 & 0 & +1 & +1 & +1 & \\ +2 & +1 & -1 & +1 & -1 & \\ \cdots & \cdots & \cdots & \cdots & \cdots & > \end{array}$$

using the features in figure 1. Note that the interpretation of components of context vectors is exactly the same as the interpretation of weights in neural networks.

In addition to the "word features" there are other "learned features" that primarily serve to increase the dimensionality of context vectors.

Features are deliberately chosen so that they overlap. This makes context vectors less dependent upon any individual feature. Context vector representations give built-in sensitivity to similarity of meaning between terms. For example it is likely that the context vector for 'car' would be very close to the context vector for 'auto', somewhat close to the context vector for 'driving', and less close to the context vector for 'hippopotamus' for *any "reasonable" set of features* and *for any "reasonable" person entering the context vectors.* For more on this point see the plausibility argument in Waltz & Pollack [7]. Note that overlapping features provide a distributed representation [3], and therefore help insulate against a small number of questionable entries for context vectors.

## 2.2 Bootstrap Learning

Bootstrapping is a machine learning technique that begins with hand-entered context vectors for a small set of *core stems*, and then uses an unlabeled training corpus to create context vectors for all remaining stems.

The basic idea is to define the context vector for a new stem by making it similar to the context vectors of its neighbor stems. Note that bootstrapping takes into account *local word positioning* when assigning the context vector representation for stems. Moreover it is nearly invariant with respect to document divisions within the training corpus. This contrasts with those methods where stem representations are determined solely by those *documents* in which the stem lies.

We have also developed a fully-automated method for bootstrapping that requires no initial hand entry. This capability is very useful for specialized domains such as the tests on traditional IR corpora presented in the next section. We are currently researching

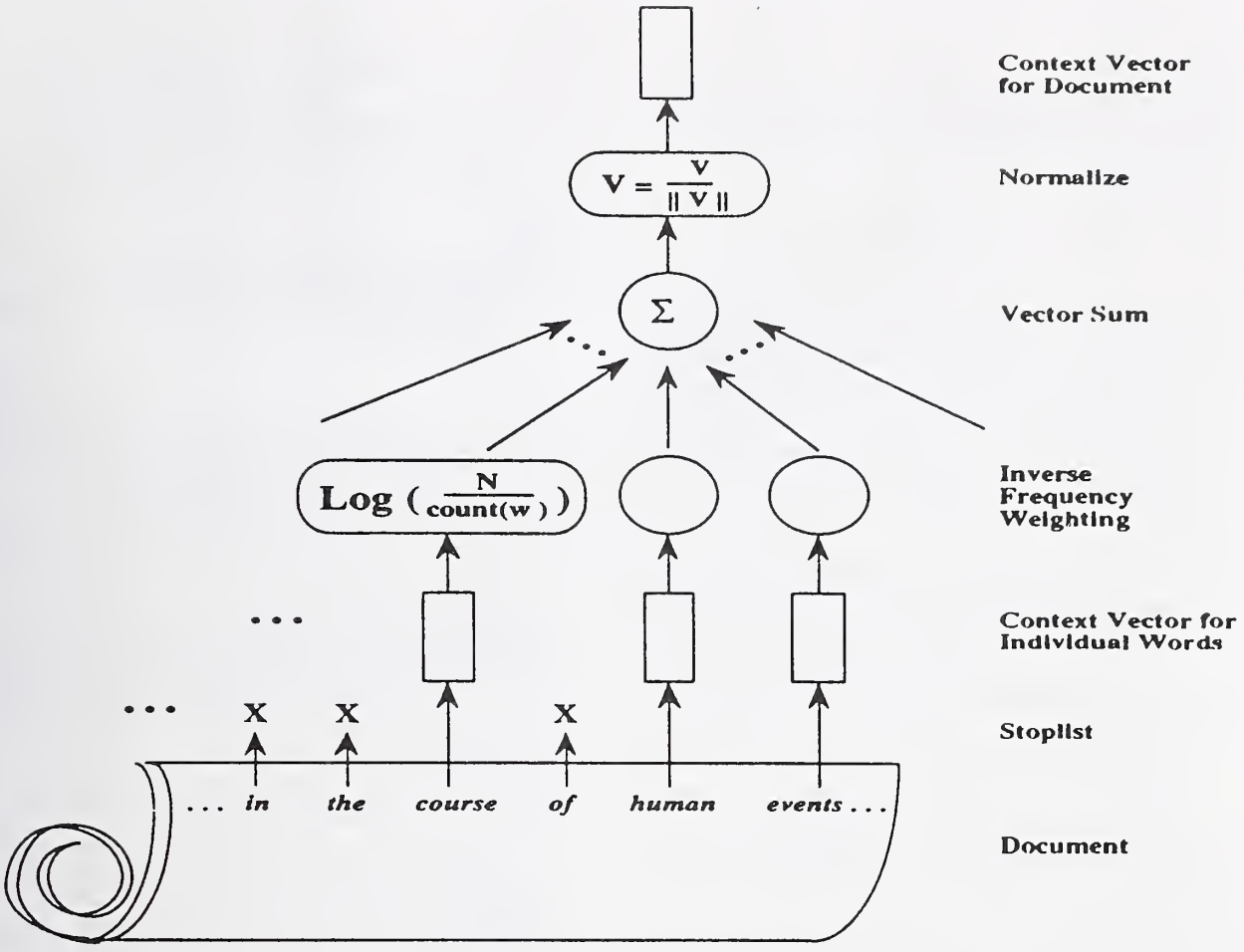



Figure 2: Generating the context vector for a document.

whether the fully-automated method can perform as well as standard bootstrapping.

## 2.3 Context Vectors for Documents

Once we have generated context vectors for stems it is easy to compute the context vector for a document. We simply take a weighted sum of context vectors for all stems appearing in the document[2] and then normalize the sum. See figure 2. This procedure applies to documents in the training corpus as well as to new documents. When adding up stem context vectors, we can use term frequency weights similar to conventional IR systems.

## 2.4 Context Vectors for Queries; Relevance Feedback

Query context vectors are formed similarly to document context vectors. For each stem in the query we can apply a user-specified weight (default 1.0). Then we can sum the corresponding context vectors and normalize the result.

Note that it is easy to implement relevance feedback. The user can specify documents (with weights) and the document context vectors are merely added in with the context vectors from the other terms. We can also find documents close to a given document by using the document context vector as a query context vector.

## 2.5 Retrieval

The basic retrieval operation is simple: we find the document context vector closest to the query context vector and return it. There are several important points to note.

1. As many documents as desired may be retrieved, and the distances from the query context vector give some measure of retrieval quality.

[2]Stopwords are discarded.

2. Because document context vectors are normalized, we may simply find the document $d$ that maximized the dot product with the query context vector, $\mathtt{V}^q$:

$$\max_d \{\mathtt{V}^d \cdot \mathtt{V}^Q\}.$$

3. It is easy to combine keyword match with context vectors. We first use the match as a filter for documents and return documents in order by closeness to the query vectors. If all matching documents have been retrieved, *MatchPlus* can revert to context vectors for finding the closest remaining document.

4. *MatchPlus* requires only about 300 multiplications and additions to search a document. Moreover it is easy to decompose the search for a corpus of documents with either parallel hardware or, less expensively, several networked conventional machines (or chips). Each machine can search a subset of the document context vectors and return the closest distances and document numbers in its subset. The closest from among the distances returned by *all* the processors then determines the documents chosen for retrieval.

   We are also investigating a *cluster tree* pruning procedure that finds nearest neighbor document context vectors without having to compute dot products for all document context vectors. This data organization affects retrieval speed, but does not change the order in which documents are retrieved.

## 3 Preliminary Results

Our system is very 'young'. It has been able to handle large corpora (1,000,000+ documents) only since July 1992. Nevertheless we have some promising results.

In figure 3, we see that *MatchPlus* gives comparable performance to Salton's SMART system [5] on small, traditional IR test corpora when corresponding term weighting schemes are used. Salton reports significantly improved performance (10—50%) with other term weighting methods, and we are in the process of running the corresponding tests with *MatchPlus*.

These experiments used fully automated bootstrapping with no hand entry of context vectors. Experiments on other corpora with a hand-entered set of core stems show 3% to 15% improvement, with larger improvement on smaller corpora.

Bootstrapping for the tests in figure 3 was on the target corpus only, with maximum size being 3200 documents. We have found a significant advantage to bootstrapping with larger corpora, as shown in figure 4. We also plan experiments where bootstrapping begins with stem context vectors generated from a larger corpus.

| | CISI | CACM | MED |
|---|---|---|---|
| MatchPlus | .1914 | .1749 | .5013 |
| SMART | .1410 | .2535 | .5062 |

*Notes:*
- *MatchPlus*: $Match(3) filter
- SMART figures from Salton [5].
- Comparisons use classical *idf* term weighting for both systems.

Figure 3: *MatchPlus* results are comparable to SMART system results on traditional IR corpora using a corresponding term weighting method.

| Bootstrap corpus size (# docs) | 50,000 | 200,000 | 500,000 |
|---|---|---|---|
| Performance | 36.5 | 39.0 | 39.9 |
| Improvement | — | 7% | 9% |

*Notes:*
- Performance was average relevant for 200 retrievals using Tipster corpus.

Figure 4: Improvement with size of bootstrap corpus.

Experiments with the Tipster/TREC corpus are in progress.

## 4 Concluding Comments

The key feature of *MatchPlus* is its uniform representation of all objects by context vectors. This makes possible a large number of interesting experiments for the next year, such as

- use of neural network learning algorithms to perform fast automated query modification based upon user feedback
- word sense disambiguation as described in [2]

- cluster tree speedup for retrieval
- different term weighting methods for document and query context vector creation, as well as for bootstrapping.

Although young, we believe *MatchPlus* has already shown encouraging results, and we look forward to growing it.

## References


[1] Gallant, S. I. Context Vector Representations for Document Retrieval. AAAI-91 Natural Language Text Retrieval Workshop, Anaheim, CA, July 15, 1991.

[2] Gallant, S. I. A Practical Approach for Representing Context And for Performing Word Sense Disambiguation Using Neural Networks. Neural Computation, Vol. 3, No. 3, 1991, 293-309.

[3] Hinton, G. E. Distributed Representations. Technical Report CMU-CS-84-157, Carnegie-Mellon University, Department of Computer Science. Revised version in Rumelhart, D. E. & McClelland, J. L. (Eds.) *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol. 1.* MIT Press, 1986.

[4] Salton, G. *The SMART retrieval system – Experiments in automatic automatic document processing.* Englewood Cliffs, NJ: Prentice-Hall, 1971.

[5] Salton, G. & Buckley, C. Term-Weighting Approaches in Automatic Text Retrieval. Information Processing & Management, Vol. 24, No. 5, 1988, pp. 513-523.

[6] Salton, G. & Buckley, C. Improving Retrieval Performance by Relevance Feedback. Journal of the American Society for Information Science, 41(4):288–297, 1990.

[7] Waltz, D. L. & Pollack, J. B. Massively Parallel Parsing: A Strongly Interactive Model of Natural Language Interpretation. Cognitive Science 9, 51-74 (1985).

# DR-LINK's Linguistic-Conceptual Approach to Document Detection[1]

Elizabeth D. Liddy
Sung H. Myaeng
School of Information Studies
Syracuse University
Syracuse, New York 132100-4100
liddy@mailbox.syr.edu; shmyaeng@mailbox.syr.edu

## 1. Overview

Our approach to the difficult problem of selecting only those documents which satisfy a user's specified information need, is to pay parallel attention to two very important aspects of the task. Firstly, there are many documents which have no likely possibility of being relevant to either a standing query or a query newly put to the system. These documents should be filtered from further consideration at an early stage in the system's processing if the system's later processing is computationally expensive, and if their presence introduces unnecessary ambiguity, while their removal produces more accurate results. This focusing process should continue at subsequent stages using additional linguistic features of the query and documents in order to further refine the flow of documents. Secondly, there is a continuum of levels of linguistic-conceptual processing which can produce enrichments of the original text in order to explicitly represent documents at more conceptual levels for more accurate matching to queries.

Our approach also recognizes that, as reflected by the topic statements, the retrieval task in TREC requires capabilities beyond what has been required in the past for traditional IR systems. Topic statements describe not only 'aboutness' but also more detailed information such as relationships among entities, characteristics of participants in an event, and temporality. We believe that richer representations of documents and topic statements are essential to meet the extended retrieval requirements of such complex information needs and to reduce the ambiguities resulting from keyword-based retrieval. To produce this enriched representation the system uses lexical, syntactic, semantic, and discourse linguistic processing techniques for distilling from documents and topic statements all the rich layers of knowledge incorporated in their deceptively simple textual surface and produces a final document representation which has been snaped by all these levels of linguistic processing.

To achieve the goals stated above, we have developed a system whose architecture is modular in design, with five separate processing modules which continuously refine the flow of documents both in terms of pure numbers and in terms of continual semantic enrichments (see Figure 1). Briefly previewed, the five modules processing is as follows:

The **Subject Field Coder** uses semantic word knowledge to produce a summary-level topical vector representation of a document's contents that is matched to a vector representation of a topic statement in order to select for further processing only those documents which have real potential of being relevant. This subset of documents is then passed to the **Text Structurer**,

---

[1] Ken McVearry, Woojin Paik, Ming Li, Edmund Yu, and Chris Khoo contributed to the design, data analysis and implementation of the system for TREC-1.

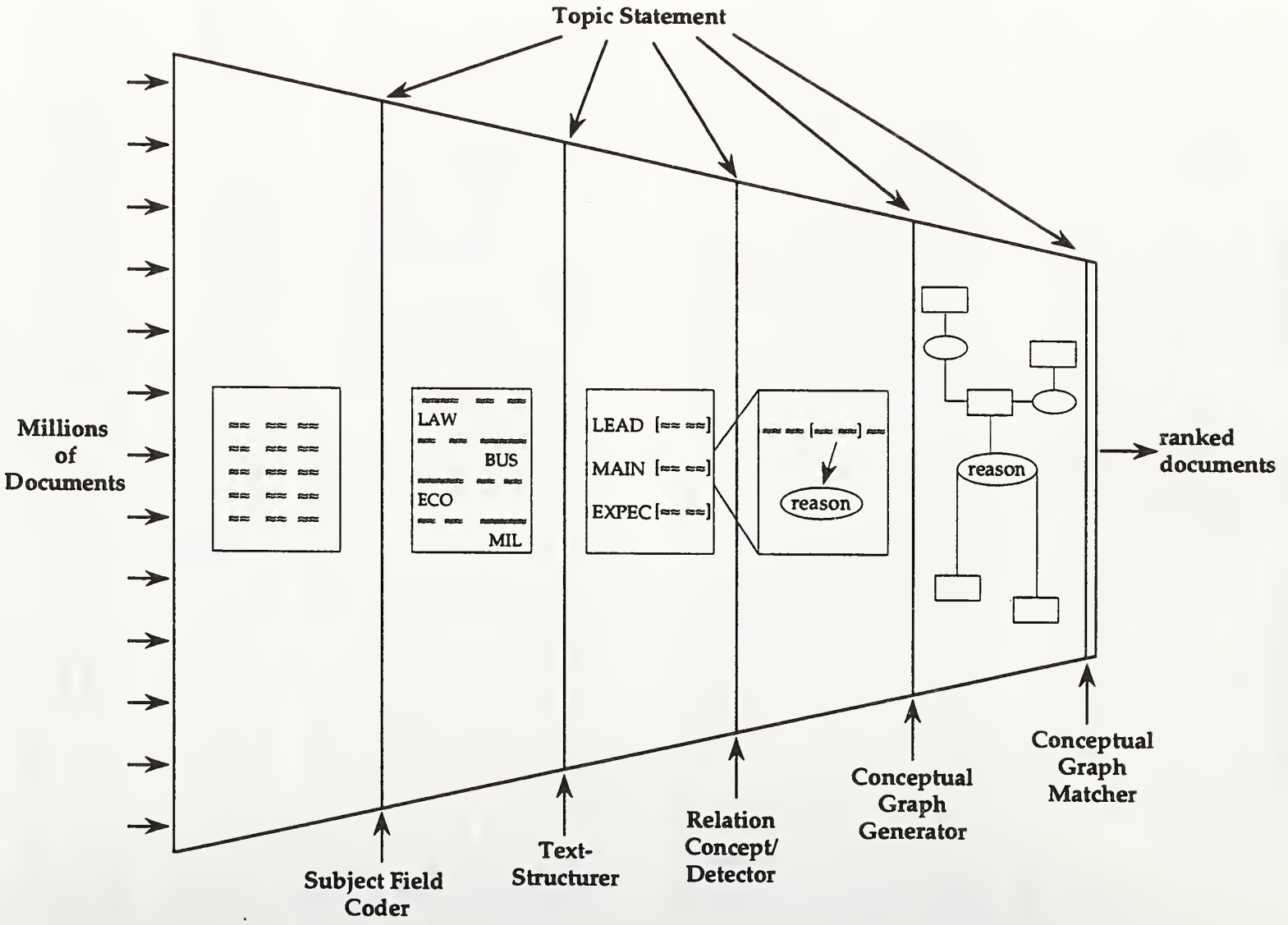

Topic Statement
Millions of Documents
LAW
BUS
ECO
MIL
LEAD [≈≈ ≈≈]
MAIN [≈≈ ≈≈]
EXPEC [≈≈ ≈≈]
reason
reason
ranked documents
Subject Field Coder
Text-Structurer
Relation Concept/ Detector
Conceptual Graph Generator
Conceptual Graph Matcher

Fig. 1: DR-LINK

which sub-divides a text into its discourse-level segments in order to focus later matching to the appropriate discourse component in response to particular types of information need. All of the structured texts, with the appropriate components high-lighted, are passed to the **Relation-Concept Detector**, whose purpose is to raise the level at which we do matching from a key-word or key-phrase level to a more conceptual level by expanding terms in the topic statement to all terms which have been shown to be 'substitutable' for them, and then by extracting semantic relations between concepts from both documents and topic statements. This component produces concept-relation-concept triples which are passed to the **Conceptual Graph Generator** which converts these triples into the CG formalism (Sowa, 1984). The resultant CGs are passed to the **Conceptual Graph Matcher**, which measures the degree to which a particular topic statement CG and candidate document CGs share a common structure, and ranks the documents accordingly.

The five modules in DR-LINK have well specified interfaces, making it possible for some of the modules to be re-combined, when appropriate, in a different order for a more advantageous flow of processing. For example, the Subject Field Coder can produce vectors for any-size unit of text (e.g. a sentence, a paragraph, a discourse-level text-type component, or the full document). Therefore, the Subject Field Coder can create an SFC representation for the full document *before* the text has been decomposed into its constituent discourse-level components by the Text Structurer, or the Subject Field Coder can be run on the document after the Text Structurer has recognized the discourse level components of a text, and can therefore produce separate vectors for the differing types of information (e.g. current event, past event, opinion, potential future event) contained in the various discourse-level components (e. g. Main Event, History, Evaluation, Expectation) within a newspaper text. This permits an SFC-vector of a particular topic statement to be matched to the representation for just that component whose content is most likely to be appropriate.

Another vital aspect of our approach which is evidenced in the various semantic enrichments (e.g. Subject Field Codes, discourse components, concept-relation-concept triples, Conceptual Graphs) added to the basic text, is the real attention paid to representation at a deeper than surface level. That is, DR-LINK deals with lexical entities using more conceptually-based syntactic groupings. For example, complex nominals will be processed as meaningful multi-word constituents because the combination of individual terms in complex nominals conveys quite different meanings than if the individual constituents were individually interpreted. In addition, verbs are represented in case-frames so that the other lexical entities in the sentence which perform particular semantic roles in conjunction with the verb are represented according to these semantic roles. Also, the very rich semantic data (e.g. location, purpose, nationality) that is conveyed in the formulaic, appositional phrases typically accompanying proper nouns are represented in such a way that the semantic relations implicitly conveyed in the appositions are explicitly available for more refined matching and the creation of CGs which contain this relational information. In each of these three examples, relations are important in that they contextually bind concepts which otherwise would be treated as if they were independent of each other. To accomplish this task, given a text database, DR-LINK extracts important relations by relying on relation-revealing formulae (RRF) that are patterns of linguistic (lexical, syntactic, and semantic) clues by which particular relations are detected.

## 2. Detailed System Description

Since our system is modular in design, with well-defined boundaries between the various

modules, the system description will also be organized according to these same divisions. The documents and the topic statements are analyzed in basically similar ways by the system's modules, with a few exceptions which will be detailed below within that module's description.

2.a. Pre-Processing

We have chosen to perform rather substantive pre-processing of the raw text that is received from DARPA because much of our later processing is dependent on clean, well-demarcated text. For example, we identify sub-headlines, and embedded figures, as well as identifying and restoring correct sentence boundaries. Also, we identify multiple stories within a single document so that each separate story can have its own SFC vector produced which will be representative of just that one story, and so that accurate text structuring can be accomplished at the individual story level.

The text is then processed by the POST part-of-speech tagger (Meteer et al, 1991) loaned to us by BBN, that stochastically attaches a part-of-speech tag to individual words. The part-of-speech tagged text is then fed into a bracketer, a deterministic finite state automaton that adds several different types of brackets for linguistic constituents (e.g. noun phrases, prepositional phrases, clauses, etc.) essential for several tasks in our system.

2.b. Subject Field Coder

The Subject Field Coder (SFCer) produces a summary-level semantic representation of a text's contents that is useable either for ranking a large set of incoming documents for their broad subject appropriateness to a standing query, or for dividing a database into clusters of documents on the same topic. One important benefit of the the SFC representation is that it implicitly handles both the synonymy and polysemy problems which have plagued the use of NLP in I.R. systems because this representation is one level above the actual words in a text. For example, Figure 2 presents a short WSJ article and a humanly readable version of the normalized SFC vector which serves as the document's semantic summary representation.

---

*A U. S. magistrate in Florida ordered Carlos Lehder Rivas, described as among the world's leading cocaine traffickers, held without bond on 11 drug-smuggling count. Lehder, who was captured last week in Colombia and immediately extradited to the U.S., pleaded innocent to the charges in federal court in Jacksonville.*

| | | | |
|---|---|---|---|
| LAW | .2667 | SOCIOLOGY | .1333 |
| BUSINESS | .1333 | ECONOMICS | .0667 |
| DRUGS | .1333 | MILITARY | .0667 |
| POLITICAL SCIENCE | .1333 | OCCUPATIONS | .0667 |

Fig. 2: Sample WSJ document and its SFC representation

---

The SFCer uses the Subject Codes from Longman's Dictionary of Contemporary English (LDOCE) to produce this semantic representation of a text's contents. The machine-readable tape of the 1987 edition of LDOCE contains 35,899 headwords and 53,838 senses, for an average of 1.499 senses per headword plus several fields of information not visible in the hard-copy version

which are extremely useful in natural language processing tasks, e. g. the Subject Codes. The Subject Codes are based on a classification scheme of 124 major fields and 250 sub-fields. Subject Codes are manually assigned to words in LDOCE by the Longman lexicographers. There is a potential problem, however, with the Subject Code assignments which become obvious when an attempt is made to use them computationally. Namely, a particular word may function as more than one part of speech and each word may also have more than one sense, and each of these entries and/or senses may be assigned different Subject Codes. This is a slight variant of the standard disambiguation problem, which has shown itself to be nearly intractable for most NLP applications, but which needed to be successfully handled if DR-LINK was to produce correct semantic SFC vectors.

We based our computational approach to successful disambiguation on a study of current psycholinguistic research literature from which we concluded that there is no single theory that can account for all the experimental results on human lexical disambiguation. We interpret the these results as suggesting that there are three potential sources of influence on the human disambiguation process:

**Local context** - the sentence containing the ambiguous word restricts the interpretation of ambiguous words

**Domain knowledge** - the recognition that a text is concerned with a particular domain activates only the senses appropriate to that domain

**Frequency data** - the frequency of each sense's general usage affects its accessibility

We have computationally approximated these three knowledge sources in our disambiguator. We consider the 'uniquely assigned' and 'high-frequency' SFCs of words within a single sentence as providing the local context which suggests the correct SFC for an ambiguous word. The SFC correlation matrix which was generated by processing a corpus of 977 Wall Street Journal (WSJ) articles containing 442,059 words, equates to the domain knowledge (WSJ topics) that is called upon for disambiguation if the local context does not resolve the ambiguity. And finally, ordering of SFCs in LDOCE replicates the frequency-of-use criterion. We implement the computational disambiguation process by moving in stages from the more local level to the most global type of disambiguation, using these sources of information to guide the disambiguation process. The work is unique in that it successfully combines large-scale statistical evidence with the more commonly espoused local heuristics.

We tested our SFC disambiguation procedures on a sample of twelve randomly selected WSJ articles containing166 sentences consisting of 1638 words which had SFCs in LDOCE. The system implementation of the disambiguation procedures was run and a single SFC was selected for each word. These SFCs were compared to the sense-selections made by an independent judge who was instructed to read the sentences and the definitions of the senses of each word and then to select that sense of the word which was most correct. The disambiguation implementation selected the correct SFC 89% of the time (Longman's Dictionary of Contemporary English (LDOCE).

Operationally, the SFCoder tags each word in a document with the appropriate, disambiguated (SFC). The within-document SFCs are then summed and normalized to produce a vector of the SFCs representing that document. Topic statements are likewise represented as SFC vectors. In

the routing situation, each topic statement SFC vector is compared to the incoming document SFC vectors and the douments are then ranked according to similarity to the topic statement SFC vector. Either a predetermined or adjustable criterion can be used to select those documents whose SFC vectors exhibit a predetermined degree of similarity to the topic statement SFC vector. This set is then passed to later system components for more refined representation and matching.

For use with retrospective or ad hoc queries, the SFC vectors are clustered using Ward's agglomerative clustering algorithm (Ward, 1963) to form classes in the document database. For retrieval, queries are likewise represented as SFC vectors and then matched to the prototype SFC vector of each cluster in the database. Clusters whose prototype SFC vectors exhibit a predetermined criterion of similarity to the query SFC vector are passed on to other system components for more computationally expensive representation and matching (Liddy, Paik, & Woelfel, 1992).

2.c. Text Structurer

The purpose of the Text Structuring module in DR-LINK is to delineate the discourse-level organization of each document's contents so that those document components where the type of information suggested by the topic statement is most likely to be found, can be selected for higher weighting. For example, in newspaper texts, opinions will be found in EVALUATION components, basic facts of the news story will be found in MAIN EVENT components, and predictions will be found in EXPECTATION components. The Text Structurer produces an enriched representation of each document by decomposing it into these smaller, conceptually labelled components. In parallel, the Topic Statement Processor evaluates each topic statement to determine if there is an indication that a particular component in the documents should be more highly weighted when matched to the topic statement representation. For example, topic statement indicator-terms such as *predict* or *anticipate* or *proposed* reveal that the time frame of the event being searched for must be in the future, in order for the document to be relevant. Therefore, documents in which this event is reported in a piece of text which has been marked by the Text Structurer as being either EXPECTATION or MAIN, FUTURE would be ranked more highly than those in which this event is reported in a different component.

Operationally, DR-LINK evaluates each sentence in the input text, comparing it to the known characteristics of the prototypical sentence of each component of the text-type model, and then assigns a component label to the sentence. For the newspaper text-type model, we took as a starting point, the hierarchical newspaper text model proposed by van Dijk (1988). With this as a preliminary model, several iterations of coding of a sample of 149 randomly chosen Wall Street Journal articles from 1987-1988 resulted in a revised News Schema which organized van Dijk's terminal node categories according to a more temporally oriented perspective. The News Schema Components account for all the text in the sample of articles. The components are: CIRCUMSTANCE, CONSEQUENCE, CREDENTIALS, DEFINITION, ERROR, EVALUATION, EXPECTATION, HISTORY, LEAD, MAIN EVENT, NO COMMENT, PREVIOUS EVENT, REFERENCES, and VERBAL REACTION.

The process of manually coding the sample also served to suggest to us that during our intellectual decomposing of texts, we were in fact relying on six different types of linguistic information to make our decisions. The data from the sample set which could be used to provide the raw data for these evidence sources was then analyzed statistically and translated into

computationally recognizable text characteristics to be used by the Text Structurer to assign a component label to each sentence. Briefly defined, the six sources of evidence used in the Text Structurer are:

Likelihood of Component Occurring - The unit of analysis for the first source of evidence is the sentence and is based on the observed frequency of each component in our coded sample set.

Order of Components - This source of evidence relies on the tendency of components to occur in a particular, relative order determined by calculating across the coded files of the sample documents, looking not at the content of the individual documents, but the component labels. The results are contained in two 19 by 19 matrices, one for probability of which component follows a given component and one for probability of which component precedes a given component.

Lexical Clues - The third source of evidence is a set of one, two and three word phrases for each component. The set of lexical clues for each component was chosen based on observed frequencies and distributions. We were looking for words with sufficient occurrences, statistically skewed observed frequency of occurrence in a particular component, and semantic indication of the role or purpose of each component.

Syntactic Sources - We make use of two types of syntactic evidence: 1) typical sentence length as measured in average number of words per sentence for each component; 2) individual part-of-speech distribution based on the output of the part-of-speech tagging of each document, using POST, a part-of-speech tagger loaned to us by BBN (Meteer et al, 1991). This evidence helps to recognize those components which, because of their nature, tend to have a disproportionate percentage of words of a particular part of speech.

Tense Distribution - Some components, as might be expected by their name alone, tend to contain verbs of a particular tense more than verbs of other tenses. For example, DEFINITION sentences seldom contain past tense, whereas the predominate tense in HISTORY and PREVIOUS EVENT sentences is the past tense, based on POST tags.

Continuation Clues - The sixth and final source of evidence is based on the conjunctive relations suggested in Halliday and Hasan's Cohesion Theory (1976). The continuation clues are lexical clues which occur in a sentence-initial position and which were observed in our coded sample data to predictably indicate either that the current sentence continues the same component as the prior sentence or that there is a change in the component.

These evidence sources for instantiating a discourse-level model of the newspaper text-model have been incorporated in the Text-Structurer, which evaluates each sentence of an input newspaper article against these six evidence sources for the purpose of assigning a text-level label to each sentence. The implementation uses the Dempster-Shafer Theory of Evidence Combination (Shafer, 1976) to coordinate information from the very complex matrices of statistical values for the various evidence sources which were generated from the intellectual analysis of the sample of 149 WSJ articles (Liddy, Paik, McVearry & Yu, In press).

Operationally within DR-LINK, each document is processed a sentence at a time and each source of evidence assigns a number between 0 and 1 to indicate the degree of support that evidence source provides to the belief that a sentence is of a particular news-text component. Then, a simple supporting function for each component is computed and the component with the greatest

support is selected as the correct tag for that sentence.

To convey how the Text Structurer output is used by DR-LINK, Figure 3 presents a Topic Statement which is highlighted to show its implicit request for future-oriented prediction information. This need for a specific type of information is recognized by the Topic Statement Processor which maps this need for future-oriented information to EXPECTATION or MAIN, FUTURE components in documents. The Text Structure Matcher then searches for documents with these components, as exemplified in the document in Figure 4, which shows a relevant WSJ article, as structured by the DR-LINK component, in which the required information occurs in sentences which have been correctly tagged as EXPECTATION.

2.d. Relation-Concept Detector

The main function of the RCD module is to extract relations that connect concepts that otherwise would be treated as isolated and independent. For example, a relation REASON can be extracted from phrases like '*because of*', '*as a result of*', and '*due to*', which form lexical RRF, in order to connect the two constituents occurring before and after the phrases. It should be noted that the RRF we are developing are domain-independent since they are based on fairly universal linguistic clues rather than a domain model. There are several types of RRF derived from a variety of linguistic constructs including verb-oriented thematic roles, complex nominals, proper noun appositions, nominalized verbs, adverbs, prepositional phrases, and some other ad hoc patterns revealing relations that appear in the literature (cf. Somers, 1987) and are expected to be suitable for IR purposes based on our preliminary analysis of the topic statements. We intend to conduct an extensive study of usefulness of individual relations as part of our failure analysis.

With different types of RRF stored in a knowledge-base, we go through multiple stages of partial linguistic analyses, as opposed to a holistic syntactic processing followed by a semantic interpreter, to extract relations and generate CGs eventually. With the tagged, bracketed, and structured text as the input, various sub-modules in the RCD selectively detect implicit relations as well as concepts being connected, by focusing on occurrences of patterns of interest found in the knowledge base and by bypassing portions of text irrelevant to the relation extraction tasks. The output of the RCD component is a set of concept-relation-concept triples where concepts are derived often from content-bearing words and relations from non-content words or indirectly from the linguistic structure by consulting the knowledge base.

For example, the Proper Noun (PN) apposition category of RRF will help categorize the many occurrences of PNs in text and determine semantic relations between a PN and the apposition which either precedes or follows it. For example, in the following sentence fragment, apposition RRF will recognize and categorize the LOCATION relation and the PRODUCT relation of the PN, General Development:

*"... General Development, **a Miami-based developer of planned communications,...**"*

General Development -> (LOCATION) -> Miami
General Development -> (PRODUCT/SERVICE) -> developer of planned communications

As another example of processing in the RCD module, the Case Frame Handler will, given a sentence fragment which has been processed by the tagger and bracketer:

Tipster Topic Description: 008

Topic: Economic Projections

Description: Document will contain quantitative projections of **the future value** of some economic indicator for countries other than the U.S.

Narrative: To be relevant, a document must include **a projection** of the value of an economic indicator (e.g., gross national product (GNP), stock market, rate of inflation, balance of payments, currency exchange rate, stock market value, per capita income, etc.) for the coming year, for a country other than the United States.

Concepts:

1. inflation, stagflation, indicators, index

2. signs, projection, forecast, future

3. rise, shift, fall, growth, drop, expansion, slowdown, recovery

4. %, billions

5. Not U.S.

Nationality: Not U.S.

Time: **Future**

Fig. 3: Sample Topic Statement

HEADLINE West Germany's Central Bank Cuts Key Rate: Little Impact Is Expected On Currency Markets Or Nation's Economy

LEAD The West German central bank, as expected, cut the interest rate it charges on securities-repurchase agreements with banks, a move that appears likely to fall short of its goal of supporting the dollar by widening the gap between German and U.S. interest rates.

MAIN The Bundesbank yesterday called for interest-rate tenders on 28-day securities-repurchase agreements at a minimum rate of 3.5%, well below the 3.8% rate it had been asking since January.

**EXPECT. Market experts estimate that the actual rate to emerge from bidding today is likely to be closer to 3.6% or 3.65%.**

**EXPECT. But they expect the Bundesbank to use later tenders to continue to push the rate lower, possibly to 3.3%.**

DEFIN. The repurchase facility is the central bank's principal open-market means for refinancing the banking system with funds backed by securities.

**EXPECT. Yesterday's Bundesbank action signals broad declines in capital-market interest rates, and could lead to lower credit rates charged by commercial banks.**

MAIN The West German move complements a recent lowering of short- term interest rates in Japan and a tightening of credit by the U.S. Federal Reserve Board.

MAIN However, foreign-exchange traders said the Bundesbank's action didn't have any immediate effect on the currency market and that small interest-rate movements are unlikely to significantly outweigh other factors weakening the dollar, including the U.S. trade deficit.

Fig. 4: Wall Street Journal article Relevant to T.S. 8

*"Venezuela and its creditor banks agreed yesterday to restructure $3 billion of its foreign debt ...."*

the verb `agree' triggers a set of case frames in the knowledge base with the additional information provided by the preposition `to' followed by another verb, the correct case frame (i.e. RRF for case relations) will be chosen:

(((A X S) ((AT ? to+V) (CT ? that)))

where the first element of each of the two list indicates the relation (e.g. A for AGENT, AT for ACTIVITY, and CT for CONTENT), the second specifies the LDOCE semantic restriction (e.g. X for `abstract or human), and the last shows the syntactic or lexical clue that must be satisfied in order for the case frame to be instantiated. In this example, S is for a subject, to+V for an infinitive, and `that' for a relative clause. The two sub-lists, (AT ? to+V) (CT ? that), indicates that only one of them is triggered at a given time. Eventually the following set of triples are generated:

[agree] -> (A) -> [country: Venezuela]
[agree] -> (A) -> [creditor_bank]
[agree] -> (AT) -> [restructure]

where the concept node [country: Venezuela] is formed by applying rules from the Proper Noun knowledge base and processor.

2.e. Conceptual Graph Generator

Given the triples generated in the RCD module by applying various types of RRF, the next step is to merge them to form a complete CG for a sentence, paragraph, or an SRF component. The resulting CGs consist of not only concepts and relations but also instantiations or referents of concepts, which are usually derived from proper nouns like company names but can be another CG derived from a nested clause, thereby allowing for nested CGs. Since the different types of RRF in the knowledge base are developed and applied in the RCD modules in a more or less independent manner, a form of conflict resolution is necessary in this component.

Given the set of concept-relation-concept triples as shown above and another set derived from the verb `restructure' with a frame ((A X S) (P W O)):

[restructure] -> (A) -> [country: Venezuela]
[restructure] -> (A) -> [creditor_bank]
[restructure] -> (P) -> [debt]

where A and P are for `agent' and `patient' relations, and

[debt] -> (ME) -> [money]
[debt] -> (CH) -> [foreign]

are produced by a special handler within the RCD, where ME and CH stand for `measure' and `characteristics', the CG generator produces the following CG for the sentence:

```
[agree] -
(A) -> [country: *1 Venezuela]
(A) -> [creditor_bank: *2]
(AT) -> [restructure] -
      (A) -> [country: *1 Venezuela]
      (A) -> [creditor_bank: *2]
      (P) -> [debt] -
           (ME) -> [money]
              (CH) -> [foreign].
```

where *1 and *2 indicate that the nodes with the same number represent the same concept.

While this process of extracting relations and constructing CGs is applied both to documents and topic statements, the latter require additional processing to capture unique features of information needs often found in the topic statement. Accordingly, CGs generated from topic statements have such additional features as importance weights on concept and relation nodes and ways of indicating whether an instantiation of a concept must exist in relevant documents. This specialized processing, in comparison with document processing, is accomplished by treating topic statements as a sub-language and building a model for them. For example, some information on weights is revealed by phrases like "... optional" and "... must exist ..." whereas the need for an instantiation of a concept id indicated by phrases like "Identification of the company must be included".

While CG theory provides a framework in which IR entities can be represented adequately, much of the representation task involves intellectual analysis of topic statements and documents so that we capture and store concepts and relations that are ontologically adequate for IR. For example, it is essential to choose, organize and classify a restricted set of relations in such a way that they facilitate matching and inferencing with two CGs representing a document and a topic statement. The efficacy of the relations we have chosen will be determined with full experiments and failure analyses.

### 2.f. Conceptual Graph Matcher

The main function of the CG matching component is to determine the degree to which two CGs share a common structure and score each document with respect to the topic statement. This is accomplished by employing techniques necessary to model plausible inferences with CGs (Myaeng & Khoo, 1992). In order to allow for approximate matching between concept nodes or relation nodes, we have developed a matrix that represents similarities among relations being used in CG representation, as well as some concepts. Our goal is to enhance both precision and recall. By exploiting the structure of the CGs and the nature of the relations, we attempt to meet the specific information needs in topic statements. By allowing for partial matching (e.g. between `debt' and `bank debt') and inexact matching (e.g. between `debt' and `loan' and between `CO-AGENT' and `AGENT') at the node level, we can increase recall.

For CG matching, we first developed and implemented a base algorithm that is flexible enough to allow for various types of partial matching between two CGs and ran experiments to test its practicality (Myaeng and Lopez-lopez, 1992). While the general subgraph isomorphism problem is known to be computationally intractable, matching CGs containing conceptual information (i.e. labels on nodes) appears to be practical. With improved understanding of the

nature of RRF and the exact features of the representation of documents and topic statements, we have developed and implemented heuristics for scoring documents for their relevancy with respect to a given topic statement, and added them to the base algorithm. With the rich representation of topic statements and documents, the expanded matching component currently is capable of discriminating documents based on the availability of rather unusual information needs specified in a topic statement, such as a certain status of an event or a need to contain a specific entity (e.g. company name) satisfying a certain condition. Also it facilitates the process of reliably identifying "hot spots" in retrieved documents.

An example of how a matching between document CG and a topic statement CG is shown in Figure 5, where the entire CG represents the example sentence discussed above and the topic statement sentence:

> *"... a current debt rescheduling agreement ... between a debtor developing country and one or more of its creditors, commercial and/or official. It will identify the debtor country and the creditor(s), the payment time period ..., and the interest rate, ..."*

The dark area and the individual nodes with gray shades represent the matched parts: a connected sub-CG that is the main contributor for the final score and some single node matches, respectively. It should be noted that information on the relative importance of the text components delineated by the Text Structure component with respect to the topic statement is incorporated in the scoring heuristics.

An independent module under development, which will have direct bearing on the final results of matching and which will be added at a later stage to determine its efficacy, is RIT (Roget's International Thesaurus) Coder. The goal is to simulate the use of a type hierarchy in the CG theory by replacing lexical terms in concept nodes in our representation with RIT semi-colon group numbers, each of which represent a set of semantically similar words and phrases in its hierarchy. Our approach is to use the words surrounding the target word to be replaced with an RIT code as context words by which we try to disambiguate the sense of the target word and find the exact location in the RIT. While this approach is to increase both recall and precision at the same time through implicit expansion of terms and sense disambiguation, we will have to see the sensitivity of incomplete disambiguation to the overall retrieval results.

## 3. Testing and Results

Although the DR-LINK entry in TREC was not tested in the same manner as the other systems, the status of the system and the testing which was done were consistent with the milestones that had been established in consultation with our TIPSTER contractor. DR-LINK was not tested as a full system due to the fact that DR-LINK is based on several theories that have never before been implemented in an information retrieval system and none of the system's components were in even the design stage of development when the project began. As a result of these facts, our system is not yet fully implemented. The results that follow are on the three modules (Subject Field Coder, Text Structurer, Conceptual Graph Matcher) that have been implemented to date. The full system will be tested at the eighteenth month meeting of TIPSTER.

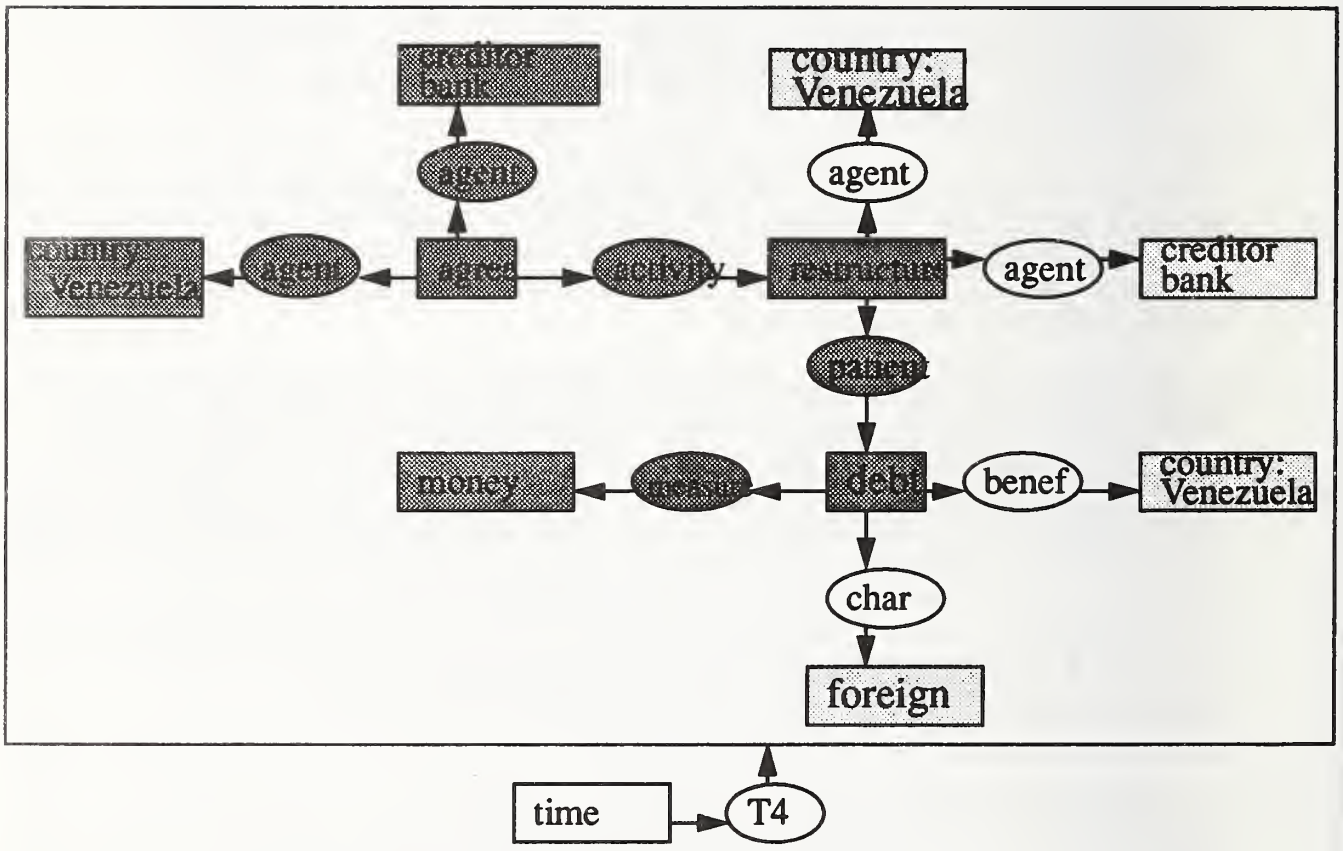



Fig. 5: Example of CG matching result

### 3.a. Subject Field Coder Testing

The Subject Field Coder can be evaluated in its filtering function in the following way: The SFC vectors of documents in a database are compared to a topic statement vector and, for each topic statement, ranked according to similarity. The question then is, how far down this ranked list would the system need to proceed in order to include all the relevant documents in the set of documents that was passed on to the next system module? This testing procedure has been run using the WSJ and Ziff collections on Disk 1 and Topic Statements 1 to 50 provided by TREC. In the Wall Street Journal database, results showed that on average across the 50 topic statements, all the relevant documents were ranked in the top 28% of the list. Therefore, on average, 72% of the WSJ database did not need to be further processed by the later modules in the system. In the Ziff database, on average across the 50 topic statements, all the relevant documents were ranked in the top 66% of the database, a much poorer result. Therefore, on average, 34% of the Ziff database need not be further processed. When Topic Statements 51 to 100 were searched on the WSJ database, all the relevant documents were ranked in the top 32% of the list.

Error analysis of the Ziff results, has revealed the cause of the poorer performance on that database. Namely, for several of the queries, documents were judged relevant only if they contained the particular proper noun mentioned in the topic statement (e.g. OS/2, Mitsubishi, IBM's SAA standards, etc.). Given these types of topic statements, the most appropriate search approach for a system would be keyword matching, which is not at all the type of matching that is done using the SFC representation. The SFCs represent a document at a higher level of abstraction, not at the keyword level. That is documents which discuss a particular computer, will have a strong weighting of the Data Processing slot on the SFC vector, but no means for matching on **a particular** computer name. Therefore, the error analysis showed that the SFC performance was hampered by its inability to match at the level of a specific company name or product. Fortunately, we do have at hand the means to improve the results, as we are in the process of incorporating a second-level of document ranking using Proper Noun processing algorithms. Results using this extended representation will be available at the eighteenth month TIPSTER meeting.

### 3.b. Text Structurer Testing

The Text Structurer was tested using five of the six evidence sources, as we have not yet implemented the algorithms for incorporating evidence from the Continuation Clues. We tested the Text Structurer on a set of 116 WSJ documents, consisting of several thousand sentences. This first testing resulted in 72% of the sentences being correctly identified. An additional hand simulation of one small, heuristic adjustment was tested and improved the system's performance to 74% of the sentences being correctly identified. A second run of a smaller sample of sentences resulted in 80% correct identification of components for sentences. Ongoing efforts at improving the quality of the evidence sources used by the Text Structurer, plus the incorporation of the Continuation Clue evidence, promise to improve these results significantly. It should be remembered, as well, that the automatic discourse structuring of documents has not been reported elsewhere in the literature, so that this very new type of text processing is in its infancy and likely has much room for future improvements.

### 3.c. Conceptual Graph Matcher Testing

As the first prototype, the CG Matcher was implemented with a set of scoring heuristics whose theoretical basis is on the Dempster-Shafer theory of evidence. The score for a document is computed progressively from node-level evidence through multiple stages to generate a score for text units of different granularities.

In order to test the feasibility of the prototype module, we have run it with manually generated CGs for twenty documents and five topic statements for which we have relevance judgments. While it is premature to draw any conclusions on the efficacy of the matching algorithm and the heuristics, mainly due to the size of the data set and the stage of the development, the results were encouraging and have provided us with much insight on how the scoring heuristics need to be tuned.

## 4. Conclusions

This paper on the DR-LINK system should be considered a report on a work in progress, since we did not have a fully developed system at the time of the TREC testing. However, we do believe that the three system components which were tested perform quite respectably, given their innovativeness. Continued development and feedback from the TREC results will provide much more refined versions of these system modules. In addition, two system modules remain to be developed and the full system, which is quite synergistic in its approach to achieving its goals, remains to be integrated and tested as a full system. The Relation Concept Detector and Conceptual Graph Generator modules are being implemented in tandem and, when completed, will make the DR-LINK system fully operational. Full system testing will be conducted for the eighteen month TIPSTER testing.

In the interim, our goal in this paper has been to describe the five unique modules which comprise DR-LINK and which, in combination, promise to provide a full system which has the necessary filtering power to make later processing more accurate and the depth of linguistic processing required to provide real conceptual level matching and retrieval.

## 5. References


Halliday, M. A. K. & Hasan, R. (1976). Cohesion in English. London, Longmans.

Liddy, E.D. & Paik, W. (1992). Statistically-Guided word sense disambiguation. In Proceedings of AAAI Fall Symposium Series: Probabilistic approaches to natural language. Menlo Park, CA: AAAI.

Liddy, E.D., Paik, W., McVearry, K. & Yu, E. (In press). Automatic discourse-level structuring of newspaper texts: Empirical testing of a model.

Liddy, E.D., Paik, W. & Woelfel, J. (1992). Use of subject field codes from a machine-readable dictionary for automatic classification of documents. Proceedings of 3rd ASIS Classification Research Workshop.

Meteer, M., Schwartz, R. & Weischedel, R. (1991). POST: Using probabilities in language processing. Proceedings of the Twelfth International Conference on Artificial Intelligence. Sydney, Australia.

Myaeng, S. H. (1992) Using conceptual graphs for information retrieval: a framework for representation and flexible inferencing. Proceedings of Symposium on Document Analysis and Information Retrieval, Las Vegas, March 16-18.

Myaeng, S. H. & Khoo, C. (1992). On uncertainty handling in plausible reasoning with conceptual graphs. Proceedings of 7th Workshop on Conceptual Graphs, Las Cruces, NM, July, 1992.

Myaeng, S. H. & Lopez-lopes, A. (1992). A conceptual graph matching: a flexible algorithm and experiments. Journal of Experimental and Theoretical Artificial Intelligence, Vol. 4, 107-126.

Shafer, G. (1976). A mathematical theory of evidence. Princeton, NJ: Princeton University Press.

Somers, H. L. (1987). Valency and Case in Computational Linguistics. Edinburgh: Edinburgh University Press.

Sowa, J. (1984). Conceptual Structures: Information Processing in Mind and Machine. Reading, MA: Addison-Wesley.

van Dijk, T. (1988). New analysis: Case studies of international and national news in the press. Hillsdale, NJ: Lawrence Earlbaum Associates.

Ward, J. (1963). Hierarchical grouping to optimize an objection function. Journal of the American Statistical Association. 58, p. 237-254.

# WORDIJ: A WORD-PAIR APPROACH TO INFORMATION RETRIEVAL

James A. Danowski

University of Illinois at Chicago

## CONCEPTUAL MODEL

WORDij is a system based on a linkage or network model for representing textual information. The fundamental unit of analysis is the word pair, or bi-gram phrase, rather than the individual term. WORDij also takes a local approach to term cooccurrence. Systems such as SMART historically used the entire document as the field within which to define term cooccurrence. More recent research has suggested that defining cooccurrence within smaller text units such as paragraphs may be better [Salton & Buckley 91]. WORDij is even more local in focus. It defines cooccurrence of terms within three word positions (after dropping stop words). In addition, WORDij uses direct and indirect pair information to compute shortest paths among words in retrieved documents. This counts both direct and indirect matches between queries and documents.

Consider a query Q containing the phrase {t1, t3} and a document D containing the phrases {t1, t2}, and {t2, t3} but not the phrase {t1, t3}. Existing algorithms [Salton & Buckley 91, Croft, Turtle & Lewis 91, Fagan 89] would not consider the dependency between t1 and t3 as there is no match for the phrase. However, tree-dependency models [van Rijsbergen 77; Yu, Buckley, Lam and Salton 83] recognize such indirect dependencies and produce a formula to compute the degree of dependency between t1 and t3. The WORDij approach considers not only the direct phrases but also indirect phrases.

## METHODS

TREC work was begun using a network of Sun workstations in the Database and Information Systems Laboratory in the Electrical Engineering and Computer Science Department at the University of Illinois at Chicago. Because the lead Research Assistant, Nainesh Khimasia, died during the project, software development using C and Unix tools was impeded. Earlier generations of tools had been optimized for an IBM mainframe computer, so work was switched to that platform. The machine used was an IBM 3090/300J platform running VMXA, CMS. A virtual machine CPU size of 16meg was used along with three gigabytes of disk space. The CPU clock speed is rated at 14.5 nanoseconds, or 69 MHz.

We modified earlier generations of WORDij software written in SPITBOL [Danowski 82, Danowski & Andrews 85]. These modifications consisted mainly of replacing some SPITBOL code where possible with CMS PIPELINE code, because it runs approximately one thousand times faster. The *.Z text files were uncompressed using a compress utility on CMS that works with Unix based compressed files. WORDij code was run on each uncompressed text file, generating an inverted file of word pairs by document identification numbers. All word pairs occurring only once in each document were dropped to save disk space.

No spell checking, stemming, morphological analysis, parsing, or tokenizing was done. A stop list of 631 words was used, comprised of the 570 stop words in SMART v.10 and some additional stop words forming the markup format of the raw text. Processing time to create the word pair index averaged three minutes per file.

Ad hoc queries were automatically processed in the same way as raw documents, except that no single pairs were dropped. Query text used to generate word pairs for matching included all text provided, except the factors and definitions, and concepts numbered higher than two. Total CPU seconds to build a query averaged .26 seconds. For the ad hoc queries, nothing further was done to them, either automatically or manually.

For the routing topics, queries were also constructed automatically, but in a different way. The training sets of relevant and irrelevant documents were separately analyzed to identify all word pairs that occurred in the relevant set but not in the irrelevant set. These unique relevant word pairs were used as routing queries.

PIPELINE matching of the query pairs against the pair files for each text file executed in approximately 16 milliseconds per file per 100 sets of query pairs. This meant that to run all 100 queries against the entire collection took approximately five hours of PIPELINE processing on the word pair index files, or three minutes per query.

Time constraints precluded completing a word and word-pair by document count on the entire collection for inverse document frequency or entropy word and word pair weighting. Retrieved documents were ranked from 1 to 200 by counting the number of matching pairs each document had to the query. Frequency of pair occurrence in documents was not used to weight except in breaking ties at the 200 document-rank threshold.

Time limitations also prevented full implementation of the indirect matching process. Only directly matching pairs were used for the main analysis to produce the results. Indirect matching was, however, later tested. This will be described after presentation of the basic results.

## RESULTS

WORDij results were greater than or equal to the median levels of performance for seven topics. Our results were within one standard deviation on 55 topics, and within two standard deviations on 82 topics. Performance was significantly lower than the median for 14 topics, as judged by counting topics whose results were greater than two standard deviations below the median. Table 1 lists the topics in two categories, those that were better than or equal to the median, and those that were significantly below the median.

*Table 1*: Topic Results Ordered by Performance

| TOPIC | Difference (median - result) | |
|---|---|---|
| **Better than or Equal to Median** | | |
| 66 | -.08980 | Natural Language Processing |
| 29 | -.04540 | OS/2 problems |
| 94 | -.03180 | Computer-aided Crime |
| 95 | -.00800 | Computer-aided Crime Detection |
| 18 | .00000 | Global Stock Market Trends |
| 44 | .00000 | What Makes CASE succeed or fail |
| 88 | .00000 | Crude Oil Price Trends |
| 100 | .00000 | Controlling High Tech Transfer |
| 50 | .00250 | Virtual Reality Military Apps. |
| **Significantly Below Median (Failures)** | | |
| 22 | .19590 | Legal Repercus.-Agrochemicals |
| 58 | .20740 | Rail Strikes |
| 37 | .21290 | Role of Minis and Mainframes |
| 20 | .21770 | Superconductors |
| 77 | .23290 | Poaching |
| 17 | .24350 | Japanese Stock Market Trends |
| 93 | .24560 | What Backing Does the NRA Have |
| 13 | .24780 | Drug Approval |
| 54 | .26840 | Satellite Launch Contracts |
| 51 | .29490 | Airbus Subsidies |
| 10 | .33340 | Space Program |
| 70 | .35440 | Surrogate Motherhood |
| 78 | .38240 | Greenpeace |
| 21 | .48710 | Counternarcotics |

### Failure Analysis

#### Query Style.

Several kinds of failure analysis were performed. To investigate whether stylistic features of queries were associated with performance, we computed the following variables for each query using the shareware program, PC-STYLE:

- Number of Sentences
- Number of Words
- Words per sentence
- Percentage of long words
- Percentage of personal words
- Percentage of action verbs
- Average number of syllables per word
- Reading grade level

These variables were correlated with a criterion variable, which was the difference between the median and our result. We subtracted for each query our obtained result from the median result on the 11-point averages of recall-precision contained in the official results across systems for the test queries 51-100. Table 2 displays these correlations. None of them are statistically significant at the .01 level. A second criterion variable was created to represent whether the query was in the "failed" category, greater than two standard deviations below the median. A dummy variable was created for each query using zero to represent success and one to represent failure. Correlations of the style variables were also computed with the failure criterion. No correlations were significant at the .01 level. This suggests that query length, complexity, and other stylistic variables are unrelated to retrieval performance.

#### Query Words.

*Table 2*: Query style & performance correlations

| | Diff. | Failure |
|---|---|---|
| Sentences | -.1053 | -.1102 |
| Words | -.1139 | -.1616 |
| Words/sent. | .0736 | -.0004 |
| Long words | -.1574 | -.1086 |
| Personal words | .0846 | -.0046 |
| Action words | .1192 | .2690 |
| Syllables/word | -.1055 | -.0763 |
| Reading grade level | -.0256 | -.0629 |

Additional failure analysis was conducted to explore whether there were particular words associated with performance. The frequencies of all words (no stop words) for each query were correlated with both types of performance criteria: 1) continuous difference from the median and 2) failure, indicated by results significantly below median performance. Table 3 presents the correlations that were significant at the .01 alpha level or better across the 98 topics, and which occurred in at least five different topics.

*Table 3*: Query words & performance correlations

| WORD | r | No. of Topics |
|---|---|---|
| Difference | | |
| to | -.2743* | 15 |
| some | -.2480* | 10 |
| who | .3570** | 8 |
| more | .2509* | 8 |
| type | .3740** | 6 |
| following | .3069* | 6 |
| been | .2580* | 6 |
| two | .3750** | 5 |
| Failure | | |
| national | .2479* | 11 |
| system | .2479* | 9 |
| who | .3828** | 8 |
| more | .2426* | 8 |
| been | .2545* | 6 |
| two | .4100** | 5 |
| support | .2479* | 5 |

* p < .01, ** p < .001

The words 'to' and 'some' increased in frequency as performance increased, while frequency of the following words was associated with lower performance: 'who, more, type, following, been, two.' For the failure criterion, 'who, more, been, two' were also significantly associated with lower performance. In addition, 'national, system, support' were also negatively associated with it. This analysis of words from queries associated with performance suggests that the pair matching approach worked best when the documents used a domain-specific vocabulary.

### *Proper Name Identification.*

At the other extreme, topics that used more domain-general words had lower performance. In particular, queries that asked for a category of documents, such as indicated by words such as 'who' and 'type' were more likely in the failure category. Words including: 'system, national, following, been, and two' were also associated with higher failure rates. This suggests that proper noun compounds may require special treatment. The names of organizations, products, locations, etc. cannot apparently be easily identified through direct pair matching when these specific proper nouns are not contained in the query. When such specific results are called for by a query, special procedures are probably desirable for identification of proper nouns in documents that match on other query pairs.

### *Domain Specificity of Words.*

An additional implication is that query expansion may be fruitful when dealing with domain-transcendent words. Through use of thesauri or databases such as WordNet, alternative word meaning senses may be disambiguated. Then synonyms specific to the proper domain could be added to the actual query pairs contained in the original raw query text.

Interestingly, queries that contained the words 'some' resulted in higher performance. This may suggest that the criteria for relevance were less stringent for such queries, in that they asked not for an exhaustive and complete fit of query to documents, but a more partial overlap. The word 'to' in queries was also associated with higher performance. This may be associated with the specificity of this word in discourse, indicating relationships of direction, degree, state, contact, possession, etc.

### *Natural Language Processing on Queries.*

Together, such query-focused results suggest that future work may benefit from performing complex natural language processing such as parsing, sense disambiguation, etc. on the queries themselves to tune them before matching. Sophisticated treatment of queries may improve performance to the point that such treatment of the raw texts themselves, which is expensive, may not add much marginal performance improvement.

### *Stemming.*

Tests were run with the training sets for three queries selected at random: 2, 26, and 49. For query 2 the difference was zero. For query 26, the relevant documents retrieved increased by 43%, while for query 49 there was a 73% improvement. Average improvement for the three queries was 37% using stemming.

### *All Pairs.*

Tests were run for three different queries to examine effects of dropping single occurring word pairs from documents. Queries 51, 71, and 78 were chosen at random. Retrieval of relevant documents increased on average by 75%, with varied results across queries. Query 51 saw relevant documents retrieved increase by 2.25 times, query 71 decreased performance by .93, and query 78 increased by 11 times, for an average of 1.75 times increase in performance.

### *Indirect Match Tests.*

The training set of documents for query 51, about Airbus subsidies, was used to test indirectness effects. One-step indirectness was assessed, meaning that two query pair words were not directly in the document, but were indirectly connected through an intermediary word.

To illustrate, here are the query pairs including the word, "aid," none of which have any direct matches in the documents:

- AID LOAN
- AID TRADE
- AID FINANCING
- AID SUBSIDIES
- AID ASSISTANCE
- AID GOVERNMENT

Table 4 contains the direct (one-step) and indirect (two-step) links that "aid" had in the documents. The leftmost pairs are direct links, while the rightmost words were directly linked only to the second word of the direct pairs, thus forming a two-step indirect link to the first word in the pair. For example, "aid" is linked to "government" only indirectly through "Airbus." Also, "aid" is linked to "subsidies" only indirectly through "Airbus." These two sets of indirect links, aid-(Airbus)-subsidies and aid-(Airbus)-government are meaningful in terms of the content of the query, which generally concerns government aid and subsidies to Airbus. If we had used only directly matching pairs, we would have missed these two conceptually meaningful sets of links. After identifying all indirect pairs in documents matching query pairs in this way, retrieval of relevant documents was 12% higher.

### *Shortest Paths.*

WORDij does not restrict detection of indirect phrases to these dual bi-gram cases. Rather, indirectness can be of n-step lengths [Danowski and Martin 79, Van Rijsbergen 77]. For example, if there is an intermediate term between two other terms not otherwise linked, then these two other terms have an indirect step linkage of two. If the connection is only through two intermediaries, then the indirect linkage is at step three, and so on. Shortest path algorithms [Gabow & Tarjan 89] find the best set of all direct and indirect links connecting all nodes in a network. Here, this is all words in the query.

We expect that indirectness at the two-step level may contribute most to recall-precision effectiveness. At larger numbers of steps the value of indirect information diminishes. This is because at the extreme lengths, every word is indirectly connected to every other word. This is equivalent to a simple within-document cooccurrence of words, such as in traditional approaches. It renders useless the local cooccurrence constraints. Note also that stop word removal from texts is necessary to represent higher degrees of indirectness. When stop words are present, they increase the connectivity of the word network.

### *Structural Equivalence and Meaning.*

In network analysis, attention to the direct links in a network is called a "cohesion" approach while examining the degree of similarity in two-step links is called "structural equivalence" [Burt 90]. Two nodes are structurally equivalent to the extent that they share the same indirect links, though they may not be directly linked themselves. For example, if word A is linked to words C,D,E and word B is linked to words C,D,E, then although A and B are not directly linked (i.e. show no cohesion), they are structurally equivalent and maximally similar because they share the same links.

Research in mathematical sociology and network analysis has found that structural equivalence is usually equal to or better than cohesion in accounting for system behavior. In text analysis using words as nodes, two words can be considered to share more meaning to the extent they have overlapping two-step links. Therefore, structural equivalence of words is meaning equivalence.

### *Latent Semantic Indexing and Indirect Pairs.*

It is interesting that another approach to indirectness is Latent Semantic Indexing (LSI) [Deerwester et al. 90; Dumais 92]. Instead of than using a network approach, however, it uses an eigenvector model. Eigenvectors represent the combined effects of direct and indirect associations among elements in the matrix. "Latent" refers

*Table 4*: Direct and indirect links to the word "aid"

| | | | FREQUENCY |
|---|---|---|---|
| aid | airbus | | 1 |
| | | fdp | 1 |
| | | back | 1 |
| | | jets | 2 |
| | | adams | 1 |
| | | board | 1 |
| | | crash | 1 |
| | | group | 1 |
| | | plans | 1 |
| | | airbus | 1 |
| | | boeing | 2 |
| | | family | 1 |
| | | german | 1 |
| | | member | 1 |
| | | planes | 2 |
| | | dispute | 1 |
| | | mandate | 1 |
| | | nations | 2 |
| | | partner | 2 |
| | | percent | 1 |
| | | program | 2 |
| | | provide | 1 |
| | | aircraft | 1 |
| | | european | 1 |
| | | products | 1 |
| | | projects | 2 |
| | | amendment | 1 |
| | | executive | 1 |
| | | industrie | 9 |
| | | initially | 1 |
| | | ministers | 1 |
| | | spokesman | 2 |
| | | structure | 1 |
| | * | subsidies | 2 |
| | | violating | 1 |
| | | consortium | 5 |
| | * | government | 1 |
| | | management | 1 |
| aid | package | | 1 |
| aid | guarantee | | 1 |
| aid | consortium | | 1 |
| | | airbus | 1 |

* These are indirect links that create pairs contained in the query pairs: aid-(Airbus)-subsidies and aid-(Airbus)-government. The other indirect links are not meaningful because they do not relate to the query at the two-step level. Nevertheless, they are listed to show the larger context of identifying meaningful indirectness.

to indirect association patterns below the manifest or direct level. Currently, eigenvector solutions to large matrices are more computationally limited than shortest path network solutions. There has been more development of large scale, parallel algorithms for shortest paths, due to the practical needs to aid routing of information in telecommunications networks. Some work, however, suggests that there is a mathematical equivalence between eigenvector and network approaches to reducing matrices of associations to a simpler underlying structure [Barnett & Richards 91].

*Shortest Path Weighting.*

Given a set of query word pairs and a list of all documents that contain each word pair--both directly and indirectly-- we can take all pairs of nodes and identify the shortest path linking them in the network. These paths are measured for length according to Euclidean distance in graph terms. Such distance is a direct function of the minimum number of link steps it requires to connect two nodes on their geodesic. Directly linked nodes have a distance of one, nodes linked through one common intermediary node have a distance of two, etc. Documents are counted that were "passed through" or "activated" as each step in the shortest path is traversed. Shortest path algorithms can find these indirect paths with large data sets provided parallel algorithms and hardware are used. We are further developing such experiments.

After IDF weighting, ranking, and selection of the best words, network analysis is conducted on the word pairs they form. The shortest paths linking every word in the set are found, and the word centrality in the network is indexed via the average of the minimum number of steps between that word and all other words in the set.

Then, for each document, it is given a weight that is based on the centrality of the words from the query it contains. The retrieved documents found along the shortest paths between all query pairs are counted and weighted by their constituent word centrality. ank ordered for each query. Documents are then rank-ordered for each query.

## CONCLUSION

Results showed that even with unexpected limitations due to the mid-project death of the lead research assistant, Nainesh Khimasia, we succeeded in processing the entire TREC collection and doing direct matching of query word pairs to document word pairs. For 15% of the topics, our results can be considered failures. Failure analysis suggests that improvements in future research may result from:

- query tuning based on natural language processing
- using special procedures for treating proper noun names for organizations, products, locations, etc.
- retaining and using word pairs occurring only once in documents
- stemming the documents and queries
- doing indirect document frequency (IDF) or entropy weighting on words and using these to weight query pairs
- computing additional weights based on shortest paths.

## ACKNOWLEDGEMENTS

The author is grateful for the contributions of the following University of Illinois at Chicago faculty, students, and staff to this project: John Andrews, Robert Goldstein, Alan Hinds, Nainesh Khimasia, Jin Hong Meng, Stephen Roy, Gary Singer, Anand Sundaram, George Yanos, and Clement Yu.

## REFERENCES

Barnett, G.A. & Richards, W.D. (1991, February). A comparison of NEGOPY'S clique detection algorithm with correspondence analysis. Paper presented to the International Social Networks Conference, Tampa, Florida.

Burt, R.S. (1990). Structure. New York: Center for Social Sciences, Columbia University.

Croft, B., Turtle, H. & Lewis, D. (1991). Proceedings of the SIGIR '91, 32-45.

Danowski, J. (1982). A network-based content analysis methodology for computer-mediated communication: An illustration with a computer bulletin board," Communication Yearbook, 6, 904-925.

Danowski, J. (1988). Organizational infographics and automated auditing: Using computers to unobtrusively gather and analyze communication. In G. Goldhaber and G. Barnett (eds.) Handbook of organizational communication (pp. 335-384). Norwood, NJ: Ablex.

Danowski, J. & Andrews, J. (1985, February). A method for automated network analysis of word cooccurrences. Paper presented to the International Social Networks Conference, San Diego.

Danowski, J. & Martin, T.H. (1979). Evaluating the health of information science: Research community and user contexts. Final report to the Division of Information Science of the National Science Foundation, no. IST78-21130.

Deerwester, S., Dumais, S.T., Landauer, T.K., Furnas, G.W. & Harshman, R.A. (1990). Indexing by latent semantic analysis. Journal of the Society for Information Science, 41:6, 391-407.

Dumais, S.T. (1992). LSI meets TREC: A status report. Paper presented to TREC.

Fagan, J. (1989). The effectiveness of a nonsyntactic approach to automatic phrase indexing for document retrieval. Journal of the American Society for Information Science, 40:2,115-132.

Gabow, H.N. & Tarjan, R.E. (1989). Faster scaling algorithms for network problems. SIAM Journal on Computing. 18(Oct),1013-36.

Salton, G. & McGill, M. (1983). Introduction to modern information retrieval. New York: McGraw-Hill.

Salton, G. & Buckley, C. (1991). Automatic text structuring and retrieval: Experiments in automatic encyclopedia searching. Proceedings of the SIG-IR '91, 21-30.

van Rijsbergen, C. (1977). A theoretical basis for the use of cooccurrence data in information retrieval. Journal of Documentation, 33,106-119.

Yu, C., Buckley, C., Lam, H. & Salton, G. (1983). A generalized term dependence model in information retrieval. Information technology: Research and development, 2,129-154.

# LSI meets TREC: A Status Report

Susan T. Dumais
Bellcore
445 South St.
Morristown, NJ 07960-1910
std@bellcore.com

## 1. Overview of Latent Semantic Indexing

Latent Semantic Indexing (LSI) is an extension of the vector retrieval method (e.g., Salton & McGill, 1983) in which the dependencies between terms and between documents, in addition to the associations between terms and documents, are explicitly taken into account. This is done by simultaneously modeling all the association of terms and documents. We assume that there is some underlying or "latent" structure in the pattern of word usage across documents, and use statistical techniques to estimate this latent structure. A description of terms, documents and user queries based on the underlying, "latent semantic", structure (rather than surface level word choice) is used for representing and retrieving information.

Latent Semantic Indexing (LSI) uses singular-value decomposition (SVD), a technique closely related to eigenvector decomposition and factor analysis (Cullum and Willoughby, 1985). A large term-document matrix is decomposed it into a set of $k$, typically 100 to 300, orthogonal factors from which the original matrix can be approximated by linear combination. Instead of representing documents and queries directly as sets of independent words, LSI represents them as continuous values on each of the $k$ orthogonal indexing dimensions. Since the number of factors or dimensions is much smaller than the number of unique terms, words will not be independent. For example, if two terms are used in similar contexts (documents), they will have similar vectors in the reduced-dimension LSI representation. The SVD technique can capture such structure better than simple term-term or document-document correlations and clusters. LSI partially overcomes some of the deficiencies of assuming independence of words, and provides a way of dealing with synonymy automatically without the need for a manually constructed thesaurus. LSI is a completely automatic method. (The Appendix provides a brief overview of the mathematics underlying the LSI/SVD method. Deerwester et al., 1990, and Furnas et al., 1988 present additional mathematical details and examples.)

One can also interpret the analysis performed by SVD geometrically. The result of the SVD is a vector representing the location of each term and document in the $k$-dimensional LSI representation. The location of term vectors reflects the correlations in their usage across documents. In this space the cosine or dot product between vectors corresponds to their estimated similarity. Since both term and document vectors are represented in the same space,

similarities between any combination of terms and documents can be easily obtained. Retrieval proceeds by using the terms in a query to identify a point in the space, and all documents are then ranked by their similarity to the query.

The LSI method has been applied to many of the standard IR collections with favorable results. Using the same tokenization and term weightings, the LSI method has equaled or outperformed standard vector methods and other variants in almost every case, and was as much as 30% better in some cases (Deerwester et al., 1990). As with the standard vector method, differential term weighting and relevance feedback both improve LSI performance substantially (Dumais, 1991). LSI has also been applied in experiments on relevance feedback (Dumais and Schmitt, 1991), and in filtering applications (Foltz and Dumais, 1992).

The TREC conference was an opportunity for us to "scale up" our tools, and to explore the LSI dimension-reduction ideas using a very rich corpus of word usage. This large collection of standard documents and relevance judgements should be a valuable IR resource and an important step in the systematic development of more effective retrieval systems.

## 2. Application of LSI to the TREC collection

### 2.1 Overview

We used existing LSI/SVD software for analyzing the training and test collections, and for query processing and retrieval. For pragmatic reasons, we divided the TREC collection into 9 subcollections - AP1, DOE1, FR1, WSJ1, ZIFF1, AP2, FR2, WSJ2, ZIFF2. Queries were passed against the appropriate subcollections, and the returns recombined to arrive at a single ranked output.

There were three main stages involved in processing documents and constructing the relevant data structures. All steps were **completely automatic** and involved no human intervention. The resulting reduced-dimension representations were used for matching and retrieval.

1. Pre-processing and indexing (extracting terms, calculating term weights, etc.)
2. Computing the SVD (number of dimensions ranged from 235-310)
3. Adding new documents and/or terms

#### *2.1.1 Pre-processing and indexing*

We did minimal pre-processing on the raw text of the TREC documents. Some markups (any text within <> delimiters) were removed, and all hand-indexed entries were removed from the WSJ and ZIFF collections. Upper case characters were translated into lower case, punctuation was removed, and white spaces were used to delimit terms. A minimum term length of 2 was used.

All terms occurring in more than one document, and not on a stop list of 439 words were used to generate a term-document matrix. We **did not** use: stemming, phrases, syntactic or semantic parsing, word sense disambiguation, heuristic association, spelling checking or correction,

proper noun identification, complex tokenizers, a controlled vocabulary, a thesaurus, or any manual indexing. The entries in the term-document matrix were then transformed using a $\log((tf_{td}+1))\times(1-entropy_t))$ weighting. The weight assigned to each term was: 1 - entropy or noise, where entropy = $1-\sum_d \frac{p_{td}\log(p_{td})}{\log(ndocs)}$, $ndocs$ is the number of documents in the collection, $t$ is the index for terms, $d$ is the index for documents, $p_{td}=\frac{tf_{td}}{gf_t}$, $tf_{td}$ is the frequency of term $t$ in document $d$, and $gf_t$ is the global frequency of occurrence of term $t$ in the collection. (For simplicity, we refer to this as the log*entropy term weight.) The transformed term-document matrix was used as input to the SVD. The results of the SVD analysis, a $k$-dimensional real vector for each term and each document and $k$ singular values, were stored in a database. The terms and their log*entropy weights were also stored in a database.

Each of the 9 subcollections was processed separately. Because of software constraints, the initial indexing and SVD analysis were done on a random subset of 20,000-57,000 documents. The remaining documents were added into the resulting data structure as described below. (We have recently completed an SVD analysis of the complete 226,000 document by 90,000 term DOE collection, but this was not used in the experiments reported below.)

Table 1 summarizes the number of terms and documents in the samples used for scaling as well as the total number of terms and documents in the databases.

| | SVD scaling | | | | | | |
|---|---|---|---|---|---|---|---|
| collection | sampled docs | terms | added docs | total docs | total terms | ndim | database (meg)[2,3] |
| DOE1 | 50000 | 42221 | 176087 | 226087 | 42221 | 250 | 262 |
| WSJ1 | 49555 | 70019 | 49556 | 99111 | 70019 | 250 | 169 |
| AP1 | 42465 | 78167 | 42465 | 84930 | 78167 | 250 | 163 |
| ZIFF1 | 37590 | 60565 | 37590 | 75180 | 60565 | 250 | 135 |
| FR1 | 26207 | 54713 | 0 | 26207 | 54713 | 250 | 80 |
| WSJ2 | 50000 | 76080 | 24520 | 74520 | 76080 | 235 | 141 |
| AP2 | 50000 | 82997 | 29923 | 79923 | 82997 | 235 | 153 |
| ZIFF2 | 56920 | 72197 | 0 | 56920 | 72197 | 235 | 121 |
| FR2 | 20108 | 48728 | 0 | 20108 | 48728 | 235 | 64 |
| totals | 382845 | 585687 | 360141 | 742986 | 585687[1] | | |

**Table 1.** Summary of 9 subcollections

NOTES

1. In the union of the 9 subcollections, there were 585687 word **tokens**, and 200785 word **types.**
2. In general, database size will be: (ndocs+nterms)*ndim*4
3. The total combined database size was 1288 meg (750000 docs and 585000 terms). If a single combined database had been used, the total database size would have been smaller

because many terms are now represented in more than one database. There were only about 200000 unique terms, so a combined database would have been about 930 meg. The fact that the number of terms does not grow linearly with the number of documents will help make large SVD calculations possible.

### *2.1.2 SVD*

The SVD program takes the log*entropy transformed term-document matrix as input, and calculates best "reduced-dimension" approximation to this matrix. The result of the SVD analysis is a reduced-dimension vector for each term and each document, and a vector of the singular values. For TREC, we computed a separate SVD for each of the 9 subcollection. The number of dimensions, $k$, ranged from 235-310.

### *2.1.3 Adding new documents and terms*

As noted above, the initial indexing and SVD were typically performed on a random sample of documents from each subcollection. The documents not included in the sample were "folded in" to the database. These documents were located at the weighted vector sum of their constituent terms. That is, the vector for a new document was computed using the term vectors for all terms in the document. These term vectors were combined using the appropriate term weights, and the singular values to differentially weight each dimension. (Details are given in Deerwester et al., 1990, p. 399.) For documents that are actually present in the term-document matrix, the derived vector corresponds exactly to the document vector given by the SVD. New terms can be added in an analogous fashion. The vector for new terms is computed using the document vectors of all documents in which the term appears. For the TREC experiments, only new documents, not terms, were added. The sizes of the complete databases (including all documents which were added) are summarized in Table 1.

When adding documents and terms in this manner, we assume that the derived "semantic space" is fixed and that new items can be fit into it. In general, this is not the same space that one would obtain if a new SVD were calculated using both the original and new documents. In previous experiments, we found that sampling and scaling 50% of the documents, and "folding in" the remaining documents resulted in performance that was indistinguishable from that observed when all documents were scaled.

## 2.2 Timing data

For the TREC experiments, all the pre-processing and retrieval was done on a Sparc2 with 384 meg of RAM. The SVD analyses were run on a Dec5000 with approximately 380 meg of RAM. Table 2 provides a summary of times (in minutes) to process documents and create the necessary data structures. It is important to note that these costs are incurred only *once* at the beginning. Subsequent query processing does not require any new SVD calculations or database updates.

### 2.2.1 *Pre-processing and indexing*

This includes the time for sampling documents (if necessary), processing the raw ascii text, creating the raw term-document matrix, calculating the log*entropy term weights (which requires two-passes), and transforming the matrix entries using a log*entropy weights. A combination of C-code, awk, and shell-scripts were used. The time required for this depends on the amount of raw text, the number of terms, and the number of documents.

### 2.2.2 *SVD*

The SVD program computes the best "$k$-dimensional" approximation to the transformed term-document matrix. We used a sparse, iterative Single-Vector Lanczos SVD code (Berry, 1992). The code is written in ANSI Fortran-77 using double precision arithmetic, and is available from Netlib. The number of singular values (dimensions) calculated for the TREC subcollections ranged from 235 to 310. As it turned out, we used only 235 or 250 dimensions for retrieval, so fewer dimensions could have been computed. Thus some of the reported SVD times are higher than necessary - in particular, the SVD times for AP1, ZIFF1, and FR1 would be approximately 20% lower if only 250 dimensions had been computed.

### 2.2.3 *Adding new documents*

The time required to add new documents includes the time to pre-process and index the text of the new documents as well as the time to compute the new document vectors.

### 2.2.4 *I/O translation*

Because several existing tools were patched together for the TREC experiments, there were some additional I/O translation involved. This will be removed soon.

| collection | index | SVD | adding new docs | I/O | TOTAL (mins) |
|---|---|---|---|---|---|
| DOE1 | 49 | 1219 | 591 | 194 | 2053 |
| WSJ1 | 241 | 1474 | 404 | 174 | 2293 |
| AP1 | 271 | 1644 | 455 | 214 | 2584 |
| ZIFF1 | 241 | 1359 | 352 | 156 | 2108 |
| FR1 | 241 | 939 | 0 | 133 | 1313 |
| WSJ2 | 427 | 1382 | 461 | 220 | 2490 |
| AP2 | 338 | 1210 | 273 | 218 | 2039 |
| ZIFF2 | 260 | 1452 | 0 | 208 | 1920 |
| FR2 | 187 | 486 | 0 | 105 | 778 |

**Table 2.** Summary of LSI/SVD times (in minutes)

## 2.3 Retrieval

### *2.3.1 Query processing*

Queries were automatically processed in the same way as documents. For queries derived from the topic statement, we began with the full text of each topic (all topic fields), and stripped out the SGML field identifiers. For feedback queries, we used the full text of relevant documents.

We **did not** use: stemming, phrases, syntactic or semantic parsing, word sense disambiguation, heuristic association, spelling checking or correction, proper noun identification, complex tokenizers, a controlled vocabulary, a thesaurus, or any manual indexing. Note also that we did **not** use any Boolean connectors or proximity operators in query formulation. The implicit connectives, as in ordinary vector methods, fall somewhere between ORs and ANDs, but with an additional kind of "fuzziness" introduced by the dimension-reduced association matrix representation of terms and documents.

### *2.3.2 Adhoc queries:*

The topic statements were automatically processed as described above to generate a list of query terms and their frequencies. This histogram of query terms was used to form a "query vector". A query vector was the weighted vector sum of its constituent term vectors. A separate query vector was created for matching against each of 9 databases (DOE1, WSJ1, AP1, FR1, ZIFF1, WSJ2, AP2, FR2, ZIFF2). For each subcollection, all query terms occurring in that database and their term weights were used. For example, a DOE1 query vector was created using the term weights and term vectors from the DOE1 database, and it was compared against the 226k documents in DOE1. This procedure was repeated for the remaining 8 collections, resulting in a total of 746k similarities between the adhoc queries and the 746k documents in the full TREC collection. (Note that we always started with the same query terms. However, these terms usually had somewhat different weights in the different collections, and some terms were not present in some subcollections.) We used 235 dimensions and a cosine similarity measure for all collections.

We submitted results from two sets of adhoc queries. The two sets of adhoc results differed only in how the information from the 9 subcollections was combined to arrive at a single ranking. In one case, we combined the information from the 9 databases by simply taking the raw cosines from the different collections and ranking them from largest to smallest. We call these the **adhoc_topic cosine** results. In another case, we normalized the cosines within each subcollection before combining. That is, within each subcollection, we transformed the cosines to z-scores so that they had a mean of 0 and a standard deviation of 1. We then combined across collections using these normalized z-scores rather than the raw cosines, again ranking from largest to smallest. We call these the **adhoc_topic normalized_cosine** results. This method of normalizing scores offers somewhat more flexibility in combining information from many subcollections. It means, for example, that different numbers of dimensions or similarity measures could be used in the different subcollections, but combined on the basis of a comparable score. In the TREC experiments, all comparisons used the same number of dimensions, so this normalization will equate for real differences in the data rather than statistical artifacts of the analysis.

*2.3.3 Routing queries:*

For the routing queries, we created a filter or profile for each of the 50 training topics. We submitted results from two sets of routing queries. In one case, the filter was based on just the topic statements - i.e., we treated the routing queries as if they were adhoc queries. The filter was located at the vector sum of the terms in the topic. We call these the **routing_topic cosine** results. In the other case, we used feedback about relevant documents from the training set. The filter in this case was derived by taking the vector sum of all relevant documents. We call these the **routing_reldocs cosine** results. This was an atypical variant of relevance feedback in that we replaced (rather than combined) the original topic with relevant documents. These two extremes provide baselines against which to compare methods for combining information from the original query and feedback about relevant documents.

In both cases, the filter was a single vector. New documents were matched against the filter vector and ranked in decreasing order of similarity. We have previously conducted experiments which suggest that performance can be improved if the filter is represented as several separate vectors. We did not use this method for the TREC results we submitted, but would like to do so in subsequent experiments. (See also Kane-Esrig et al., 1991 or Foltz and Dumais, 1992, for a discussion of multi-point interest profiles in LSI.)

For each topic, a separate filter vector was created in 4 training databases (WSJ1, AP1, FR1, ZIFF1). Thus each of the 50 filters was represented in 4 different databases. New documents (those from WSJ2, AP2, FR2, ZIFF2) were automatically pre-processed and indexed as described above. New documents were "folded in" to the comparable training database and compared to the filter vector in that database. For example, a new document from WSJ2 was added to the WSJ1 database and compared to the 50 filters in that database. In is important to note that only term vectors and term weights from the training subcollections were used in indexing and comparing these new documents to the routing filters.

We used 250 dimensions and a cosine similarity measure for all routing comparisons. Results from the different training subcollections were combined using raw cosine values.

*2.3.4 Matching/retrieval times*

Both queries (or filters) and documents were represented as reduced-dimension vectors. The cosine between each query and every document was computed, and documents were ranked in decreasing order of similarity to the query. The cosines were computer using 235 dimensions for the adhoc queries and 250 dimensions for the routing queries. Although there are many fewer dimensions than in standard vector retrieval, the entries are almost all non-zero so inverted indices are not useful. This means that each query must be compared to every document.

For 250-dimensional vectors, about 50,000 cosines can be computed per minute on a Sparc2. This time includes both comparison time and ranking time, and assumes that all document vectors are pre-loaded into memory. For the adhoc queries, the time to compare a query to the 742,986 documents was about 12 minutes if all comparisons were sequential. It is straightforward to split this matching across several machines or to use parallel hardware. Preliminary experiments using a 16,000 PE MasPar showed that 50,000 cosines could be

computed and sorted in 1 second. For the routing queries, the time to compare a new document to the filters (50 filters in each of 4 databases) was about .2 sec on a Sparc2 even when the term and filter vectors were read from disk.

## 3. Improving performance

### 3.1 Time

The LSI/SVD system was built as a research prototype to investigate many different information retrieval and interface issues. Retrieval efficiency was not a central concern because we first wanted to assess whether the method worked before worrying about efficiency, and because the initial applications of LSI involved much smaller databases of a few thousand documents. Almost no effort went into re-designing the tools to work efficiently for the large TREC databases. There are some obvious ways to decrease indexing, SVD, and retrieval time, and we discuss some of them below.

#### *3.1.1 Pre-processing, SVD, and database creation*

About 10% of the time is spent in unnecessary I/O translation (e.g., transposing large matrices), and this will be eliminated soon.

About 60% - 70% of the time is spent in the SVD. SVD algorithms get faster all the time. The sparse, iterative algorithm we now use is about 100 times faster than the method we used initially. There are the usual speed-memory tradeoffs in the SVD algorithms, so time can probably be decreased some by using a different algorithm and more memory. Parallel algorithms will help a little, but probably only by a factor of 2. Finally, all calculations are now done in double precision, so both time and memory could be decreased by using single precision. Preliminary experiments with smaller IR test collections suggest that this decrease in precision will not lead to numerical problems for the SVD algorithm.

It is important to note that the pre-processing and SVD analyses are one-time-only costs for relatively stable domains. We have also found that at least as many new items can be added to an existing SVD database without redoing the SVD scaling or seriously diminishing retrieval effectiveness.

#### *3.1.2 Query construction and retrieval*

Some calculations (e.g., scalings of various sorts) were done on the fly but could easily be precomputed. In addition, all calculations were done in floating point, and could be speeded up using integer arithmetic. By dropping all very low vector values, sparse vectors could be constructed to take advantage of efficient methods currently used by vector methods.

Query vectors were compared to *every* document. Document clustering could be used to reduce the number of comparisons. Query matching can also be improved tremendously by simply using more than one machine or parallel hardware. Parallel query matching produces much larger gains than any of the modifications discussed above. Using a 16,000 PE MasPar, with no

attempt to optimize the data storage or sorting, we decreased the time required to match a 250-dimensional query vector against all document vectors and sort by a factor of 60 to 100.

### 3.2 Accuracy

This section on accuracy is divided into two main parts - one examining the results of the two Adhoc and two Routing runs we submitted, and the other looking in detail at some failures of the LSI system.

Compared to other systems, LSI performance was average on the adhoc topics and somewhat below average on the routing topics (tho see the section on misses for a discussion of sizable improvements). Because there were so many differences between systems (tokenization, query construction, representation, matching, amount of human effort, etc.) it is difficult to isolate performance differences to specific, theoretically interesting components. For this reason, we focus on our own experimental results and failure analyses as a first step in understanding and improving performance.

#### *3.2.1 LSI experiments*

*3.2.1.1 Adhoc - normalization experiment.* We submitted results from two sets of adhoc queries. The two sets of adhoc query results differed only in how the similarities (cosines) from the 9 subcollections were combined to arrive at a single ranking. In one case, **adhoc_topic cosine**, we simply used the raw cosines from the different collections. In the other case, **adhoc_topic normalized_cosine**, we normalized the cosines within each subcollection before combining.

The differences in accuracy between these two methods were not very large. The raw cosine method of combining was about 15% better overall that the normalized cosine method (2786 vs. 2469 relevant articles, and .1274 vs. .1100 11-pt precision). In general, some form of normalization is needed to correct for measurement artifacts (e.g., lower mean cosine in higher dimensions) when combining scores from many different subcollections. Since we used the same number of dimensions for comparisons in each subcollection this correction was unnecessary in the present experiments. Normalization appeared to have some undesirable consequences for a few topics. Because normalization subtracts the mean cosine and divides by the variance, the same raw cosine will have a higher normalized score if it comes from a subcollection with a low mean cosine and low variance - but these are precisely the subcollections that we probably want to avoid!

*3.2.1.2 Routing - feedback experiment.* For the routing queries, we created two filters or queries for each of the 50 training topics. In one case, **routing_topic cosine**, the routing query was based on just terms in the topic statements, as if it had been an adhoc query. In the other case, **routing_reldocs cosine**, we used feedback about relevant documents from the training set and located the filter at the vector sum of the relevant documents. Our intent is to use these two runs as baselines against which alternative methods for combining the original query and relevant documents can be compared.

Somewhat surprisingly, the query using just the topic terms was about 25% more accurate than the feedback query (2234 vs. 1837 relevant articles; .1235 vs. .0972 11-pt precision). We suspect that part of the problem was attributable to the small number and inaccuracy of relevance judgements in the initial training set. This had substantial impact on performance for some topics because our feedback queries were based only on the relevant articles (and ignored the original topic description). For Topic 050, for example, there was only one relevant articles, and it did not appear to us to be relevant to the topic - the Topic was about "potential military interest in virtual reality", and the so-called relevant article was about "Denmark's crisis on nuclear power threatening its membership in NATO". Not surprisingly, using only this article as a query, no relevant articles about virtual reality were returned. Now that we have a larger number of hopefully more accurate relevance judgements, we will repeat this basic comparison.

We will then use these two baseline runs to explore: a) combining the relevant documents and the original topic; b) selecting only some relevant documents and/or discriminating terms; and c) representing the query vector as several points of interest rather than a single average.

### *3.2.2 Failure analyses*

In order to better understand retrieval performance we examined two kinds of retrieval failures: false alarms, and misses. *False alarms* are documents that LSI ranks highly that are judged to be irrelevant. *Misses* are relevant documents that are not in the top 200 returned by LSI. The observations presented below are based on preliminary analyses of some topics on which LSI performed poorly. Although we suggest methods for improving performance, most have not been tested systematically on the entire TREC collection, although we plan to do so.

*3.2.2.1 False Alarms.* The most common reason for false alarms (accounting for approximately 50% of those we examined) was lack of specificity. These highly ranked but irrelevant articles were generally about the topic of interest but did not meet some of the restrictions described in the topic. Many topics required this kind of detailed processing or fact-finding that the LSI system was not designed to address. Precision of LSI matching can be increased by many of the standard techniques - proper noun identification, use of syntactic or statistically-derived phrases, or a two-pass approach involving a standard initial global matching followed by a more detailed analysis of the top few thousand documents. Salton and Buckley (SMART's global and local matching), Evans (CLARIT's evoke and discriminate strategy), Nelson (ConQuest's global match followed by the use of locality of information), and Jakobs, Krupka and Rau (GE's pre-filter followed by a variety of more stringent tests) all used two-pass approaches to good advantage in the TREC tests. We intend to try some of these methods for TREC-2, and will focus on general-purpose, completely automatic methods that do not have to be modified for each new domain or query restriction.

Another common cause of false alarms appears to be the result of inappropriate query pre-processing. The use of negation is the best example of this problem. About 20% of the TREC topics contained explicit negations. LSI included negated words in the query along with all the other words. Topic 094, about computer-aided crime, also stated that articles that simply mentioned the spread of a computer virus or worm were NOT relevant. The first 20 documents that LSI returned were all about computer viruses! Another example of inappropriate query

processing involved the use of logical connectives. LSI does not handle Boolean combinations of words, and sometimes returned articles covering only a subset of ANDed topics.

Finally, it is not at all clear why about 20% of the false alarms were returned by LSI. Since LSI uses a statistically-derived "semantic" space and not surface-level word overlap for matching queries to documents, it is sometimes difficult to understand why a particular document was returned. One advantage of the LSI method is that documents can match queries even when they have no words in common; but this can also produce some spurious hits. Topic 066, about natural language processing technology, returned several articles about chip processing technologies and high technology products in general. Another reason for false alarms could be inappropriate word sense disambiguation. LSI queries were located at the weighted vector sum of the words, so words were "disambiguated" to some extent by the other query words. Similarly, the initial SVD analysis used the context of other words in articles to determine the location for each word in the LSI space. However, since each word has only one location, it sometimes appears as if it is "in the middle of nowhere". A related possibility concerns long articles. Lengthy articles which talk about many distinct subtopics were averaged into a single document vector, and this can sometimes produce spurious matches. Breaking larger documents into smaller subsections and matching on these might help.

*3.2.2.2 Misses.* For this analysis we examined a random subset of relevant articles that were not in the *top 200* returned by LSI. Many of the relevant articles were fairly highly ranked by LSI, but there were also some notable failures that would be seen only by the most persistent readers. So far, we have not systematically distinguished between misses that "almost made it" and those that were much further down the list.

About 40% of the misses we examined, represent articles that were primarily about a different topic than the query, but contained a small section that was relevant to the query. Because documents are located at the average of their terms in LSI space, they will generally be near the dominant theme, and this is a desirable feature of the LSI representation. Some kind of local matching should help in identifying less central themes in documents.

Another 40% of the misses appear to be the result of inappropriate selection of subcollections. Recall that we analyzed 9 subcollections separately and combined the similarities later to arrive at a single ranked list. The different subcollections sometimes had different densities of documents on some topics. This is most evident when considering computer-related topics. For general collections like AP or WSJ, relatively few of the articles were about computers, and we suspect that few of the 235-250 dimensions in the LSI semantic space were devoted to distinguishing among such documents. Thus, the similarities of these documents to queries about computers were relatively high and undifferentiated. For the ZIFF collections, on the other hand, most of the LSI dimensions were used to represent differences among computer concepts. Similarities of the top few hundred articles to computer queries were *lower* on average, but much finer distinctions among subtopics were possible. One consequence of this was that, when combining across collections, few articles from the ZIFF subcollections were included for queries about computer-related topics! Different term weights in the different subcollections also contributed to this problem.

Inappropriate subcollection selection accounts for many of LSI's failures on computer-related topics. Consider, for example, topic 037 about IBM's SAA standards. LSI performs very poorly compared to other systems on this topic, returning the fewest relevant articles (19) and having the lowest 11-pt precision (.0088). Summing over all systems for this topic, 68% of the total number of returned articles (554/812) and 99% of the relevant articles (444/449) were from ZIFF2. For LSI, however, only 17 of the top200 articles (8%) were from ZIFF2; all 17 of these articles were relevant. When a comparable proportion of LSI documents were selected from ZIFF2, the number relevant increased to 175 and the 11-pt precision increased to .3532. This performance places LSI slightly above the median for topic 037. We performed the same analysis for all topics in which more than 33% of the total returned articles were from ZIFF. There were 26 such topics (19 Adhoc Topics from 026-050, and 7 Routing Topics). For these topics, the mean percent of ZIFF articles chosen by all systems was 59%, compared with 9% for LSI. When comparable proportions of ZIFF articles were selected for LSI, the average number of relevant documents increased from 26 to 58, and the 11-pt precision increased from .0554 to .1111. A total of 700 new relevant documents were found for the 19 Adhoc Topics, and 125 new relevant documents were found for the 7 Routing Topics. Performance improvements were observed for 23 of the 26 topics - two of those in which there were no improvements were about AT&T products where poor pre-processing omitted AT&T from the LSI query (see below).

While these results are encouraging, the problem of how to select appropriate subcollections is not solved. For the routing topics, we could use training data to set some apriori mixture of articles from various subcollections. This strategy is not, however, generally applicable for adhoc queries. We will examine some more appropriate way of combining across subcollections to take distributional effects like this into account. Alternatively, we could use randomly selected documents (rather than topically organized ones) to create the subcollections. Finally, we could use a single large combined scaling in which there would be no need to combine across subcollections.

Finally, some misses were attributable to poor text (and query) pre-processing and tokenization. Since we do not keep single letters or use a database of company and location names, several important acronyms like U.S. and AT&T disappeared completely from both articles and queries! Not surprisingly, this resulted in many missed documents. We noticed that many of the top performing automatic systems used SMART's pre-processing, and we hope to do so as well for TREC-2. This will allow us to better understand the usefulness of LSI per se without many of the additional confounds introduced with different indexing.

### 3.3 Open experimental issues

The results of the failure analyses suggest several directions to pursue for TREC-2, including: improving pre-processing and tokenization; exploring some precision-enhancing methods; and developing methods for more effectively combining across subcollections. We also hope to explore three additional.

### *3.3.1 Separate vs. combined scaling*

We used 9 separate subscalings for the TREC experiments. Our decision to use the 9 subcollections was largely a pragmatic one initially. Would like to create a single large scaling and compare the results with those obtained using the subcollections.

### *3.3.2 Centroid query vs. many separate points of interest*

A single vector was used to represent each query. In some cases the vector was the average of terms in the topic statement, and in other cases the vector was the average of previously identified relevant documents. A single query vector can be inappropriate if interests are multifacted and these facets are not near each other in the LSI space. In the case of the routing queries, for example, we could match new documents against each of the previously identified relevant documents separately rather than against their average. We have developed techniques that allow us to match using a controllable compromise between averaged and separate vectors (Kane-Esrig et al., 1991). We did not use this method for TREC, but would like to do so for TREC-2.

### *3.3.3 Interactive interfaces*

All LSI evaluations were conducted using a non-interactive system in essentially batch mode. It is well known that one can have the same underlying retrieval and matching engine, but achieve very different retrieval success using different interfaces. We would like to examine the performance of real users with interactive interfaces. A number of interface features could be used to help users make faster (and perhaps more accurate) relevance judgements, or to help them explicitly reformulate queries. (See Dumais and Schmitt, 1991, for some preliminary results on query reformulation and relevance feedback.) Another interesting possibility involves returning something richer than a rank-ordered list of documents to users. For example, a clustering and graphical display of the top-k documents might be quite useful. We have done some preliminary experiments using clustered return sets, and would like to extend them to the TREC-2 collection. The general idea is to provide people with useful interactive tools that let them make good use of their knowledge and skills, rather than attempting to build all the smarts into the database representation or matching components.

## 4. Onward to TREC-2

We were quite pleased that we were able to use many of the existing LSI/SVD tools on the TREC collection. The most important finding in this regard was that the large, sparse SVD problems could be computed without numerical or convergence problems. The computations were not fast, but a day of CPU on existing workstations (for each subcollection) is certainly feasible, especially given that these calculations are only done once at the beginning to create the database. We have already computed some larger SVDs (100k x 200k) and would like to take advantage of this for TREC-2.

In terms of accuracy, LSI performance was reasonable. There are some obvious ways to improve

the initial pre-processing and indexing and we will do so. We would like to use indexing methods that are as similar as possible to other automatic vector methods, so that we can examine the contribution of LSI per se. We also now have many of the basic tools in place and should be able to conduct more experiments comparing various indexing and query matching ideas using the same underlying LSI engine.

LSI was designed as a method to increase recall, especially for the short queries that users typical generate. For the TREC application we would like to explore some precision enhancing tools as well. Some of these will probably consist of more refined matching algorithms, but we also hope to move in the direction of interactive interfaces. We see this as an effective way of combining human and machine intelligence.

## 5. References


[1] Berry, M. W. Large scale singular value computations. *International Journal of Supercomputer Applications*, 1992, *6(1)*, 13-49.

[2] Cullum, J.K. and Willoughby, R.A. *Lanczos algorithms for large symmetric eigenvalue computations - Vol 1 Theory*, (Chapter 5: Real rectangular matrices). Brikhauser, Boston, 1985.

[3] Deerwester, S., Dumais, S. T., Landauer, T. K., Furnas, G. W. and Harshman, R. A. Indexing by latent semantic analysis. *Journal of the Society for Information Science*, 1990, *41(6)*, 391-407.

[4] Dumais, S. T. Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments and Computers*, 1991, *23(2)*, 229-236.

[5] Dumais, S. T. and Schmitt, D. G. Iterative searching in an online database. In *Proceedings of Human Factors Society 35th Annual Meeting*, 1991, 398-402.

[6] Foltz, P. W. and Dumais, S. T. Personalized information delivery: An analysis of information filtering methods. *Communications of the ACM*, Dec. 1992, *35(12)*, 51-60.

[7] Furnas, G. W., Deerwester, S., Dumais, S. T., Landauer, T. K., Harshman, R. A., Streeter, L. A., and Lochbaum, K. E. Information retrieval using a singular value decomposition model of latent semantic structure. In *Proceedings of SIGIR*, 1988, 465-480.

[8] Kane-Esrig, Y., Streeter, L., Dumais, S. T., Keese, W. and Casella, G. The relevance density method for multi-topic queries in information retrieval. In *Proceedings of the 23rd Symposium on the Interface*, E. Keramidas (Ed.), 1991, 407-410.

[9] Salton, G. and McGill, M.J. *Introduction to Modern Information Retrieval.* McGraw-Hill, 1983.

## Appendix

Latent Semantic Indexing (LSI) uses singular-value decomposition (SVD), a technique closely related to eigenvector decomposition and factor analysis (Cullum and Willoughby, 1985). We take a large term-document matrix and decompose it into a set of $k$, typically 100 to 300, orthogonal factors from which the original matrix can be approximated by linear combination.

More formally, any rectangular matrix, $X$, for example a $t \times d$ matrix of terms and documents, can be decomposed into the product of three other matrices:

$$\underset{t \times d}{X} = \underset{t \times r}{T_0} \cdot \underset{r \times r}{S_0} \cdot \underset{r \times d}{D_0'},$$

such that $T_0$ and $D_0$ have orthonormal columns, $S_0$ is diagonal, and $r$ is the rank of $X$. This is so-called *singular value decomposition* of $X$.

If only the $k$ largest singular values of $S_0$ are kept along with their corresponding columns in the $T_0$ and $D_0$ matrices, and the rest deleted (yielding matrices $S$, $T$ and $D$), the resulting matrix, $\hat{X}$, is the unique matrix of rank $k$ that is closest in the least squares sense to $X$:

$$\underset{t \times d}{X} \approx \underset{t \times d}{\hat{X}} = \underset{t \times k}{T} \cdot \underset{k \times k}{S} \cdot \underset{k \times d}{D'}.$$

The idea is that the $\hat{X}$ matrix, by containing only the first $k$ independent linear components of $X$, captures the major associational structure in the matrix and throws out noise. It is this reduced model, usually with $k \approx 100$, that we use to approximate the term to document association data in $X$. Since the number of dimensions in the reduced model ($k$) is much smaller than the number of unique terms ($t$), minor differences in terminology are ignored. In this reduced model, the closeness of documents is determined by the overall pattern of term usage, so documents can be near each other regardless of the precise words that are used to describe them, and their description depends on a kind of consensus of their term meanings, thus dampening the effects of polysemy. In particular, this means that documents which share no words with a user's query may still be near it if that is consistent with the major patterns of word usage. We use the term "semantic" indexing to describe our method because the reduced SVD representation captures the major associative relationships between terms and documents.

One can also interpret the analysis performed by SVD geometrically. The result of the SVD is a $k$-dimensional vector representing the location of each term and document in the $k$-dimensional representation. The location of term vectors reflects the correlations in their usage across documents. In this space the cosine or dot product between vectors corresponds to their estimated similarity. Since both term and document vectors are represented in the same space, similarities between any combination of terms and documents can be easily obtained. Retrieval proceeds by using the terms in a query to identify a point in the space, and all documents are then ranked by their similarity to the query. We make no attempt to interpret the underlying dimensions or factors, nor to rotate them to some intuitively meaningful orientation. The analysis does not require us to be able to describe the factors verbally but merely to be able to represent terms, documents and queries in a way that escapes the unreliability, ambiguity and redundancy of individual terms as descriptors.

Choosing the appropriate number of dimensions for the LSI representation is an open research question. Ideally, we want a value of $k$ that is large enough to fit all the real structure in the data, but small enough so that we do not also fit the sampling error or unimportant details. If too many dimensions are used, the method begins to approximate standard vector methods and loses its power to represent the similarity between words. If too few dimensions are used, there is not enough discrimination among similar words and documents. We typically find that performance improves as $k$ increases for a while, and then decreases (Dumais, 1991). That LSI typically works well with a relatively small (compared to the number of unique terms) number of dimensions shows that these dimensions are, in fact, capturing a major portion of the meaningful structure.

# Retrieval Experiments with a Large Collection using PIRCS

K.L. Kwok, L. Papadopoulos and Kathy Y.Y. Kwan

email:kklqc@cunyvm.bitnet
Computer Science Department, Queens College, CUNY
Flushing, NY 11367

## ABSTRACT

Our strategy to Information Retrieval and to the TREC experiments is based on techniques that have previously been demonstrated to work for small to medium size collections: 1) use of document components for retrieval and term weighting; 2) two-word phrases to achieve better precision and recall; 3) combination of retrieval methods, and 4) network implementation with learning capability to support feedback and query expansion. Evaluation shows that we return the best results for Category B in both ad hoc and routing retrievals, and our approach is comparable to the best methods used in Category A experiments. It appears techniques that work for small collections such as combining soft-boolean retrieval with probabilistic model, user relevance feedback, and feedback with query expansion also work for this large collection.

## 1. Introduction

Over the past years, we have built an experimental system for automated information retrieval (IR) called PIRCS, acronym for Probabilistic Indexing and Retrieval - Components - System. It provides storage and retrieval capability based on collection statistics using content terms as index terms for document representation. Its design is based on a network, and has flexibility in mind more than efficiency or performance, so that future and unforseen approaches to IR may be supported. As it turns out, when the TREC experiments were announced, we found that our system fits nicely with the requirements. Certain changes (mostly because of large file sizes) and a number of peripheral programs are needed, but the basic design remains intact. Because available RAM and disk space are limited, we can only handle Category B Wall Street Journal files (WSJ 500 MByte raw data); however, we forsee no problem running the system for Category A files (2 GByte raw data) if we have the appropriate hardware. In what follows we shall describe our strategies for IR and TREC in Section 2, system description in Section 3 and the Appendix, results and discussions in Section 4, and conclusions in Section 5.

## 2. Strategies for IR and TREC Experiments

Over the past twenty five or so years, a number of techniques have been known to work in IR for small to medium collections. These include: manually created thesauri and phrases to enhance recall and precision; term weighting that can account for importance of content and discrimination; user relevance feedback and feedback with query expansion; combining multiple retrieval methods, and using multiple document and query representations. Many other methods exist and have been experimented with, (such as clustering, natural language processing, various artificial intelligence techniques, etc.) but their effectiveness in general are still in question. Our strategy to IR and to the TREC experiments is based on the following that reflect on the previous workable techniques: 1) use of document components; 2) two-word phrases; 3) combination of retrieval methods, and 4) network implementation with learning. We shall discuss how and why each of these methodologies may help contribute to the effectiveness of our results.

### 2.1. Use of Document Components and Query Formulation

We view each source document not as monolithic, but as constituted of components. Document components are utilized in two ways: for constructing a more restricted context for term weighting, retrieval and feedback, and for defining initial weights to the content terms for representation. These are discussed in the following sub-sections.

#### 2.1.1 Sub-Documents as Document Components

A survey of the WSJ files shows that document lengths vary substantially, from a couple of lines to hundreds, with several thousand words. Moreover, many documents carry unrelated news stories, separated by three dashes '---' on an independent line. We believe that treating such documents as monolithic objects will have adverse impact on: a) precision, because they might lead to high probability that homographs would occur in a different sense and context from what one intends; b) term weighting, because unreliable estimates of the necessary probabilities for the index terms might result, and affecting retrieval; c) feedback and query expansion, because documents that are long and have mixed unrelated topics will make these processes imprecise; and d) system output effectiveness, because after retrieval, users still have to manipulate a large document to locate where the relevant passage is.

There may be some risks in using document breakups. Boolean AND's would not be satisfied if the two factors for an AND happen to be split up in separate sub-documents. Coordinate matching would get less term counting if one does not somehow combine the counts of each sub-documents for ranking purposes, assumming all match terms are topically relevant. List queries with term weights may or may not suffer: even though a sub-document would have less term match than a full document, shorter document lengths may lead to higher term weights depending on the weighting method used. Since all documents are treated in the same fashion, the sub-document weights will be affected to the same degree. If we allow for a substantial chunk of text, such as a few hundred words, a writer generally would have a chance to express what s/he intends fairly completely, either in this or in another sub-document. Since we are neither using Boolean retrieval nor coordinate matching, we believe using sub-documents with more uniform lengths outweigh the risks. Our experimental results seem to support our conjecture.

Therefore the first processing we did was to break each document into component units of approximately equal lengths. We creat a new component, (which we call a sub-document, with two more digits attached at the end of the original document ID, assumming a breakup of at most 100 sub-documents per document) whenever we recognize the story separation mark '---', or when we have a run of texts exceeding N words and ending on a paragraph boundary. A sub-document should not be too short lest it carries little content. Moreover, since each sub-document is an independent entry consuming space and time resources, we do not want to exceed a certain limit, which we arbitrary set as double the original number. After some experimentation, we found that a break at N=360 raw words satisfies our design goal. The original number of documents in WSJ is (first half plus second half) 98733+74486 = 173219; after breaking into components, we end up with 192935+156880 = 349815 sub-documents. This forms our database for subsequent processing. We would prefer to break documents based on more sophisticated strategies such as context, but we have not done so.

#### 2.1.2 Query Formulation

In PIRCS without soft-boolean, queries and documents are items of the same category, each containing a list of content terms. For a query, we obtain the list from the <title>, <desc>, <narr> and <con> paragraphs of the topic in a fully automatic fashion. These paragraphs also go through a filtering program that removes standard introductory phrases such as: 'To be relevant, a document (will | must | ..) (discuss

| cite | report | ..)', etc. If a capital 'NOT' is found in a sentence, the rest of the sentence including the 'NOT' is also removed, because it is difficult for list queries to handle negation. We understand that this does not completely solve the negation problem because some 'not's are not capitalized, and many negations are expressed by other means. The remaining words from the four paragraphs are then merged and processed against the collection dictionary to form a query representation. No breakup into sub-documents is done for topics.

We also manually form a boolean query for each topic for soft-boolean retrieval, thus providing both an alternative representation for queries as well as a different retrieval method. We essentially scan the same paragraphs as before. Sometimes we also consult the document frequency of a term to screen out high frequency terms, to arrive at a smaller expression. We might occasionally add to the boolean expression some new terms that are not in the original paragraphs. However, the way our evaluation program works is that these new terms are ignored because they are not part of the automatically formed query.

### 2.1.3 Initial Term Weighting based on Single Terms as Conceptual Components

After the previous processes, we apply the use of document components a second time. We regard each content term within a sub-document or query as an independent concept. This allows us to use the principle of document self-recovery to give initial weights to each term of an item, or to use the simpler Inverse Collection Term Frequency (ICTF) weighting [1,2]. Because the former requires experimentally adjusting some paramenters and we did not have sufficient relevance judgment information, we decided to use the simplier ICTF for our initial weighting of a term in an item (query or document) as follows:

$$w_{ik} = \ln [p/(1-p)] + \ln [(1-s_{ik})/s_{ik}]. \qquad (1)$$

$w_{ik}$ is the weight given to term k in item i; p = 1/50, a constant chosen based on previous experience; and $s_{ik} = (F_k - d_{ik})/(N_w - L_i)$ if item i is a document, and $s_{ik} = F_k/N_w$ if item i is a query. Here $F_k = \sum_i d_{ik}$ is the collection frequency of term k, $d_{ik}$ is the term frequency of term k in item i, $L_i = \sum_k d_{ik}$ is the length of item i, and $N_w = \sum_i L_i = \sum_k F_k$ is the total number of terms in the database. The fraction $(1-s_{ik})/s_{ik}$ inside the logarithm ln in Eqn.1 is approximately $N_w/F_k$, if $N_w >> F_k >> d_{ik}$, hence the nomenclature ICTF.

## 2.2. Two-word Phrases and Other Vocabulary Control

Document frequencies (of terms) of a few hundred are small compared with a few hundred thousand documents. Yet a few hundred high-ranked documents that are irrelevant would stretch a user's patience to great limits. We therefore believe that in large collection environments, precision enhancement tools are very important. Syntactic phrases, or statistically generated phrases within tight context would probably be useful as indexing terms. However, we do not have these tools yet for the experiments.

A look at the collection also shows that WSJ jargon contains many two-word phrases that are a combination of very common words. Examples are 'big three', 'buy down', 'drug money', 'go public', 'take over', etc. By themselves, many of the single terms would be screened out because they are either on the stopword list, or have high document frequencies. In combination however these two-word phrases are precise in meaning and would probably impact favorably on both precision and recall. We therefore spent a fair amount of manual effort to record these content-specific combinations in a two-word phrase file. During processing, whenever such two-word phrases occur in adjacent positions in a sentence, a new index term is created consisting of the combination, in addition to the single terms. Such two-word combinations are also common in other fields. Our file has 396 such pairs, containing also some from the computing field, and not necessarily just consisting of function or common words. We would like to have a larger set, but we did not have the resource nor the domain expertise. Another vocabulary control

tool is a stopword list with 595 entries, some of which are morphological variants of the same word. In addition, we use Porter's stemming [3] algorithm for suffix stripping.

There are many other vocabulary control tools such as spell-checker, proper noun and date identification, synonym list, thesaurus, semantic nets, etc. These could be added in the future.

### 2.3. Multiple Retrieval Methods

It is well known that different retrieval algorithms on the same collection would lead to different retrieval results and that combining them could substantially enhance effectiveness [2,4,5,6]. Our network approach to IR (Section 4) can support multiple retrieval methods fairly easily. Three methods of retrieval are used in PIRCS for these experiments, and they agree nicely with the different types of TREC query requirements:

(a) Query-Focused Retrieval

This method involves the question: given a query $q_a$, should document $d_i$ be considered as relevant to it? This point of view is appropriate for ad hoc environment where we have a set of static documents being processed against a query stream. Each query serves as a focus to which documents are ranked. An approximate probabilistic measure $W_i$ of this answer for $d_i$ is evaluated as follows, and is used as the RSV (retrieval status value) to rank the whole collection with respect to $q_a$:

$$W_i = \sum_k w_{ak} * d_{ik}/L_i. \qquad (2)$$

$w_{ak}$ is the weight of term k in query $q_a$ as given in Eqn.1, and $d_{ik}/L_i$ measures the proportion that term k is used in $d_i$. The sum is over all terms k that overlap between $d_i$ and $q_a$.

(b) Document-Focused Retrieval

This method involves the reverse question: given document $d_i$, should query $q_a$ be considered as relevant to it? This point of view is appropriate for routing where we have a set of static queries being processed against a document stream. Each document serves as a focus to which queries can be ranked. However, evaluation of retrieval is usually done with respect to a specific query $q_a$, and so an approximate probabilistic measure $V_i$ of this answer is given to each document $d_i$ and used as RSV as follows:

$$V_i = \sum_k w_{ik} * d_{ak}/L_a. \qquad (3)$$

(c) Soft-Boolean Retrieval (Query-Focused)

This method depends on a boolean query being available for each topic, which we creat manually for this experiment. The idea is that topics that are represented as a list of terms (as in (a) and (b) above) have no structure. Boolean queries have structure but its logic is 'hard' and we lose the ability to attach term importance or rank the collection. The soft-boolean approach allows one to retain the query structure yet provides weights to both the terms and boolean operators, so that we can soften the logic and provide ranking capability. We have followed the extended Boolean approach of the vector model [7] for this implementation, but with simplified weights. For example, a query $q_a$ may be of the form:

$$q_a = B_b \wedge_p X, \quad \text{with} \quad X = C_c \vee_p D_d. \qquad (4)$$

B,C,D are index terms with weights b,c,d respectively, $\wedge$,$\vee$ are the boolean AND and OR each given a fixed weight p=2, and all clause weights are set to 1. If document $d_i$ also has these three terms with weights 0 <= b',c',d'<= 1, then the similarity between $d_i$ and $q_a$ could be evaluated recursively as:

$$x' = Sim(d_i,X) = sqrt[((cc')^2 + (dd')^2)/(c^2+d^2)],$$
$$Sim(d_i,q_a) = 1 - sqrt[ (b^2(1-b')^2+1*(1-x')^2 )/(b^2 +1)] \qquad (5)$$

All document and query term weights are taken from the edges of the net, so that the system is fully automatic once the boolean expression has been defined manually.

Our retrieval results for automatic query construction is then based on combining methods (a) and (b), thus: $W_i^{auto} = (W_i + V_i)/2$. Those for manual query construction is based on $W_i^{man} = r*W_i^{auto} + s*Sim(d_i,q_a)$. Both make use of combination of retrieval methods. The constants r, s are chosen as 0.65 and 0.35 respectively. The objective is that adding soft-boolean structure may enhance the retrieval results of the automatic method for the same queries. Our soft-boolean evaluation algorithm currently only accounts for terms that also appear in the network for this query; additional terms that may have been inserted manually are ignored.

## 2.4. Network Implementation with Learning

### 2.4.1 Network for Routing, Ad Hoc and Feedback without/with Query Expansion

The use of a network can provide a unified view of many retrieval algorithms and is a flexible tool for implementation. In PIRCS, retrieval methods (a) and (b) of the previous section are implemented as feedforward and feedbackwards processing in a Query-Term-Document (Q-T-D) network as presented in [8,9]. A binary tree representing a boolean expression can also be hung onto the net for method (c). These are shown in Figs.1,2.

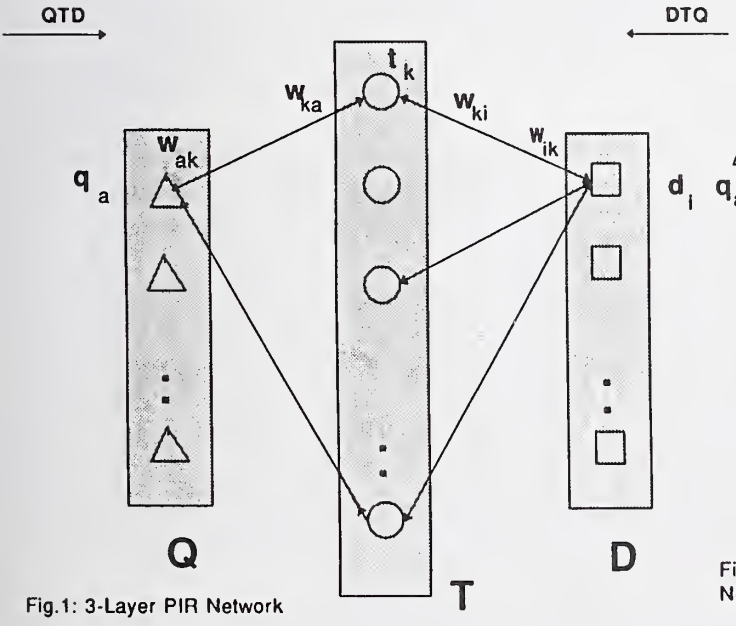


Fig.1: 3-Layer PIR Network

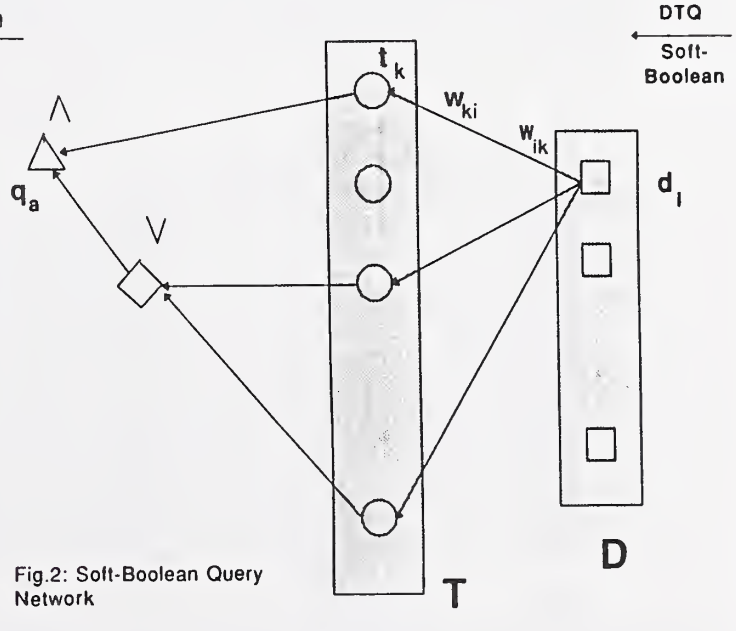


Fig.2: Soft-Boolean Query Network

The edges of the net are initialized as follows: $w_{ak}$ ($t_k$ acting on $q_a$) as in Eqn.1 and $w_{ki}$ ($d_i$ acting on $t_k$) = $d_{ik}/L_i$ as in Eqn.2, and similarly for $w_{ik}$ and $w_{ka}$. Activation on $d_i$=1 gated through $w_{ki}$ deposits on $t_k$

activation equal to the proportion that the term is used in $d_i$. These activated terms spread to a target $q_a$ gated through $w_{ak}$ which has the odds that given $t_k$ that $q_a$ is relevant. The sum of activations received at $q_a$ implements the query-focused retrieval method (a) of Section 2.3, while the reverse QTD direction processing implements document-focused retrieval method (b). Moreover, edge weights from the net can also be used to initialize the tree leaf nodes for soft-boolean evaluation in the DTQ direction, Fig.2.

Relevance feedback, which has been demonstrated in numerous experiments as the most effective tool for improving retrieval, is modeled as training in our net. For example, the edge weight $w_{ak}$ ($w_{ik}$) adapts according to a learning algorithm based on the average activation $x_k$ induced in term k by the relevant set, the current p factor (see Eqn.1) on the edge, and a learning rate $\eta_Q$ ($\eta_D$), thus:

DTQ: $\Delta w_{ak} = \Delta p_{ak}/[p_{ak}^{old}(1-p_{ak}^{old})]$, $\Delta p_{ak} = \eta_Q(x_k - p_{ak}^{old})$
QTD: $\Delta w_{ik} = \Delta p_{ik}/[p_{ik}^{old}(1-p_{ik}^{old})]$, $\Delta p_{ik} = \eta_D(x_k - p_{ik}^{old})$

see [8,9] for details and Fig.3 illustrating the DTQ process. Learning in both directions are allowed: DTQ query-focused training is done when we know the set of documents relevant to a query and corresponds to probabilistic retrieval [10,11], and QTD document-focused training is done when we know the set of queries relevant to a document and corresponds to probabilistic indexing [12]. Query-focused training let query representations improve with experience and prepare them to match new similar documents better, and normally associated with relevance feedback [13]. Document-focused training let document representations improve with experience and prepare them to match new similar queries better, and normally associated with dynamic document space modification [13]. They provide a precision enhancing tool because term weights are rendered 'sharper' towards relevant items. Our network implementation differs from the traditional approaches in that two sets of weights are associated with an item, e.g. $w_{ik}$ and $w_{ki}$. $w_{ki} = d_{ik}/L_i$ are the observed properties and not modified, while $w_{ik}$ embeds inference and adapts as more evidence is gathered. Moreover, effects of training from both query-focused and document-focused processes are combined into one RSV, $W_i^{auto}$, that has been shown to lead to cooperatively better ranking results. These ideas were first introduced in a series of papers [14,15,1,2]. With learning capability, the net becomes a two layer direct-connect artificial neural network in each direction.

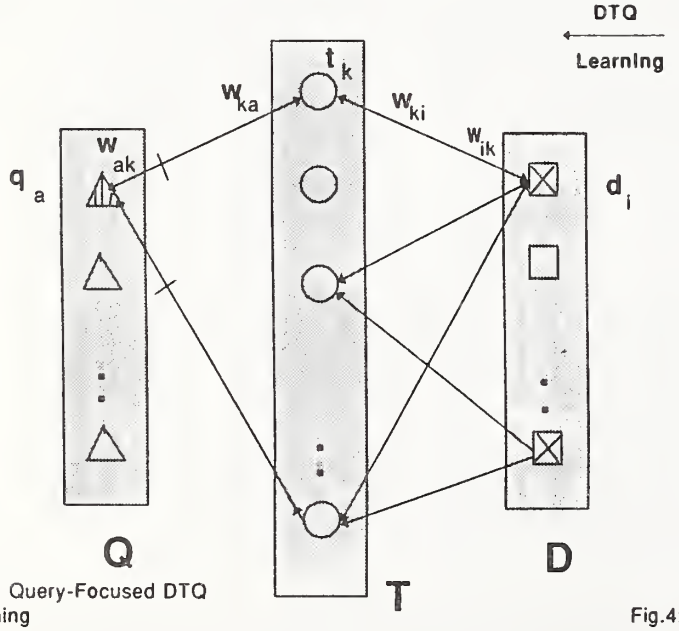


Fig.3: Query-Focused DTQ Learning

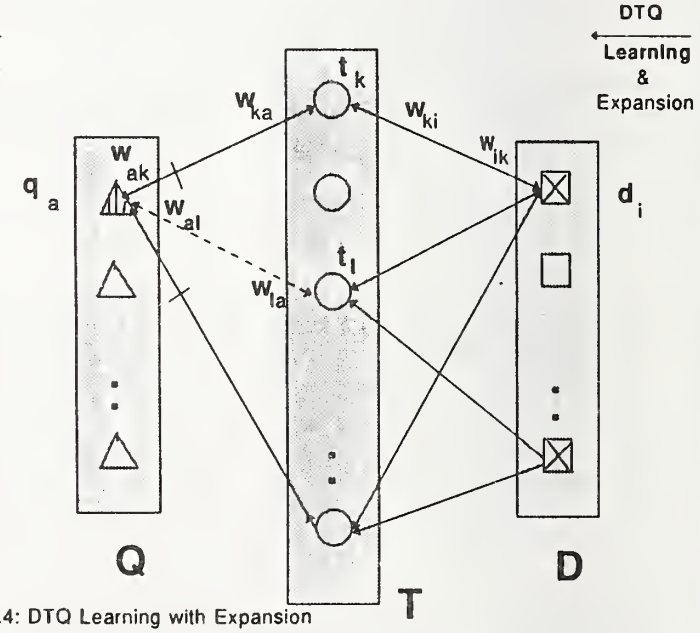


Fig.4: DTQ Learning with Expansion

An additional feature allowed in our feedback learning is query expansion. The idea is that terms that are

highly activated in the set of relevant documents should be highly related to the concepts and topics wanted for that query. These terms can therefore be added to the original query as illustrated in Fig.4, and could be expected to enhance both recall and precision, since these terms come from relevant samples. Experiments with small collections have shown that this is indeed the case [16,8,9]. This tool resembles that of an automatic thesaurus, associated terms being derived from user experience. From the network viewpoint, query expansion corresponds to growing new edges from a query to highly activated terms during relevance feedback, and weights are assigned according to the following learning algorithm:

DTQ: $w_{la} = \alpha * x_k$; $p_{al} = \beta * \eta_Q * x_k$, $w_{al} = \ln [p_{al}/(1-p_{al})] + \ln [N_w/F_k]$

which we introduced in [8,9]. In the TREC WSJ collection, we realize that documents can contain totally different stories and they can also be exceptionally lengthy, while both feedback and feedback with query expansion requires restricted context to work. This is a major reason why we decide to break documents into sub-documents, as discussed in Section 2.

### 2.4.2 Implementation of Retrieval in a Network

To satisfy TREC requirements, we need to simulate different querying circumstances as given in the following table:

| Query Construction Method | Query Type: ad hoc | Query Type: routing |
|---|---|---|
| automatic | PIRCS1 | PIRCS1 |
| manual | PIRCS2 | PIRCS2 |
| feedback | PIRCS3 | n.a. |
| fdbk + qry expansion | PIRCS4 | n.a. |

We have submitted results to all six types of experiments. The last Sections describe the rationale and how RSV's are calculated in our approach. The following discusses the processing of routing, ad hoc and feedback queries:

(a) Routing Queries: a special requirement in TREC is to do routing retrieval. This simulates the classification of new test documents with respect to a set of static queries that have been trained with past documents. The (past) training documents are the first half of the TREC collection c(A). We process c(A) and capture the necessary statistics of term usage. The queries from the first set of topics are then processed against c(A), and a network created using ICTF weighting. Query-focused DTQ learning is now applied with the supplied relevant (but not irrelevant) documents, and the resultant edge weights on the Q-T side of the net saved. Note that each relevant document is split into multiple sub-documents for this learning, and because we do not have relevance judgment at the sub-document level, we choose not to expand the queries. These become our routing queries q(A). The test documents from collection B, c(B), are then processed against c(A) as if they were queries, so that only collection A statistics are used for the ICTF term weighting. The q(A) are now loaded with c(B) and the dictionary from c(A) to form a new network, from which routing retrieval results based on $W_i^{auto}$ and $W_i^{man}$ (Section 2.3) are obtained, for fully automatic and manual routing queries.

(b) Ad Hoc Queries: collection B is now re-processed as additions to collection A, forming a total collection c(AB) and accumulating their total term usage statistics as well as a new dictionary. The ad hoc topics are processed to form ad hoc queries q(B) against the whole c(AB). We did not perform

document-focused QTD learning of c(AB) based on known relevant queries in q(A). The ad hoc queries q(B), the total dictionary, and the total collection c(AB) then form a network using ICTF weighting, from which both automatic and manual ad hoc query retrieval results ($W_i^{auto}$ and $W_i^{man}$) are obtained.

(c) Feedback Queries: To simulate user relevance feedback, we employ the ten highest ranked sub-documents of the automatic ad hoc results in (b), and determine their relevance to their respective topics manually. Two queries out of 25 have no relevant documents within the first ten, and they are removed leaving 23. We have performed two types of feedback training: without and with query expansion. In both cases, query-focused and document-focused training were done. Our method of adding terms to existing queries is based on [8,9] during DTQ direction training. The $n_a$ documents relevant to a query $q_a$ are instantiated to 1 and their activation to each term node, after gated through the weights $w_{ki}=d_{ik}/L_i$, is averaged. The highest 20 activated terms with document frequency less than 2000 are then used to expand $q_a$. However, not all the terms are different from the existing terms of $q_a$, so that on average, each query got expanded with about 12.3 terms (284 new terms for 23 queries). After this, training proceeds in the QTD direction. We have not used expansion for documents because usually there is only one relevant query per document. We have done only one round of feedback iteration.

## 3. System Description

Our system was designed over a number of years for IR experimentation. Its primary goal is to be flexible so that unforseen developments or algorithms can be supported for testing without much change in the basic implementation. Many intermediate files are produced for more convenient hooks to and from the system. These will be trimmed in later verisons. It was originally programmed in Pascal, but has been translated and is now running in a UNIX and C environment on a Sun SparcStation 2GS. The system has 48MB RAM, a TREC-dedicated 1.7GB disk drive, plus about 0.5GB on another drive that we can occasionally use. Our system characteristics and timings are described in the Appendix of this paper.

## 4. Results and Discussions

Our runs are named PIRCS'n', where n=1 denotes fully automatic retrieval, n=2 denotes manual, i.e. automatic combined with soft-boolean where boolean queries are manually created, n=3 denotes feedback using automatic retrieval, and n=4 is the same as n=3 but with query expansion. Although we use sub-documents for indexing and retrieval, our final 200-document output list has only unique document ID's; all sub-documents of the same document ID are suppressed except for the one with the highest rank. Precision-recall and other measures are gathered in the Appendix of this volume. Our runs are based on twenty five queries for ad hoc and another twenty five for routing retrievals, which we recognize as too few and may not reflect sufficiently the variety of query types in real-life. On the other hand, the evaluation curves and average values serve as a lower bound to the power of our system because relevance judgments are rendered only to the combined first 100 documents of all systems. This means documents ranked 101 to 200 in our retrieval are automatically considered irrelevant if they are not in the judged relevant set. Also, Category B texts come from a single source, viz. WSJ of multiple years, not like Category A with texts from multiple sources. With these reservations we like to make the following observations:

(a) Inter-site comparison shows that the PIRCS results are the best among Category B participants for both ad hoc and routing retrievals, and comparable to the best methods in Category A based on one evaluation with crnlB.

(b) Some of the topics have very specific requirements for documents to be relevant. For example: Topic #1 needs antitrust cases as a result of complaint, not routine review; #2 needs acquisitions between a U.S. company and another non-U.S. company; #53 needs leveraged buyout cases valued at or above $200 million; while #60 requires a policy change from merit-pay vs. seniority or vice versa. Data like 'above $200 million' or other numerics are removed either because they are on our stopword list or because of high frequency. Other topics involve very general concepts that require the system to understand their specific inferences. Examples are: course of action to decrease the U.S. deficit (#7); the body of water being polluted (#12); specific commercial applications of superconductors (#21); hypocritical and conflicting policies of the U.S. government (#74). The possible 'course of action', 'commercial applications', 'conflicting policies', etc. are essentially open-ended. Yet others need synonym lists or other aids to interpret proper terms in order not to miss documents. Examples are: Japanese, U.S. or foreign companies (#2,3), European Community or countries (#5, #69), third world or developing countries (#4,6), or economic indicators (#8). When would a proper noun, if identifiable, represents a company? And if it is, is it a foreign company? Also, a few of the topics like #3, 15, 53, 56, 66 have short descriptions with many general words, so that after stemming and stop-word processing, these queries end up with few terms. PIRCS does not have tools for these problems. Its precision values at 50% recall and at 11-pt Avg for ad hoc and routing retrievals are tabulated below:

| | Ad Hoc | | Routing | |
|---|---|---|---|---|
| | PIRCS1 | PIRCS2 | PIRCS1 | PIRCS2 |
| 50% Recall | .276 | .278 | .340 | .342 |
| 11-pt Avg | .311 | .322 | .343 | .369 |

The ad hoc precision value of about 0.28 at 50% recall says that, averaged over 25 queries, if one wants to retrieve half of all relevant documents, one would have to read about eleven documents to get three relevant, and about nine documents to get three relevants for routing. Routing queries receive some help from the few relevants provided for training purposes, just as in experiments first reported in [8] where we simulate users posing queries equiped with some known relevants. The 11-pt Avg precison values sample eleven recall points and simulate a uniform distribution of users with different recall needs, and may reflect actual usage better. Effectiveness improves to between three in ten and three in nine retrieved documents being relevant for ad hoc, and slightly better for routing. These results naturally leave much room for improvement; but considering that PIRCS1 is fully automatic and relies only on statistical methods, results seem reasonable for this large WSJ collection. See also analysis in (c) and (f).

(c) Another evaluation of the system is to look at the precision-recall values at different cut-off points of 5, 15, 30, 100 and 200 retrieved documents. This may give users a better 'feel' than the hypothetical 11-pt Avg. A question is what should these values be compared to? The theoretical limit is of course 1.0, for perfect recall and precision. However, this would punish the system unfairly. For example, at 15 retrieved documents, many queries have x relevants with $x > 15$. Hence the best recall at this cut-off should be 15/x. This we call the best operational recall in contrast to the theoretical best of 1.0. Similarly, if $x < 15$, these queries would have the best operational precision at this cut-off of x/15, instead of 1.0. We have listed below for PIRCS1 the precision-recall values at various cut-offs and also the best operational values for comparison:

| Cut-off: | 5 | 15 | 30 | 100 | 200 |
|---|---|---|---|---|---|
| Ad Hoc | | | | | |
| Best Oper. Recall: | .146 | .298 | .456 | .828 | .916 |
| Recall: | .066 | .130 | .204 | .419 | .586 |
| | 45% | 44% | 45% | 51% | 64% |

| | | | | | |
|---|---|---|---|---|---|
| Best Oper. Precision: | .984 | .944 | .899 | .689 | .469 |
| Precision: | .624 | .523 | .480 | .364 | .281 |
| | 63% | 55% | 53% | 53% | 60% |
| Routing | | | | | |
| Best Oper. Recall: | .083 | .241 | .387 | .777 | .915 |
| Recall: | .048 | .145 | .228 | .443 | .576 |
| | 58% | 60% | 59% | 57% | 63% |
| Best Oper. Precision: | 1.00 | .995 | .945 | .781 | .551 |
| Precision: | .608 | .597 | .521 | .398 | .294 |
| | 61% | 60% | 55% | 51% | 53% |

For ad hoc retrieval at 30 retrieved document cut-off for example, it means that on average we get about 20% (.204) of all the relevant documents, but nearly one out of two (.480) retrieved documents are relevant. However, because of the number of known relevant documents for each of the queries, the best operational recall and precision at this cut-off is only .456 and .899. Our .204 recall and .480 precision achieve 45% and 53% of these best values. On the whole we have achieved between 44-64% of the best operational recall and 53-63% of the best operational precision values for ad hoc, and respectively between 57-63% and 51-61% for routing. These are quite respectable figures.

(d) Based on our strategy and experimental data, techniques that work for small collections also work for this large WSJ collection as well. For example, using the 11-pt Avg precision values, PIRCS2 performs better than PIRCS1 in both ad hoc (0.322 vs 0.311, +3.6%) and routing (0.369 vs 0.343, +7.6%) environments. Thus, adding soft-boolean as a third retrieval method helps, but requires forming the boolean queries manually. An illustration of the better results of PIRCS2 over PIRCS1 can be seen by comparing between pairs of methods using the 11-pt Avg measure, where 'equal' means values within 5%:

| | AD HOC | ROUTING |
|---|---|---|
| | PIRCS1 @ PIRCS2 | PIRCS1 @ PIRCS2 |
| @ = better: | 5 | 3 |
| @ = equal: | 9 | 6 |
| @ = worse: | 11 | 16 |

Both PIRCS3 and PIRCS4 make use of feedback based on evaluating the first ten retrieved sub-documents of PIRCS1, but PIRCS4 employs query expansion as well. These methods are automatic. Comparing PIRCS4 with PIRCS3 in ad hoc feedback retrieval shows that query expansion is better (0.305 vs 0.282, +8.1%) than no expansion. Note that in this feedback, only 23 queries are used because topic #66 and #73 have no relevants in the first ten retrieved. PIRCS3 and PIRCS4 have also been re-evaluated using the frozen rank method, viz. the ten documents used for feedback are given the same rank as in PIRCS1, while the other retrieved documents follow. In addition, the two queries without relevants in first ten are put back carrying their PIRCS1 results. This way, PIRCS1,3f,4f can be directly comparable. It can be seen that feedback by itself works (0.3407 vs 0.3107, +9.7%) and query expansion improves further (0.3634 vs 0.3107, +17%). There are about 5.5 (126/23) relevant documents in the first ten retrieved.

(e) To illustrate the power of feedback, we have tabulated the number of queries that perform better, about equal, or worse between pairs of methods using the 11-pt Avg measure:

| | PIRCS1 @ PIRCS3f | PIRCS3 @ PIRCS4f |
|---|---|---|
| @ = better: | 1 | 5 |
| @ = equal: | 5 | 3 |
| @ = worse: | 19 | 17 |

PIRCS3f overwhelmingly improves over PIRCS1 19 to 1 with 5 ties. This is because PIRCS1 starts with low performance. PIRCS4f improves over PIRCS3f, but not with such wide margin, because PIRCS3f already achieves better results. It can been seen that feedback with or without query expansion within this collection is definitely worthwhile. Looking at the recall and precision at various retrieved document cut-off in the Appendix of this volume, it can be seen that PIRCS4f performs better than PIRCS3f, except at the high precision cut-ff of 5. This reflects that query expansion is more of a recall enhancement tool. This can also be seen from the number of relevants retrieved at the 200 document cut-off: 1403, 1526 and 1582 respectively for PIRCS1,3f,4f. At high precision region, the added terms could lead to more noise. The effect however is small, and may be related to the number and type of added terms. Thus, query expansion behaves like an automatic thesaurus, terms added being indirectly based on user relevance.

(f) It is tempting to compare the results of PIRCS in this large WSJ collection with those in small collections. We have listed the ad hoc 10-pt Avg precision of the four standard collections popular with IR research [2] below, with that of WSJ PIRCS1 results:

| MED | CRAN | CACM | WSJ | ISI |
|---|---|---|---|---|
| (30q,1.03Kd) | (225q,1.40Kd) | (52q,3.2Kd) | (25q,350Kd) | (76q,1.46Kd) |
| 0.472 | 0.374 | 0.297 | 0.263 | 0.174 |

MED (medicine) and CRAN (aerodynamics) use more specific, scientific vocabulary and can be expected to have better performance. CACM (computer science) terminology generally is less 'scientific' and can often have general vocabulary; its performance and that of WSJ are similar. ISI (information science) has very broad nonspecific queries, and the vocabulary is quite general and has the worst retrieval results. We were afraid WSJ would have low performance like ISI. Listed below is a comparison between the CACM [2] and WSJ PIRCS1 P-R curves:

| Recall: | 0.0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| WSJ: | .78 | .66 | .55 | .41 | .31 | .28 | .19 | .11 | .07 | .04 | .01 |
| CACM: | | .62 | .50 | .41 | .36 | .30 | .24 | .18 | .15 | .11 | .09 |

A characteristic of WSJ is that precision falls off to much smaller values at high recall region, compared with CACM or other small collections. One should expect a large collection to have more noise, and it is interesting that this noise impacts predominantly the low signal region. For example, WSJ has generality ratio of only 0.00077 versus 0.00478 for CACM, i.e. more than six times as much noise in WSJ. Another reason for this phenomenon, perhaps to a lesser degree, is that of the incomplete evaluation procedure for documents ranked between 101 to 200 as discussed earlier. At low recall region, however, precision of WSJ is comparable or better than the small collections. Why is that? Our interpretation is that first, queries are much richer and better formed in WSJ compared to those in CACM. Second, when a collection is large, there is a very good chance that a number of relevant documents exist using closely the same terms as the queries describing their content, especially if the queries are well-worded. These documents will rank high, and hence precision at low recall does not suffer in spite of adverse generality ratio. At high recall region, relevant documents do not express their content in similar terms to the queries and have few term matches, and interference from poor generality ratio noise magnifies. Techniques to improve precision at high recall would therefore be very important if one needs exhaustive search. It has been quite popular to criticise IR research in that small collection results do not reflect those of large collections. Our hope is that as experience is gained with more of these large scale collections, we might be able to predict their behavior based on small collection results. Small collection experiments can be performed in a matter of hours, while large collections take days using current technology. It should be noted that all databases considered here are from one homogeneous field of knowledge. Many commercial database providers produce CDROMs that are homogeneous and of about the same size as WSJ, and these

results may be relevant.

## 5. Conclusion

Our system called PIRCS, acronym for Probabilistic Indexing and Retrieval -Component- System, has been shown to be able to support storage and retrieval for a large collection of 0.5 GB. If appropriate hardware is available, and with some software modification, it should support 2GB size collections. Our strategy to IR and TREC consists of: 1) use of document components to provide a more restricted context for retrieval and feedback, and to provide an initial ICTF term weighting; 2) two-word phrases, especially consisting of stopwords and high frequency stems as a precision and recall enhancing tool; 3) combination of retrieval methods, including soft-boolean, to capture cooperative effects between retrieval methods, and 4) network implementation with learning to implement feedback and feedback with query expansion, resulting in a two-layer direct-connect artificial neural network with adaptive architecture. Our approach leads to results that are better than expected.

## 6. Acknowledgment

We would like to thank our department Chairman and the Dean of Mathematics and Natural Science for their support throughout the project. This work is partially supported by a grant from DARPA and a PSC-CUNY grant #6-63288.

## References

1. Kwok, K.L (1989). A neural network for probabilistic information retrieval. Proc. ACM SIGIR 12th Ann. Intl. Conf. on R&D in IR. N.J. Belkin & C.J. van Rijsbergen, eds. ACM: NY, pp.21-30.
2. Kwok, K.L (1990). Experiments with a component theory of probabilistic information retrieval based on single terms as document components. ACM TOIS 8:363-386.
3. Porter, M.F (1980). An algorithm fro suffix stripping. Program 14:130-137.
4. Smith, M (1990). Aspects of the p-norm model of information retrieval: Syntactic query generation, efficiency, and theoretical properties. TR 90-1128, Ph.D. Thesis, Cornell University.
5. Turtle, H.R & Croft, W.B (1991). Evaluation of an inference network-based retrieval model. ACM TOIS 9:187-222.
6. Fox, E.A; Nunn, G.L & Lee, W.C (1988). Coefficients for combining concept classes in a collection. Proc. ACM SIGIR 11th Ann. Intl. Conf. on R&D in IR. Y. Chiaramella, ed. PUG: Grenoble, pp.291-307.
7. Salton, G; Fox, E.A & Wu, H (1983). Extended boolean information retrieval. Comm. ACM 26:1022-1036.
8. Kwok, K.L (1991). Query modification and expansion in a network with adaptive architecture. Proc. ACM SIGIR 14th Ann. Intl. Conf. on R&D in IR. A. Bookstein, Y. Chiaramella, G. Salton & V.V. Raghavan eds. ACM: NY, pp.192-201.
9. Kwok, K.L (199x). A network approach to probabilistic information retrieval. submitted for publication.
10. Robertson, S.E & Sparck Jones, K (1976). Relevance weighting of search terms. J. ASIS. 27:129-146.
11. van Rijsbergen, C.J (1979). Information Retrieval, 2nd Ed. Butterworths: London.
12. Maron M.E & Kuhns, L.J (1960). On relevance, probabilistic indexing and information retrieval. J. ACM 7:216-244.
13. Salton, G (1989). Automatic Text Processing. Addison-Wesley: NY.

14. Kwok, K.L (1986). An interpretation of index term weighting schemes based on document components. Proc. 1986 ACM Conf. on R&D in IR. F. Rabitti, ed. ACM: NY, pp.275-283.
15. Kwok, K.L & Kuan, W (1988). Experiments with document components for indexing and retrieval. Inform. Proc. Mgmnt. 24:405-417.
16. Salton, G & Buckley, C (1990). Improving retrieval performance by relevance feedback. J. of ASIS. 41:288-297.

## Appendix

System Summary and Timing

I. Construction of indices, knowledge bases, and other data structures
(please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures

| | |
|---|---|
| 1. Stopword List | YES |
| a. how many words in list? | 595 |
| 2. is a controlled vocabulary used? | NO |
| 3. stemming | |
| a. standard stemming algorithms | YES |
| which ones? | PORTER'S ALGORITHM |
| b. morphological analysis | NO |
| 4. term weighting | YES |
| 5. phrase discovery | NO |
| a. what kind of phrase? | |
| b. using statistical methods | |
| c. using syntactic methods | |
| 6. syntactic parsing | NO |
| 7. word sense disambiguation | NO |
| 8. heuristic associations | NO |
| a. short definition of these associations | |
| 9. spelling checking (with manual correction) | NO |
| 10. spelling correction | NO |
| 11. proper noun identification algorithm | NO |
| 12. tokenizer (recognizes dates, phone numbers, common patterns) | NO |
| a. which patterns are tokenized? | |
| 13. are the manually-indexed terms used? | NO |

14. other techniques used to build data structures (brief description)
A TABLE OF 396 MANUALLY CREATED 2-WORD PHRASES. WHEN THESE ARE IDENTIFIED IN ADJACENT POSITIONS IN DOCUMENTS OR QUERIES THEY ARE USED AS ADDITIONAL INDEX TERMS.

B. Statistics on data structures built from TREC text
(please fill out each applicable section)

| | |
|---|---|
| 1. inverted index | |
| a. total amount of storage (megabytes) | 372 |
| b. total computer time to build (approximate number of hours) | 95+11+2=108 for 500MB. CLOCK TIME |
| c. Is the process completely automatic? | YES, IF SUFFICIENT DISK. NOT IN THIS EXPERIMENT. |

if not, approximately how many hours of manual labor?
0.5

d. are term positions within documents stored?
NO, BUT SENTENCE YES.

f. single terms only? YES, EXCEPT FOR I.A.14.

2. clusters NO
a. total amount of storage (megabytes)
b. total computer time to build (approximate number of hours)
c. brief description of clustering method
d. is the process completely automatic?
if not, approximately how many hours of manual labor?

3. ngrams, suffix arrays, signature files NO
a. total amount of storage (megabytes)
b. total computer time to build (approximate number of hours)
c. brief description of methods used
d. is the process completely automatic?
if not, approximately how many hours of manual labor?

4. knowledge bases NO
a. total amount of storage (megabytes)
b. total number of concepts represented
c. type of representation (frames, semantic nets, rules, etc.)
d. total computer time to build (approximate number of hours)
e. total manual time to build (approximate number of hours)
f. use of manual labor
(1) mostly manually built using special interface
(2) mostly machine built with manual correction
(3) initial core manually built to "bootstrap" for completely machine-built completion
(4) other (describe)
g. auxiliary files needed for machine use
(1) machine-readable dictionary (which one?)
(2) other (identify)

5. special routing structures (what?) SEE I.B.6
NETWORK NODE, EDGE FILES.
ROUTING USING NETWORK NODE AND EDGE FILES IS STRAIGHTFORWARD.

a. total amount of storage (megabytes)
NODE FILE: 4x7.5 EDGE FILE: 4x4
NETWORK SEGMENTED INTO 4, BECAUSE OF INSUFFICIENT RAM.

b. total computer time to build (approximate number of hours)
40+5+1+4x0.2=46.8, STARTING FROM TEXT FILE.

c. is the process completely automatic? YES, IF SUFFICIENT RAM AND DISK SPACE.

d. brief description of methods used
1. PROCESS (OLD) COLLECTION A.
2. PROCESS QUERIES AGAINST COLLECTION A.
3. PROCESS NEW COLLECTION B AS IF THEY WERE QUERIES TO MAKE USE OF COLLECTION A STATISTICS.
4. COMBINE QUERIES, (OLD) DICTIONARY AND COLLECTION B INTO NETWORK FOR RETRIEVAL.

6. other data structures built from TREC text (what?)

1. SUBDOCUMENT FILE
2. CODED FILE
3. DOCID CHECKING FILE
4. TERMID CHECKING FILE
5. DOCNUM FILE
6. TERMNUM (DICTIONARY) FILE
7. DIRECT FILE
8. INDEX TO DIRECT FILE
9. NODE FILE
10. EDGE FILE

a. total amount of storage (megabytes)

1. 481
2. 324
3. 7
4. 4
5. 11
6. 6
7. 372
8. 19
9. 4x14
10. 4x9

SYSTEM WAS DEVELOPED FOR EXPERIMENTAL RESEARCH, WITH FLEXIBILITY TO GENERATE OTHER DATA. SOME OF THE FILES ARE NOT NECESSARY FOR RETRIEVAL.

b. total computer time to build (approximate number of hours)

1. 1.5
2,3,4,5,6. 95
7,8. 11
9,10. 4x0.25=1

c. is the process completely automatic?

YES IF SUFFICIENT RAM AND DISK SPACE. FOR THIS EXPT, NO.

if not, approximately how many hours of manual labor?

2

d. brief description of methods used

RAW TEXT --> SUBDOCUMENT FILE
SUBDOCUMENT --> CODED FILE, DOCID FILE, TERMID FILE
DOCNUM FILE, TERMNUM (DICTIONARY) FILE.
ZIPF-LAW PROGRAM TRUNCATES DICTIONARY VIA USER ASSIGNED LIMITS.
CODED, TERMNUM --> DIRECT FILE with INDEX
DIRECT --> INVERTED FILE
DIRECT, INVERTED --> NODE, EDGE FILES.

C. Data built from sources other than the input text

1. internally-built auxiliary files

a. domain independent or domain specific (if two separate files, please fill out one set of questions for each file)

DOMAIN SPECIFIC

b. type of file (thesaurus, knowledge base, lexicon, etc.)

WORD PAIR

c. total amount of storage (megabytes)

0.005

d. total number of concepts represented

396

e. type of representation (frames, semantic nets, rules, etc.)

f. total computer time to build (approximate number of hours)

(1) if already built, how much time to modify for TREC?

0

(THIS IS A FILE CREATED VIA EDITOR).

g. total manual time to build (approximate number of hours)

16

(1) if already built, how much time to modify for TREC?

h. use of manual labor
(1) mostly manually built using special interface
(2) mostly machine built with manual correction
(3) initial core manually built to "bootstrap" for completely machine-built completion
(4) other (describe)

SEARCH FOR WSJ TERMINOLOGY IN LIBRARY AND FROM TOPICS.

2. externally-built auxiliary file — NONE
a. type of file (Treebank, WordNet, etc.)
b. total amount of storage (megabytes)
c. total number of concepts represented
d. type of representation (frames, semantic nets, rules, etc.)

II. Query construction
(please fill out a section for each query construction method used)

A. Automatically built queries (ad-hoc)
1. topic fields used

<TITLE>, <DESC>, <NARR>, <CON>

2. total computer time to build query (cpu seconds)

5 (AVERAGE FOR EACH QUERY).

3. which of the following were used?
a. term weighting with weights based on terms in topics

YES + OTHER WEIGHTS

b. phrase extraction from topics — NO
c. syntactic parsing of topics — NO
d. word sense disambiguation — NO
e. proper noun identification algorithm — NO
f. tokenizer (recognizes dates, phone numbers, common patterns)
(1) which patterns are tokenized? — NO
g. heuristic associations to add terms — NO
h. expansion of queries using previously-constructed data structure (from part I) — YES
(1) which structure? — WORD-PAIR PHRASE FILE
i. automatic addition of Boolean connectors or proximity operators

NO

j. other (describe) — NONE

B. Manually constructed queries (ad-hoc)
1. topic fields used

<TITLE>, <DESC>, <NARR>, <CON>

2. average time to build query (minutes)

300 minutes for 25 queries.

3. type of query builder
a. domain expert — NO
b. computer system expert — YES
4. tools used to build query
a. word frequency list — SOMETIMES
b. knowledge base browser (knowledge base described in part I)
(1) which structure from part I — NO
c. other lexical tools (identify) — NO
5. which of the following were used?
a. term weighting — YES
b. Boolean connectors (AND, OR, NOT) — YES

c. proximity operators — NO
d. addition of terms not included in topic — YES
(1) source of terms — WORD-PAIR PHRASE FILE.
e. other (describe)

C. Feedback (ad-hoc)
1. initial query built by method 1 or method 2? — METHOD 1
2. type of person doing feedback
a. domain expert — NO
b. system expert — YES
3. average time to do complete feedback
a. cpu time (total cpu seconds for all iterations)
12 PER QUERY PER ITERATION - NO EXPANSION.
50 " " " " - WITH EXPANSION.

b. clock time from initial construction of query to completion of final query (minutes)
60 PER QUERY TO DO RELEVANCE JUDGMENT.

3. average number of iterations — 1
a. average number of documents examined per iteration — 10
4. minimum number of iterations — 1
5. maximum number of iterations — 1
6. what determines the end of an iteration? — DEADLINE + LACK OF MANPOWER
6. feedback methods used
a. automatic term reweighting from relevant documents — YES
b. automatic query expansion from relevant documents
(1) all terms in relevant documents added — NO
(2) only top X terms added (what is X)
TOP 20 MOST 'ACTIVATED' TERMS THAT HAVE DOCUMENT FREQUENCY < 2000 WERE USED. BECAUSE MANY WERE ALREADY IN QUERY, ABOUT 12 ON THE AVERAGE WERE NEW AND ADDED PER QUERY.

(3) user selected terms added — NO
c. other automatic methods
brief description
FEEDBACK IS BASED ON SUB-DOCUMENTS.

d. manual methods — NO
(1) using individual judgment with no set algorithm
(2) following a given algorithm (brief description)

D. Automatically built queries (routing)
1. topic fields used — <TITLE>, <DESC>, <NARR>, <CON>
2. total computer time to build query (cpu seconds) — 5 (AVERAGE FOR EACH QUERY).
3. which of the following were used in building the query?
a. terms selected from
(1) topic — YES
(2) all training documents — NO
(3) only documents with relevance judgments — NO

b. term weighting
(1) with weights based on terms in topics YES
(2) with weights based on terms in all training documents
YES
(3) with weights based on terms from documents with relevance judgme
nts YES
c. phrase extraction NO
(1) from topics
(2) from all training documents
(3) from documents with relevance judgments
d. syntactic parsing NO
(1) of topics
(2) of all training documents
(3) of documents with relevance judgments
e. word sense disambiguation NO
(1) using topic
(2) using all training documents
(3) using documents with relevance judgments
f. proper noun identification algorithm NO
(1) from topics
(2) from all training documents
(3) from documents with relevance judgments
g. tokenizer (recognizes dates, phone numbers, common patterns)
NO
(1) which patterns are tokenized?
(2) from topics
(3) from all training documents
(4) from documents with relevance judgments
h. heuristic associations to add terms NO
(1) from topics
(2) from all training documents
(3) from documents with relevance judgments
i. expansion of queries using previously-constructed data structure
(from part I) YES
(1) which structure? WORD-PAIR PHRASE FILE
j. automatic addition of Boolean connectors or proximity operators
NO
(1) using information from the topics
(2) using information from the all training documents
(3) using information from documents with relevance judgments
k. other (brief description) NO

E. Manually constructed queries (routing)
1. topic fields used
<TITLE>, <DESC>, <NARR>, <CON>
2. average time to build query (minutes)
300 MINUTES FOR 25 QUERIES.
3. type of query builder
a. domain expert NO
b. system expert YES
4. data used for building query
a. from training topic YES
b. from all training documents NO
c. from documents with relevance judgments NO
d. from other sources (what?) NO

5. tools used to build query
   a. word frequency list — SOMETIMES
   b. knowledge base browser (knowledge base described in part I)
      (1) which structure from part I
   c. other lexical tools (identify)
   d. machine analysis of training documents
      (1) describe

5. which of the following were used?
   a. term weighting — YES
   b. Boolean connectors (AND, OR, NOT) — YES
   c. proximity operators — NO
   d. addition of terms not included in topic — NO
      (1) source of terms
   e. other (brief description) — NO

III. Searching

A. Total computer time to search (cpu seconds)

1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)
   18-30 PER QUERY, NO SOFT-BOOLEAN (COMBINE 2 METHODS).
   30-70 " " WITH " " (COMBINE 3 METHODS).

2. ranking time (total cpu seconds to sort document list)
   4 - 12 PER QUERY.

B. Which methods best describe your machine searching methods

1. vector space model — NO
2. probabilistic model — YES
3. cluster searching — NO
4. ngram matching — NO
5. Boolean matching — NO
6. fuzzy logic (include your definition) — YES, SOFT-BOOLEAN
7. free text scanning — NO
8. neural networks — YES
9. conceptual graph matching — NO
10. other (describe) — NONE

B. What factors are included in your ranking?

1. term frequency — YES
2. inverse document frequency — NO
3. other term weights (where do they come from?)
   INVERSE COLLECTION TERM FREQUENCY
   TOTAL WORD OCCURRENCES.
4. semantic closeness (as in semantic net distance) — NO
5. position in document — NO
6. syntactic clues (state how) — NO
7. proximity of terms — NO
8. information theoretic weights — NO
9. document length — YES
10. completeness (what % of the query terms are present) — NO
11. ngram frequency — NO
12. word specificity (i.e., animal vs. dog vs. poodle) — NO
13. word sense frequency — NO
14. cluster distance — NO
15. other (specify) — NONE

IV. What machine did you conduct the TREC experiment on? SPARC-2GS
How much RAM did it have? 48 MB
What was the clock rate of the CPU? 40 MHz

V. Some systems are research prototypes and others are commercial.
To help compare these systems:

1. How much "software engineering" went into the development of your system?
   TIME WAS SPENT TO TRUNCATE RECORD SIZES TO SAVE SPACE AND FIT CERTAIN STRUCTURES IN MEMORY; REPLACE SOME LINKED LISTS WITH ARRAYS.

2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?
   YES, 50-100%. LOTS OF CODE WAS TRANSLATED FROM PASCAL TO C & USED AS IS.

3. What features is your system missing that it would benefit by if it had them?
   DEDICATED SPARCSTATION WITH MORE RAM, DISK SPACE.

# NATURAL LANGUAGE PROCESSING IN LARGE-SCALE TEXT RETRIEVAL TASKS

Tomek Strzalkowski
*Courant Institute of Mathematical Sciences*
New York University
715 Broadway, rm. 704
New York, NY 10003
tomek@cs.nyu.edu

**ABSTRACT**

We developed a prototype text retrieval system which uses advanced natural language processing techniques to enhance the effectiveness of key-word based document retrieval. The backbone of our system is a traditional statistical engine which builds inverted index files from pre-processed documents, and then searches and ranks the documents in response to user queries. Natural language processing is used to (1) preprocess the documents in order to extract contents-carrying terms, (2) discover inter-term dependencies and build a conceptual hierarchy specific to the database domain, and (3) process user's natural language requests into effective search queries. For the present TREC effort, the total of 500 MBytes of Wall Street Journal articles have been processed in two batches of 250 MBytes each. Due to time and space limits, two separate inverted indexes were produced for each half of the data, with a partial concept hierarchy built from the first 250 Mbytes only but used for retrieval on either half. Retrieval were performed independently on both halfs of the database and the partial results were merged to produce the final rankings.

## INTRODUCTION

A typical information retrieval (IR) task is to select documents from a database in response to a user's query, and rank these documents according to relevance. This has been usually accomplished using statistical methods (often coupled with manual encoding) that (a) select terms (words, phrases, and other units) from documents that are deemed to best represent their contents, and (b) create an inverted index file (or files) that provide and easy access to documents containing these terms. A subsequent search process will attempt to match a preprocessed user query (or queries) against term-based representations of documents in each case determining a degree of relevance between the two which depends upon the number and types of matching terms. Although many sophisticated search and matching methods are available, the crucial problem remains to be that of an adequate representation of contents for both the documents and the queries.

The simplest word-based representations of contents are usually inadequate since single words are rarely specific enough for accurate discrimination, and their grouping is often accidental. A better method is to identify groups of words that create meaningful *phrases*, especially if these phrases denote important concepts in database domain. For example, *joint venture* is an important term in Wall Street Journal (WSJ henceforth) database, while neither *joint* nor *venture* is important by itself. In the retrieval experiments with the training TREC database, we noticed that both *joint* and *venture* were dropped from the list of terms by the system because their idf (*inverted document frequency*) weights were too low. In large databases, such as TIPSTER, the use of phrasal terms is not just desirable, it becomes necessary.

The question thus becomes, how to identify the correct phrases in the text? Both statistical and syntactic methods were used before with only limited success. Statistical methods based on word co-occurrences and mutual information are prone to high error rates, turning out many unwanted associations. Syntactic methods suffered from low quality of generated parse structures that could be attributed to limited coverage grammars and the lack of adequate lexicons. In fact, the difficulties encountered in applying computational linguistics technologies to text processing have contributed to a wide-spread belief that automated natural language processing may not be suitable in IR. These difficulties included inefficiency, lack of robustness, and prohibitive cost of manual effort required to build lexicons and knowledge bases for each new text domain. On the other hand, while numerous experiments did not establish the usefulness of linguistic methods in IR, they cannot be considered conclusive because of their

limited scale. [1]

The rapid progress in Computational Linguistics over the last few years has changed this equation in various ways. First of all, large-scale resources became available: on-line lexicons, including Oxford Advanced Learner's Dictionary (OALD), Longman Dictionary of Contemporary English (LDOCE), Webster's Dictionary, Oxford English Dictionary, Collins Dictionary, and others, as well as large text corpora, many of which can now be obtained for research purposes. Robust text-oriented software tools have been built, including part of speech taggers (stochastic and otherwise), and fast parsers capable of processing text at speeds of 2600 words per minute or more (e.g., TTP parser developed by the author). While many of the fast parsers are not very accurate (they are usually partial analyzers by design),[2] some, like TTP, perform in fact no worse than standard full-analysis parsers which are many times slower and far less robust. [3]

An accurate syntactic analysis is an essential prerequisite for term selection, but it is by no means sufficient. Syntactic parsing of the database contents is usually attempted in order to extract linguistically motivated phrases, which presumably are better indicators of contents than "statistical phrases" where words are grouped solely on the basis of physical proximity (e.g., "college junior" is not the same as "junior college"). However, creation of such compound terms makes term matching process more complex since in addition to the usual problems of synonymy and subsumption, one must deal with their structure (e.g., "college junior" is the same as "junior in college"). In order to deal with structure, parser's output needs to be "normalized" or "regularized" so that complex terms with the same or closely related meanings would indeed receive matching representations. This goal has been achieved to a certain extent in the present work. As it will be discussed in more detail below, indexing terms were selected from among head-modifier pairs extracted from predicate-argument representations of sentences.

Introduction of compound terms also complicates the task of discovery of various semantic relationships among them, including synonymy and subsumption. For example, the term *natural language* can be considered, in certain domains at least, to subsume any term denoting a specific human language, such as *English*. Therefore, a query containing the former may be expected to retrieve documents containing the latter. The same can be said about *language* and *English*, unless *language* is in fact a part of compound term *programming language* in which case the association *language - Fortran* is appropriate. This is a problem because (a) it is a standard practice to include both simple and compound terms in document representation, and (b) term associations have thus far been computed primarily on word level (including fixed phrases) and therefore care must be taken when such associations are used in term matching. This may prove particularly troublesome for systems that attempt term clustering in order to create "meta-terms" to be used in document representation.

The system presented here computes term associations from text on word and fixed phrase level and then uses these associations in query expansion. A fairly primitive filter is employed to separate synonymy and subsumption relationships from others including antonymy and complementation, some of which are strongly domain-dependent. This process has led to an increased retrieval precision in experiments with smaller and more cohesive collections (CACM-3204), but may be less effective with large databases. We are presently studying more advanced clustering methods along with the changes in interpretation of resulting associations, as signalled in the previous paragraph.

Working with TREC topics has also helped to identify other, perhaps unanticipated problems, some of which may render the traditional statistical approach to information retrieval quite unworkable. A typical document retrieval query will specify (in one way or another) a set of concepts that are of interest to the originator of the query. Thus if the user is interested in documents that report on anticipated rail strikes, the following query may be appropriate: (TREC topic 058) *A relevant document will report an impending rail strike ....* Any other wording can be used so long as it is going to denote a *concept* to be found in a document. The system's task is then to discover that the same concept is being denoted in both the query and a document, no matter how different the surface descriptions happen to be. In other words, no new information is requested, the query being entirely self contained and completely

---

[1] Standard IR benchmark collections are statistically too small and the experiments can easily produce counterintuitive results. For example, Cranfield collection is only approx. 180,000 English words, while CACM-3204 collection is approx. 200,000 words.

[2] Partial parsing is usually fast enough, but it also generates noisy data: as many as 50% of all generated phrases could be incorrect (Lewis and Croft, 1990).

[3] TTP has been shown to produce parse structures which are no worse in recall, precision and crossing rate than those generated by full-scale linguistic parsers when compared to hand-coded Treebank parse trees.

specified. [4] An altogether different situation arises when the query actually requests that certain underspecified information is found before a document could be judged relevant. In TREC topics (e.g., 058, as in many others) the following request is commonplace: *to be relevant, the document will identify the location of the strike or potential strike.* What we ask the system here is to *extract* the value of certain variable that satisfies certain conditions, i.e., find X such that location-of-strike(X). It is impossible to properly evaluate such query using any kind of constant term based retrieval. What is required, at the minimum is a general pattern matching capability, and appropriately advanced representation of contents (including more careful NLP processes).

In the remainder of this paper we discuss particulars of the present system and some of the observations made while processing TREC data. The above comments will provide the background for situating our present effort and state-of-the-art with respect to where we should be in the future.

## OVERALL DESIGN

Our information retrieval system consists of a traditional statistical backbone (Harman and Candela, 1989) augmented with various natural language processing components that assist the system in database processing (stemming, indexing, word and phrase clustering, selectional restrictions), and translate a user's information request into an effective query. This design is a careful compromise between purely statistical non-linguistic approaches and those requiring rather accomplished (and expensive) semantic analysis of data, often referred to as 'conceptual retrieval'. The conceptual retrieval systems, though quite effective, are not yet mature enough to be considered in serious information retrieval applications, the major problems being their extreme inefficiency and the need for manual encoding of domain knowledge (Mauldin, 1991). However, as pointed out in the previous section, a more careful text processing may be required for certain types of requests.

In our system the database text is first processed with a fast syntactic parser. Subsequently certain types of phrases are extracted from the parse trees and used as compound indexing terms in addition to single-word terms. The extracted phrases are statistically analyzed as syntactic contexts in order to discover a variety of similarity links between smaller subphrases and words occurring in them. A further filtering process maps these similarity links onto semantic relations (generalization, specialization, synonymy, etc.) after which they are used to transform user's request into a search query.

The user's natural language request is also parsed, and all indexing terms occurring in them are identified. Certain highly ambiguous, usually single-word terms may be dropped, provided that they also occur as elements in some compound terms. For example, "natural" is deleted from a query already containing "natural language" because "natural" occurs in many unrelated contexts: "natural number", "natural logarithm", "natural approach", etc. At the same time, other terms may be added, namely those which are linked to some query term through admissible similarity relations. For example, "unlawful activity" is added to a query (TREC topic 055) containing the compound term "illegal activity" via a synonymy link between "illegal" and "unlawful". After the final query is constructed, the database search follows, and a ranked list of documents is returned.

It should be noted that all the processing steps, those performed by the backbone system, and these performed by the natural language processing components, are fully automated, and no human intervention or manual encoding is required.

## FAST PARSING WITH TTP PARSER

TTP (Tagged Text Parser) is based on the Linguistic String Grammar developed by Sager (1981). The parser currently encompasses some 400 grammar productions, but it is by no means complete. The parser's output is a regularized parse tree representation of each sentence, that is, a representation that reflects the sentence's logical predicate-argument structure. For example, logical subject and logical object are identified in both passive and active sentences, and noun phrases are organized around their head elements. The significance of this representation will be discussed below. The parser is equipped with a powerful skip-and-fit recovery mechanism that allows it to operate effectively in the face of ill-formed input or under a severe time pressure. In the runs with approximately 83 million words of TREC's Wall Street Journal texts,[5] the parser's speed averaged between 0.45 and 0.5 seconds per sentence, or up to 2600 words per minute, on a 21 MIPS SparcStation ELC.

[4] This does not mean that the query has to accurately reflect the user's intentions. We take what we've got and give it our best try.

[5] Approximately 0.5 GBytes of text, over 4 million sentences.

TTP is a full grammar parser, and initially, it attempts to generate a complete analysis for each sentence. However, unlike an ordinary parser, it has a built-in timer which regulates the amount of time allowed for parsing any one sentence. If a parse is not returned before the allotted time elapses, the parser enters the skip-and-fit mode in which it will try to "fit" the parse. While in the skip-and-fit mode, the parser will attempt to forcibly reduce incomplete constituents, possibly skipping portions of input in order to restart processing at a next unattempted constituent. In other words, the parser will favor reduction to backtracking while in the skip-and-fit mode. The result of this strategy is an approximate parse, partially fitted using top-down predictions. The fragments skipped in the first pass are not thrown out, instead they are analyzed by a simple phrasal parser that looks for noun phrases and relative clauses and then attaches the recovered material to the main parse structure. As an illustration, consider the following sentence taken from the CACM-3204 corpus:

> The method is illustrated by the automatic construction of both recursive and iterative programs operating on natural numbers, lists, and trees, in order to construct a program satisfying certain specifications *a theorem induced by those specifications is proved,* and the desired program is extracted from the proof.

The italicized fragment is likely to cause additional complications in parsing this lengthy string, and the parser may be better off ignoring this fragment altogether. To do so successfully, the parser must close the currently open constituent (i.e., reduce *a program satisfying certain specifications* to *NP*), and possibly a few of its parent constituents, removing corresponding productions from further consideration, until an appropriate production is reactivated. In this case, TTP may force the following reductions: $SI \rightarrow$ *to V NP;* $SA \rightarrow SI$; $S \rightarrow$ *NP V NP SA*, until the production $S \rightarrow S$ *and* $S$ is reached. Next, the parser skips input to find *and*, and resumes normal processing.

As may be expected, the skip-and-fit strategy will only be effective if the input skipping can be performed with a degree of determinism. This means that most of the lexical level ambiguity must be removed from the input text, prior to parsing. We achieve this using a stochastic parts of speech tagger to preprocess the text. Full details of the parser can be found in (Strzalkowski, 1992).

## PART OF SPEECH TAGGER

One way of dealing with lexical ambiguity is to use a tagger to preprocess the input marking each word with a tags that indicates its syntactic categorization: a part of speech with selected morphological features such as number, tense, mode, case and degree. The following are tagged sentences from the CACM-3204 collection: [6]

> The/*dt* paper/*nn* presents/*vbz* a/*dt* proposal/*nn* for/*in* structured/*vbn* representation/*nn* of/*in* multiprogramming/*vbg* in/*in* a/*dt* high/*jj* level/*nn* language/*nn* ./*per*

> The/*dt* notation/*nn* used/*vbn* explicitly/*rb* associates/*vbz* a/*dt* data/*nns* structure/*nn* shared/*vbn* by/*in* concurrent/*jj* processes/*nns* with/*in* operations/*nns* defined/*vbn* on/*in* it/*pp* ./*per*

The tags are understood as follows: dt - determiner, nn - singular noun, nns - plural noun, in - preposition, jj - adjective, vbz - verb in present tense third person singular, to - particle "to", vbg - present participle, vbn - past participle, vbd - past tense verb, vb - infinitive verb, cc - coordinate conjunction.

Tagging of the input text substantially reduces the search space of a top-down parser since it resolves most of the lexical level ambiguities. In the examples above, tagging of *presents* as "vbz" in the first sentence cuts off a potentially long and costly "garden path" with *presents* as a plural noun followed by a headless relative clause starting with *(that) a proposal ....* In the second sentence, tagging resolves ambiguity of *used* (vbn vs. vbd), and *associates* (vbz vs. nns). Perhaps more importantly, elimination of word-level lexical ambiguity allows the parser to make projection about the input which is yet to be parsed, using a simple lookahead; in particular, phrase boundaries can be determined with a degree of confidence (Church, 1988). This latter property is critical for implementing skip-and-fit recovery technique outlined in the previous section.

Tagging of input also helps to reduce the number of parse structures that can be assigned to a sentence, decreases the demand for consulting of the dictionary, and simplifies dealing with unknown words. Since every item in the sentence is assigned a tag, so are the words for which we have no entry in the lexicon. Many of these words will be tagged as "np" (proper noun), however, the surrounding tags may force other selections. In the following example, *chinese*, which does not appear in the dictionary, is tagged as "jj":[7]

> this/*dt* paper/*nn* dates/*vbz* back/*rb* the/*dt* genesis/*nn* of/*in* binary/*jj* conception/*nn* circa/*in*

[6] Tagged using the 35-tag Penn Treebank Tagset created at the University of Pennsylvania.

[7] We use the machine readable version of the Oxford Advanced Learner's Dictionary (OALD).

5000/*cd* years/*nns* ago/*rb* ,/*com* as/*rb* derived/*vbn* by/*in* the/*dt* chinese/*jj* ancients/*nns* ./*per*

The tagger which we use to process the input text prior to parsing is based upon a bi-gram model; it selects most likely tag for a word given co-occurrence probabilities computed from a relatively small training set.[8] While the peak accuracy of the best-tag option of the tagger is predicted to approach 97% (Meteer et al., 1991), we noted that the actual performance on unprocessed WSJ text was in fact somewhat worse. The main problem, it appears, were frequent mistakes in tokenization of input, especially in recognizing sentence boundaries. For example, when a sentence ended with a period but wasn't followed by at least two blanks or an end-of-line, this and the next sentence would be collapsed together. On the other hand, intra-sentential periods (like those following abbreviated words) were occasionally found followed by a new-line character, and the sentence was split into two. While the parser contains a provision to deal with the case of collapsed sentences, the tags were likely to be incorrect. The following example is typical; note tagging errors at the second apostrophe, and *plans*.

Gorbachev was running into trouble at home, including the August coup, "which I thought would be the end of it," Mr. Costa says. Still, plans to send the tank to the U.S. somehow moved ahead.

Gorbachev/*np* was/*vbd* running/*vbg* into/*in* trouble/*nn* at/*in* home/*nn* ,/*com* including/*vbg* the/*dt* August/*np* coup/*nn* ,/*com* "/*apos* which/*wdt* I/*pp* thought/*vbd* would/*md* be/*vb* the/*dt* end/*nn* of/*in* it/*pp* ,/*com* "/*nn* Mr/*nn* ./*per*

Costa/*np* says/*vbz* ./*per* still/*rb* ,/*com* plans/*vbz* to/*to* send/*vb* the/*dt* tank/*nn* to/*to* the/*dt* U.S./*np* somehow/*rb* moved/*vbd* ahead/*rb* ./*per*

## WORD SUFFIX TRIMMER

Word stemming has been an effective way of improving document recall since it reduces words to their common morphological root, thus allowing more successful matches. On the other hand, stemming tends to decrease retrieval precision, if care is not taken to prevent situations where otherwise unrelated words are reduced to the same stem. In our system we replaced a traditional morphological stemmer with a conservative dictionary-assisted suffix trimmer. [9] The suffix trimmer performs essentially two tasks: (1) it reduces inflected word forms to their root forms as specified in the dictionary, and (2) it converts nominalized verb forms (e.g., "implementation", "storage") to the root forms of corresponding verbs (i.e., "implement", "store"). This is accomplished by removing a standard suffix, e.g., "stor+age", replacing it with a standard root ending ("+e"), and checking the newly created word against the dictionary, i.e., we check whether the new root ("store") is indeed a legal word, and whether the original root ("storage") is defined using the new root ("store") or one of its standard inflectional forms (e.g., "storing"). For example, the following definitions are excerpted from the *Oxford Advanced Learner's Dictionary* (OALD):

**storage** *n* [U] (space used for, money paid for) the storing of goods ...
**diversion** *n* [U] diverting ...
**procession** *n* [C] number of persons, vehicles, etc moving forward and following each other in an orderly way.

Therefore, we can reduce "diversion" to "divert" by removing the suffix "+sion" and adding root form suffix "+t". On the other hand, "process+ion" is not reduced to "process". [10]

Earlier experiments with CACM-3204 collection showed an improvement in retrieval precision by 6% to 8% over the base system equipped with a standard morphological stemmer (the SMART stemmer). Due to time limitations these numbers are not available for TREC database at this time.

## HEAD-MODIFIER STRUCTURES

Syntactic phrases extracted from TTP parse trees are head-modifier pairs. The head in such a pair is a central element of a phrase (main verb, main noun, etc.), while the modifier is one of the adjunct arguments of the head. In the TREC experiments reported here we extracted head-modifier word and fixed-phrase pairs only. While TREC WSJ database is large enough to warrant generation of larger compounds, we were in no position to verify their effectiveness in indexing (largely because of the tight schedule). We discuss some options below.

[8] The program, supplied to us by Bolt Beranek and Newman, operates in two alternative modes, either selecting a single most likely tag for each word (best-tag option, the one we use at present), or supplying a short ranked list of alternatives (Meteer et al., 1991).

[9] Dealing with prefixes is a more complicated matter, since they may have quite strong effect upon the meaning of the resulting term, e.g., *un-* usually introduces explicit negation.

[10] Definition checking is not implemented yet.

Let us consider a specific example from WSJ database:

> The former Soviet president has been a local hero ever since a Russian tank invaded Wisconsin.

The tagged sentence is given below, followed by the regularized parse structure generated by TTP, given in Figure 1.

> The/*dt* former/*jj* Soviet/*jj* president/*nn* has/*vbz* been/*vbn* a/*dt* local/*jj* hero/*nn* ever/*rb* since/*in* a/*dt* Russian/*jj* tank/*nn* invaded/*vbd* Wisconsin/*np* ./*per*

It should be noted that the parser's output is a predicate-argument structure centered around main elements of various phrases. In Figure 1, BE is the main predicate (modified by HAVE) with 2 arguments (*subject, object*) and 2 adjuncts (*adv, sub_ord*). INVADE is the predicate in the subordinate clause with 2 arguments (*subject, object*). The subject of BE is a noun phrase with PRESIDENT as the head element, two modifiers (FORMER, SOVIET) and a determiner (THE). From this structure, we extract head-modifier pairs that become candidates for compound terms. The following types of pairs are considered: (1) a head noun and its left adjective or noun adjunct, (2) a head noun and the head of its right adjunct, (3) the main verb of a clause and the head of its object phrase, and (4) the head of the subject phrase and the main verb. These types of pairs account for most of the syntactic variants for relating two words (or simple phrases) into pairs carrying compatible semantic content. For example, the pair *retrieve+information* will be extracted from any of the following fragments: *information retrieval system; retrieval of information from databases*; and *information that can be retrieved by a user-controlled interactive search process*. In the example at hand, the following head-modifier pairs are extracted (pairs containing low-contents elements, such as BE and FORMER, or names, such as WISCONSIN, will be later discarded):

[PRESIDENT,BE]
[PRESIDENT,FORMER]
[PRESIDENT,SOVIET]
[BE,HERO]
[HERO,LOCAL]
[TANK,INVADE]
[TANK,RUSSIAN]
[INVADE,WISCONSIN]

We may note that the three-word phrase *former Soviet president* has been broken into two pairs *former president* and *Soviet president*, both of which denote things that are potentially quite different from what the original phrase refers to, and this fact may have potentially negative effect on retrieval precision. This is one place where a longer phrase appears more appropriate. An further example is shown in Figure 2.[11] One difficulty in obtaining head-modifier pairs of highest accuracy is the notorious ambiguity of nominal compounds. For example, the phrase *natural language processing* should generate *language+natural* and *processing+language*, while *dynamic information processing* is expected to yield *processing+dynamic* and *processing+information*. Since our parser has no knowledge about the text domain, and uses no semantic preferences, it does not attempt to guess any internal associations within such phrases. Instead, this task is passed to the pair extractor module which processes ambiguous parse structures in two phases. In phase one, all and only unambiguous head-modifier pairs are extracted, and the frequencies of their occurrences are recorded. In phase two, frequency information about pairs generated in the first pass is used to form associations from ambiguous structures. For example, if

```
[assert
 [[perf [HAVE]]
  [[verb [BE]]
   [subject
    [np
     [n PRESIDENT]
     [t_pos THE]
     [adj [FORMER]]
     [adj [SOVIET]]]]
   [object
    [np
     [n HERO]
     [t_pos A]
     [adj [LOCAL]]]]
   [adv EVER]
   [sub_ord
    [SINCE
     [[verb [INVADE]]
      [subject
       [np
        [n TANK]
        [t_pos A]
        [adj [RUSSIAN]]]]
      [object
       [np
        [name [WISCONSIN]]]]]]]]]
```

**Figure 1.** Predicate-argument parse structure.

[11] Note that working with the parsed text ensures a degree of precision in capturing the meaningful phrases, which is especially evident when compared with the results usually obtained from either unprocessed or only partially processed text (Lewis and Croft, 1990). Note also that names, pronouns and dummy verbs are not allowed to create pairs.

```
SENTENCE:
Gorbachev ordered the tank shipped to a plant in the Ukraine,
where seven mechanics worked for three months restoring it.

PARSE STRUCTURE:
[assert
 [[verb [ORDER]]
  [subject
   [np
    [name [GORBACHEV]]]]
  [object
   [[verb []]
    [subject
     [np
      [n TANK]
      [t_pos THE]
      [m_wh
       [[verb [SHIP]]
        [subject ANYONE]
        [object VAR]
        [TO
         [np
          [n PLANT]
          [t_pos A]]]
        [IN
         [np
          [name [UKRAINE]]
          [t_pos THE]
          [m_wh
           [[verb [WORK]]
            [subject
             [np
              [n MECHANIC]
              [count [SEVEN]]]]
            [FOR
             [np
              [n MONTH]
              [count [THREE]]]]
            [sa_wh
             [[verb [RESTORE]]
              [subject PRO]
              [object
               [np
                [n IT]]]]]]]]]]]]]

EXTRACTED PAIRS:
MECHANIC WORK
SHIP TANK
TANK SHIP
```

**Figure 2.** Extraction of syntactic pairs.

*language+natural* has occurred unambiguously a number times in contexts such as *parser for natural language*, while *processing+natural* has occurred significantly fewer times or perhaps none at all, then we will prefer the former association as valid.

Although the noun phrase disambiguation routine has been implemented to work with the pair extractor program, it has not been used in the current installment of TREC. A more conservative version of pair extractor was used instead (it generates fewer pairs) since that version was found effective in test runs. [12]

## TERM CORRELATIONS FROM TEXT

Head-modifier pairs form compound terms used in database indexing. They also serve as occurrence contexts for smaller terms, including single-word terms. In order to determine whether such pairs signify any important association between terms, we calculate the value of the *Informational Contribution (IC)* function for each element in a pair. This is important because not every syntactic association necessarily translates into a semantic one, and we may also need to eliminate spurious pairs generated by the parser. Higher values of *IC* indicate stronger association, and moreover the element which has the larger value is considered semantically dominant. These values are context dependent, and will vary from one corpus to another. [13]

The likelihood of a given word being paired with another word, within one predicate-argument structure, can be expressed in statistical terms as a conditional probability. In our present approach, the required measure had to be uniform for all word occurrences, covering different types of predicate-argument links, i.e., verb-object, noun-adjunct, etc. This is reflected by an additional dispersion parameter, introduced to evaluate the heterogeneity of word associations. The resulting new formula $IC(x, [x,y])$ is based on (an estimate of) the conditional probability of seeing a word $y$ as a modifier of the word $x$, normalized with a dispersion parameter for $x$.

$$IC(x, [x,y]) = \frac{f_{x,y}}{n_x + d_x - 1}$$

where $f_{x,y}$ is the frequency of $[x,y]$ in the corpus, $n_x$ is the number of pairs in which $x$ occurs at the same position as in $[x,y]$, and $d(x)$ is the dispersion parameter understood as the number of distinct words with

[12] In a few test runs with TREC topics 001 to 005 against the training database (disk 1), we observed that the inclusion of compound terms obtained from head-modifier pairs increased both recall and precision quite sugnificantly. In particular, for topic 003 no relevant documents could be found without compound terms.

[13] For more details please refer to (Strzalkowski and Vauthey, 1992).

which $x$ is paired. When $IC(x,[x,y]) = 0$, $x$ and $y$ never occur together (i.e., $f_{x,y} = 0$); when $IC(x,[x,y]) = 1$, $x$ occurs only with $y$ (i.e., $f_{x,y} = n_x$ and $d_x = 1$).

So defined, IC function is asymmetric, a property found desirable by Wilks et al. (1990) in their study of word co-occurrences in the Longman dictionary. In addition, IC is quite stable even for relatively low frequency words (dispersion parameter helps here), and in this respect it compares favourably to Fano's mutual information formula. The lack of stability on low frequency terms is particularly worrisome for IR applications since many important indexing terms could be eliminated from consideration.

It should also be pointed out that the particular way of generating syntactic pairs was dictated, to some degree at least, by statistical considerations. Our original experiments with *IC* formula were performed on the relatively small CACM-3204 collection, and therefore we combined pairs obtained from different syntactic relations (e.g., verb-object, subject-verb, noun-adjunct, etc.) in order to increase frequencies of some associations. This became largely unnecessary in a large collection such as TIPSTER, but we had no means to test alternative options, and thus decided to stay with the original. It should not be difficult to see that this was a compromise solution, since many important distinctions were potentially lost, and strong associations could be produced where there weren't any. A way to improve things is to consider different syntactic relations independently, perhaps as independent sources of evidence that could lend support (or not) to certain term similarity predictions. We have already started testing this option. A few examples of IC coefficients obtained from CACM-3204 corpus are listed in Table 1.

IC values for terms become the basis for calculating term-to-term similarity coefficients. If two terms tend to be modified with a number of common modifiers and otherwise appear in few distinct contexts, we assign them a similarity coefficient, a real number between 0 and 1. The similarity is determined by comparing distribution characteristics for both terms within the corpus: how much information contents do they carry, do their information contribution over contexts vary greatly, are the common contexts in which these terms occur specific enough? In general we will credit high-contents terms appearing in identical contexts, especially if these contexts are not too commonplace.[14] The relative similarity

[14] It would not be appropriate to predict similarity between *language* and *logarithm* on the basis of their co-occurrence with *natural*.

| word | head+modifier | IC coeff. |
|---|---|---|
| *distribute* | *distribute+normal* | 0.040 |
| *normal* | *distribute+normal* | 0.115 |
| *minimum* | *minimum+relative* | 0.200 |
| *relative* | *minimum+relative* | 0.016 |
| *retrieve* | *retrieve+inform* | 0.086 |
| *inform* | *retrieve+inform* | 0.004 |
| *size* | *size+medium* | 0.009 |
| *medium* | *size+medium* | 0.250 |
| *editor* | *editor+text* | 0.142 |
| *text* | *editor+text* | 0.025 |
| *system* | *system+parallel* | 0.001 |
| *parallel* | *system+parallel* | 0.014 |
| *read* | *read+character* | 0.023 |
| *character* | *read+character* | 0.007 |
| *implicate* | *implicate+legal* | 0.035 |
| *legal* | *implicate+legal* | 0.083 |
| *system* | *system+distribute* | 0.002 |
| *distribute* | *system+distribute* | 0.037 |
| *make* | *make+recommend* | 0.024 |
| *recommend* | *make+recommend* | 0.142 |
| *infer* | *infer+deductive* | 0.095 |
| *deductive* | *infer+deductive* | 0.142 |
| *share* | *share+resource* | 0.054 |
| *resource* | *share+resource* | 0.042 |

**Table 1.** Examples of IC coefficients.

between two words $x_1$ and $x_2$ is obtained using the following formula ($\alpha$ is a large constant): [15]

$$SIM(x_1,x_2) = log\,(\alpha \sum_y sim_y(x_1,x_2))$$

where

$$sim_y(x_1,x_2) = MIN\,(IC\,(x_1,[x_1,y]),IC\,(x_2,[x_2,y])) * MIN\,(IC\,(y,[x_1,y]),IC\,(y,[x_2,y]))$$

The similarity function is further normalized with respect to $SIM\,(x_1,x_1)$.

In addition, we require that words $x_1$ and $x_2$ appear in at least two *distinct* common contexts, where a common context is a couple of pairs $[x_1,y]$ and $[x_2,y]$, or $[y,x_1]$ and $[y,x_2]$ such that they each occurred at least twice. Thus, *banana* and *baltic* will

[15] This is inspired by a formula used by Hindle (1990), and subsequently modified to take into account the asymmetry of IC measure.

not be considered for similarity relation on the basis of their occurrences in the common context of *republic*, no matter how frequent, unless there is another such common context comparably frequent (there wasn't any in TREC WSJ database). [16]

It may be worth pointing out that the similarities are calculated using term co-occurrences in syntactic rather than in document-size contexts, the latter being the usual practice in non-linguistic clustering (e.g., Sparck Jones and Barber, 1971; Crouch, 1988; Lewis and Croft, 1990). Although the two methods of term clustering may be considered mutually complementary in certain situations, we believe that more and stronger associations can be obtained through syntactic-context clustering, given sufficient amount of data and a reasonably accurate syntactic parser.[17]

## QUERY EXPANSION

Similarity relations are used to expand user queries with new terms, in an attempt to make the final search query more comprehensive (adding synonyms) and/or more pointed (adding specializations).[18] It follows that not all similarity relations will be equally useful in query expansion, for instance, complementary and antonymous relations like the one between *australian* and *canadian*, or *accept* and *reject* may actually harm system's performance, since we may end up retrieving many irrelevant documents. Similarly, the effectiveness of a query containing *vitamin* is likely to diminish if we add a similar but far more general term such as *acid*. On the other hand, database search is likely to miss relevant documents if we overlook the fact that *fortran* is a *programming language*, or that *infant* is a *baby* and *baby* is a *child*. We noted that an average set of similarities generated from a text corpus contains about as many "good" relations (synonymy, specialization) as "bad" relations (antonymy, complementation, generalization), as seen from the query expansion viewpoint. Therefore any attempt to separate these two classes and to increase the proportion of "good" relations should result in improved retrieval. This has indeed been confirmed in our earlier experiments where a relatively crude filter has visibly increased retrieval precision.

In order to create an appropriate filter, we expanded the IC function into a global specificity measure called the *cumulative informational contribution function* (ICW). ICW is calculated for each term across all contexts in which it occurs. The general philosophy here is that a more specific word/phrase would have a more limited its use, i.e., a more specific term would appear in fewer *distinct* contexts. In this respect, ICW is similar to the standard *inverted document frequency (idf)* measure except that term frequency is measured over syntactic units rather than document size units.[19] Terms with higher ICW values are generally considered more specific, but the specificity comparison is only meaningful for terms which are already known to be similar. The new function is calculated according to the following formula:

$$ICW(w) = \begin{cases} IC_L(w) * IC_R(w) & \text{if } \textit{both exist} \\ IC_R(w) & \text{if } \textit{only } IC_R(w) \textit{ exists} \\ 0 & \textit{otherwise} \end{cases}$$

where (with $n_w$, $d_w > 0$):

$$IC_L(w) = IC([w,\_]) = \frac{n_w}{d_w(n_w+d_w-1)}$$

$$IC_R(w) = IC([\_,w]) = \frac{n_w}{d_w(n_w+d_w-1)}$$

For any two terms $w_1$ and $w_2$, and constants $\delta_1 > 1$, $\delta_2 > 1$, the following situations were considered. If $ICW(w_2) \geq \delta_1 * ICW(w_1)$ then $w_2$ is considered more specific than $w_1$. If $ICW(w_2) < \delta_2 * ICW(w_1)$ and $ICW(w_2) > ICW(w_1)$ then $w_2$ is considered synonymous with $w_1$.[20] In addition, if $SIM_{norm}(w_1,w_2) = \sigma > \theta$, where $\theta$ is an empirically established threshold, then $w_2$ can be added to the query containing term $w_1$ with weight $\sigma$.[21] The

[16] However, *SIM(banana,dominican)* was generated since two independent contexts were indeed found: *republic* and *plant*, even though word *plant* apparently occurs in different senses in *banana plant* and *dominican plant*.

[17] Non-syntactic contexts cross sentence boundaries with no fuss, which is helpful with short, succinct documents (such as CACM abstracts), but less so with longer texts; see also (Grishman et al., 1986).

[18] Query expansion (in the sense considered here, though not quite in the same way) has been used in information retrieval research before (e.g., Sparck Jones and Tait, 1984; Harman, 1988), usually with mixed results. An altemative is to use term clusters to create new terms, "metaterms", and use them to index the database instead (e.g., Crouch, 1988; Lewis and Croft, 1990). We found that the query expansion approach gives the system more flexibility, for instance, by making room for hypertext-style topic exploration via user feedback.

[19] We believe that measuring term specificity over document-size contexts (e.g., Sparck Jones, 1972) may not be appropriate in this case. In particular, syntax-based contexts allow for processing texts without any internal document structure.

[20] In TREC runs we used $\delta_1 = 10$ and $\delta_2 = 3$.

[21] For CACM-3204 collection the filter was most effective at $\sigma = 0.57$. For TREC we changed the similarity formula slightly in order to obtain correct normalizations in all cases. This however lowered similarity coefficients in general and a new threshold had to be selected. We used $\sigma = 0.1$ in TREC runs.

cutoffs could be different for synonymy and specialization. For example, the following were obtained from TREC WSJ training database:

$ICW(child) = 0.000001$
$ICW(baby) = 0.000013$
$ICW(infant) = 0.000055$

with

$SIM(child,infant) = 0.131381$
$SIM(baby,child) = 0.183064$
$SIM(baby,infant) = 0.323121$

Therefore both *baby* and *infant* can be used to specialize *child* (with $\delta_1 = 10$), while *baby* and *infant* can be considered synonyms (with $\delta_2 = 5$). Note that if $\delta$ is well chosen, then the above filter will also help to reject antonymous and complementary relations, such as $SIM_{norm}(back,front)=0.305287$ with $ICW(back)=0.000001$ and $ICW(front)=0.000006$. We continue working to develop more effective filters. Examples of filtered similarity relations obtained from TREC corpus are listed in Tables 2 and 3.

## SUMMARY OF RESULTS

We have processed the total of 500 MBytes of articles from Wall Street Journal section of TREC database. Runs were made independently on training corpus (250 MBytes) and the test corpus (another 250 Mbytes). Natural language processing of each half of the corpus, including tagging, stemming, parsing and pair extraction required approximately 2 weeks of nearly uninterrupted processing on two Sparc workstations (we use an assortment of SparcStation 1, ELC and 2, with MIPS rates varying from 13 to 21 to 28.5). [22] Computing term similarities from the first 250 Mbytes of text took up another week. This time limitations in our computing resources, mostly memory, were the main reason. With a sufficient amount of RAM (on the order of 256 MBytes or more) we estimated that this process should take no more than 20 hours. It should be noted that we computed term similarities based only on the training corpus, but we used them in retrieval from either database. We assumed that the underlying set of concepts and describing them terms does not vary greatly between the two databases (the first covered Wall Street Journal from 1987 to 1989, the second from 1990 to 1992).

Subsequent indexing process performed by the NIST IR system required additional 2 weeks for each half of the database, on a single SparcStation 2. [23] In total, we created three inverted file indexes: two were derived from the training corpus, and one from the test corpus. The first index created from training corpus did not include compound terms; the remaining two indexes included them. No cumulative index was produced, since we estimated it would require more than 6 weeks to build. Some interesting discrepancies were observed. While the postings file generated from the training corpus was 25% larger that the one generated from the test corpus, the corresponding dictionary of terms was about 25% smaller in the training corpus. This seems to suggest that in 1990-92 volumes of Wall Street Journal (test corpus) a greater number of unique terms were used, while these terms were on average occurring more sparsely in the text. This fact may partially undermine an earlier assumption that term similarities generated from training corpus were adequate for test corpus as well.

During the training stage of TREC we performed only very limited tests, mostly because the base system retrieval was quite slow on a database of this size (approx. 173 MBytes, 99+K documents). [24] We run experiments with topics 001 to 005, using relevance judgements supplied with the training database. The purpose of these tests was to (1) observe any improvement in performance resulting from the use of compound phrase terms, (2) tune the term similarity filter by adjusting certain threshold values in an attempt to obtain the most effective set of similarities, and (3) note if any manual intervention into the queries could improve retrieval results. As may be expected, none of these experiments were conclusive, but this was all we could rely upon in the short time given. First of all, we noticed that system overall performance (cumulative recall and precision levels) were what we could have expected from runs on smaller benchmark collections such as CACM-3204. This may be partially explained by the diversity of the subject matter covered in the TREC data, and also by the character of the queries (i.e., requests to *extract* information), but some problem with the base search engine may also be to blame. [25] We were

[22] Processing of the training corpus took significantly longer in practice (real time) since we had only one Sparcstation available at any given time, and also due too errors made and resulting from them need for re-processing some parts.

[23] Indexing process could not be done in parallel because NIST system requires serial processing. Moreover, as indexing progressed, and the partial database grows in size, the process slowed down significantly from approx. 6 min/file to 37 min/file.

[24] The system needed approx. 60 minutes to process a query.

[25] In the middle of the training run we discovered that the NIST system has been designed to handle databases up to 65K documents ($2^{16}$). Subsequently we rewrote portions of the code to fix this, but we were not certain if other decisions made by the retrieval module (e.g., bitmapping, idf cutoffs) were not in fact coun-

| word1 | word2 | SIMnorm |
|---|---|---|
| *abm* | **anti+ballistic* | 0.534894 |
| *absence* | **maternity* | 0.233082 |
| *accept* | *acquire* | 0.179078 |
| *accord* | *pact* | 0.492332 |
| *acquire* | *purchase* | 0.449362 |
| *speech* | *address* | 0.263789 |
| *adjustable* | *one+year* | 0.824053 |
| *maxsaver* | **advance+purchase* | 0.734008 |
| *affair* | *scandal* | 0.684877 |
| *affordable* | *low+income* | 0.181795 |
| *disease* | **ailment* | 0.247382 |
| *medium+range* | **air+to+air* | 0.874508 |
| *aircraft* | **jetliner* | 0.166777 |
| *aircraft* | *plane* | 0.423831 |
| *airline* | *carrier* | 0.345490 |
| *alien* | *immigrate* | 0.270412 |
| *anniversary* | **bicentennial* | 0.588210 |
| *anti+age* | *anti+wrinkle* | 0.153918 |
| *anti+clot* | *cholesterol+lower* | 0.856712 |
| *contra* | **anti+sandinista* | 0.294677 |
| *candidate* | **aspirant* | 0.116025 |
| *contend* | **aspirant* | 0.143459 |
| *property* | *asset* | 0.285299 |
| *attempt* | *bid* | 0.641592 |
| *await* | *pend* | 0.572960 |
| *stealth* | **b+1* | 0.877582 |
| *child* | **baby* | 0.183064 |
| *baggage* | *luggage* | 0.607333 |
| *ban* | *restrict* | 0.321943 |
| *bearish* | *bullish* | 0.847103 |
| *bee* | **honeybee* | 0.461023 |
| *roller+coast* | **bumpy* | 0.898278 |
| *two+income* | *two+earner* | 0.293104 |
| *television* | *tv* | 0.806018 |
| *soldier* | *troop* | 0.374410 |
| *treasury* | **short+term* | 0.661133 |
| *research* | *study* | 0.209257 |
| *withdrawal* | **pullout* | 0.622558 |

**Table 2.** Filtered word similarities (* indicates the more specific term).

terproductive with larger databases. Later experiment have confirmed this as we obtained new code from NIST.

| word1 | word2 | SIMnorm |
|---|---|---|
| *lord* | *abuser* | 0.261507 |
| *break* | *accelerator* | 0.106775 |
| *accept* | *reject* | 0.275802 |
| *accord* | *level* | 0.152023 |
| *cent* | *accrue* | 0.259478 |
| *acquire* | *deficient* | 0.615525 |
| *fix+rate* | *adjustable* | 0.930180 |
| *advertise* | *environmental* | 0.124026 |
| *petroleum* | *aerospace* | 0.207406 |
| *afghan* | *nicaraguan* | 0.421478 |
| *banana* | *dominican* | 0.444193 |
| *begin* | *month* | 0.346558 |
| *german* | *british* | 0.112465 |
| *superpower* | *reagan+gorbachev* | 0.400145 |
| *republic* | *sea* | 0.227662 |
| *sunglasses* | *sneakers* | 0.126749 |

**Table 3.** Some (apparently?) "bad" similarities generated.

also quite disappointed with the effect that the query expansion had (or did not have) on either precision or recall figures. This was somewhat more directly related to the wide domain scope of WSJ articles, and we noticed many obvious word sense mixups leading to unwanted term associations. Also, the rather straightforward method of query expansion terms proved less adequate here, and we are considering new methods as outlined briefly in the introduction to this paper. Our final test with manually altered queries (terms were deleted or added after consulting the original topic) brought only negative results: any kind of intervention appeared only to further decrease both recall and precision.

Final runs were performed as follows:

(1) Topics 1-25 were processed using only direct fields (<title>, <desc>, and <narr>) with respect to the training database (disk 1). They were subsequently run in the routing mode against the test database (disk 2). [26] Unfortunately, the routing results were hindered by a bug in our program that removed duplicate

[26] While other fields provided much useful information about various aspects of a given topic (including definitions of specialized terms), we thought that their inclusion in the final query would make it difficult to assess the advantages of linguistic processing. This especially applies to Concepts fiels <con> which listed hand-extracted keywords. Subsequently, we realized that more often this field in fact provided further clues not found in other sections of a topic.

terms from queries while preparing them for a routing run.

(2) Topics 51-75 were processed using the same 3 fields as above with respect to the test database only (disk 2). They were then run in ad-hoc mode independently against the training database and the test database, and the top 200 documents from each run (for each query) were merged using document scores. The upper 200 documents were selected for the final result. It should be noted that we run topics against disk 1 using *idf* weights on terms from disk 2 (a routing-like mode). This, it turned out, created an even distribution of hits between the two databases. We call this method the *uniform merge*.

(3) We repeated run (2), but this time we run topics 51-75 in ad-hoc mode on both disks (i.e., different weights were used in these runs). Again, top 200 documents were selected. We call this method the *brute force merge*.

(4) Finally, we repeated uniform merge with queries manually pruned before actual retrieval.

Summary statistics for these runs are shown in Tables 4 and 5. It has to be pointed out that these results are lower than expected because of various problems with the database search program and because the Concepts field in TREC topics was not included in search queries. Subsequent runs with Concepts field included in search produced results shown in Table 6.


## ACKNOWLEDGEMENTS

We would like to thank Donna Harman of NIST for making her IR system available to us. We would also like to thank Ralph Weischedel, Marie Meteer and Heidi Fox of BBN for providing and assisting in the use of the part of speech tagger. Jose Perez Carballo has contributed a number of valuable observations during the course of this work, and his assistance in processing the TREC data was critical for our being able to finish on time. This paper is based upon work supported by the Defense Advanced Research Project Agency under Contract N00014-90-J-1851 from the Office of Naval Research, under Contract N00600-88-D-3717 from PRC Inc., and the National Science Foundation under Grant IRI-89-02304. We also acknowledge support from Canadian Institute for Robotics and Intelligent Systems (IRIS).


| Run Name | nyuir1 routing | nyuir2 routing |
|---|---|---|
| Queries | 25 | 25 |
| Tot number of docs over all queries | | |
| Ret | 5000 | 5000 |
| Rel | 3766 | 3766 |
| RelRet | 834 | 1177 |
| Recall | Precision Averages | |
| 0.00 | 0.5691 | 0.6983 |
| 0.10 | 0.3618 | 0.5072 |
| 0.20 | 0.1846 | 0.3508 |
| 0.30 | 0.1314 | 0.2997 |
| 0.40 | 0.1065 | 0.2286 |
| 0.50 | 0.0481 | 0.1481 |
| 0.60 | 0.0295 | 0.0604 |
| 0.70 | 0.0104 | 0.0296 |
| 0.80 | 0.0104 | 0.0260 |
| 0.90 | 0.0000 | 0.0125 |
| 1.00 | 0.0000 | 0.0000 |
| Average precision for all points | | |
| 11-pt | 0.1320 | 0.2147 |
| Average precision at 0.20, 0.50, 0.80 | | |
| 3-pt | 0.0811 | 0.1750 |
| Recall at | | |
| 5 docs | 0.0338 | 0.0374 |
| 15 docs | 0.0671 | 0.0961 |
| 30 docs | 0.0979 | 0.1533 |
| 100 docs | 0.2117 | 0.3119 |
| 200 docs | 0.2988 | 0.4298 |
| Precision at | | |
| 5 docs | 0.3360 | 0.4480 |
| 15 docs | 0.2960 | 0.4587 |
| 30 docs | 0.2787 | 0.3973 |
| 100 docs | 0.2276 | 0.3080 |
| 200 docs | 0.1668 | 0.2354 |

Table 4. Routing run statistics: (1) duplicate terms (unintentionally) removed from queries; and (2) duplicate terms retained and Concepts fields included.

| Run Name | nyuir1 brute | nyuir2 prune | nyuir3 uniform |
|---|---|---|---|
| Queries | 25 | 25 | 25 |
| Tot number of docs over all queries | | | |
| Ret | 4986 | 4990 | 4989 |
| Rel | 3318 | 3318 | 3318 |
| RelRet | 1161 | 958 | 1039 |
| Recall | Precision Averages | | |
| 0.00 | 0.7830 | 0.7168 | 0.7368 |
| 0.10 | 0.6216 | 0.4858 | 0.5592 |
| 0.20 | 0.4891 | 0.3722 | 0.4660 |
| 0.30 | 0.3815 | 0.2032 | 0.3235 |
| 0.40 | 0.2350 | 0.1382 | 0.2099 |
| 0.50 | 0.1871 | 0.0599 | 0.1385 |
| 0.60 | 0.1447 | 0.0419 | 0.0801 |
| 0.70 | 0.0705 | 0.0058 | 0.0560 |
| 0.80 | 0.0229 | 0.0030 | 0.0056 |
| 0.90 | 0.0000 | 0.0000 | 0.0000 |
| 1.00 | 0.0000 | 0.0000 | 0.0000 |
| Average precision for all points | | | |
| 11-pt | 0.2669 | 0.1843 | 0.2341 |
| Average precision at 0.20, 0.50, 0.80 | | | |
| 3-pt | 0.2330 | 0.1450 | 0.2034 |
| Recall at | | | |
| 5 docs | 0.0713 | 0.0444 | 0.0571 |
| 15 docs | 0.1326 | 0.0929 | 0.1297 |
| 30 docs | 0.1868 | 0.1424 | 0.1734 |
| 100 docs | 0.3350 | 0.2550 | 0.2918 |
| 200 docs | 0.4294 | 0.3466 | 0.3908 |
| Precision at | | | |
| 5 docs | 0.5680 | 0.5200 | 0.5360 |
| 15 docs | 0.5200 | 0.4267 | 0.4907 |
| 30 docs | 0.4320 | 0.3627 | 0.4147 |
| 100 docs | 0.3140 | 0.2684 | 0.2916 |
| 200 docs | 0.2322 | 0.1916 | 0.2078 |

Table 5. Ad-hoc run statistics with Automatic Brute Force Merge, Uniform Merge with hand pruning, and Automatic Uniform Merge.

| Run Name | nyuir1 brute | nyuir4 concepts | nyuir5 negations |
|---|---|---|---|
| Queries | 25 | 25 | 25 |
| Tot number of docs over all queries | | | |
| Ret | 4986 | 4984 | 4984 |
| Rel | 3318 | 3318 | 3318 |
| RelRet | 1161 | 1291 | 1309 |
| Recall | Precision Averages | | |
| 0.00 | 0.7830 | 0.7685 | 0.7823 |
| 0.10 | 0.6216 | 0.6521 | 0.6625 |
| 0.20 | 0.4891 | 0.5396 | 0.5460 |
| 0.30 | 0.3815 | 0.4306 | 0.4419 |
| 0.40 | 0.2350 | 0.2671 | 0.2755 |
| 0.50 | 0.1871 | 0.2094 | 0.2085 |
| 0.60 | 0.1447 | 0.1457 | 0.1457 |
| 0.70 | 0.0705 | 0.0660 | 0.0660 |
| 0.80 | 0.0229 | 0.0385 | 0.0385 |
| 0.90 | 0.0000 | 0.0000 | 0.0000 |
| 1.00 | 0.0000 | 0.0000 | 0.0000 |
| Average precision for all points | | | |
| 11-pt | 0.2669 | 0.2834 | 0.2879 |
| Average precision at 0.20, 0.50, 0.80 | | | |
| 3-pt | 0.2330 | 0.2625 | 0.2643 |
| Recall at | | | |
| 5 docs | 0.0713 | 0.0738 | 0.0741 |
| 15 docs | 0.1326 | 0.1362 | 0.1386 |
| 30 docs | 0.1868 | 0.2007 | 0.2011 |
| 100 docs | 0.3350 | 0.3513 | 0.3565 |
| 200 docs | 0.4294 | 0.4739 | 0.4828 |
| Precision at | | | |
| 5 docs | 0.5680 | 0.6080 | 0.6080 |
| 15 docs | 0.5200 | 0.5360 | 0.5493 |
| 30 docs | 0.4320 | 0.4760 | 0.4773 |
| 100 docs | 0.3140 | 0.3432 | 0.3484 |
| 200 docs | 0.2322 | 0.2582 | 0.2618 |

Table 6. Ad-hoc run statistics using Automatic Brute Force Merge: without Concepts field, with Concepts field, and with Concepts excluding negated terms.

## REFERENCES


Church, Kenneth Ward and Hanks, Patrick. 1990. "Word association norms, mutual information, and lexicography." *Computational Linguistics*, 16(1), MIT Press, pp. 22-29.

Crouch, Carolyn J. 1988. "A cluster-based approach to thesaurus construction." Proceedings of ACM SIGIR-88, pp. 309-320.

Grishman, Ralph, Lynette Hirschman, and Ngo T. Nhan. 1986. "Discovery procedures for sublanguage selectional patterns: initial experiments". *Computational Linguistics,* 12(3), pp. 205-215.

Grishman, Ralph and Tomek Strzalkowski. 1991. "Information Retrieval and Natural Language Processing." Position paper at the workshop on Future Directions in Natural Language Processing in Information Retrieval, Chicago.

Harman, Donna. 1988. "Towards interactive query expansion." Proceedings of ACM SIGIR-88, pp. 321-331.

Harman, Donna and Gerald Candela. 1989. "Retrieving Records from a Gigabyte of text on a Minicomputer Using Statistical Ranking." *Journal of the American Society for Information Science,* 41(8), pp. 581-589.

Hindle, Donald. 1990. "Noun classification from predicate-argument structures." Proc. 28 Meeting of the ACL, Pittsburgh, PA, pp. 268-275.

Lewis, David D. and W. Bruce Croft. 1990. "Term Clustering of Syntactic Phrases". Proceedings of ACM SIGIR-90, pp. 385-405.

Mauldin, Michael. 1991. "Retrieval Performance in Ferret: A Conceptual Information Retrieval System." Proceedings of ACM SIGIR-91, pp. 347-355.

Meteer, Marie, Richard Schwartz, and Ralph Weischedel. 1991. "Studies in Part of Speech Labelling." Proceedings of the 4th DARPA Speech and Natural Language Workshop, Morgan-Kaufman, San Mateo, CA. pp. 331-336.

Sager, Naomi. 1981. *Natural Language Information Processing.* Addison-Wesley.

Sparck Jones, Karen. 1972. "Statistical interpretation of term specificity and its application in retrieval." *Journal of Documentation,* 28(1), pp. 11-20.

Sparck Jones, K. and E. O. Barber. 1971. "What makes automatic keyword classification effective?" *Journal of the American Society for Information Science,* May-June, pp. 166-175.

Sparck Jones, K. and J. I. Tait. 1984. "Automatic search term variant generation." *Journal of Documentation,* 40(1), pp. 50-66.

Strzalkowski, Tomek and Barbara Vauthey. 1991. "Fast Text Processing for Information Retrieval." Proceedings of the 4th DARPA Speech and Natural Language Workshop, Morgan-Kaufman, pp. 346-351.

Strzalkowski, Tomek and Barbara Vauthey. 1992. "Information Retrieval Using Robust Natural Language Processing." Proc. of the 30th ACL Meeting, Newark, DE, June-July. pp. 104-111.

Strzalkowski, Tomek. 1992. "TTP: A Fast and Robust Parser for Natural Language." Proceedings of the 14th International Conference on Computational Linguistics (COLING), Nantes, France, July 1992. pp. 198-204.

Wilks, Yorick A., Dan Fass, Cheng-Ming Guo, James E. McDonald, Tony Plate, and Brian M. Slator. 1990. "Providing machine tractable dictionary tools." *Machine Translation,* 5, pp. 99-154.


## APPENDIX A: EXAMPLE QUERY

We show TREC topic 057 and a part of resulting search query with only top ranked terms showed in Table A.1. Please note that we rank terms by their *idf* scores even though their actual scores are *idf* * *weight*. It is worth pointing out, however, that the NIST system uses *idf* scores to decide if a term falls below a preset "significance" threshold. We only show fields used for query generation.

<top>
<num> Number: 057
<title> Topic: MCI
<desc> Description:
Document will discuss how MCI has been doing since the Bell System breakup.
<narr> Narrative:
A relevant document will discuss the financial health of MCI Communications Corp. since the breakup of the Bell System (AT&T and the seven regional Baby Bells) in January 1984. The status indicated may not necessarily be a direct or indirect result of the breakup of the system and ensuing regulation and deregulation of Ma Bell or of the restrictions placed upon the seven Bells; it may result from any number of factors, such as advances in telecommunications technology, MCI initiative, etc. MCI's financial health may be reported directly: a broad statement about its earnings or cash flow, or a report containing financial data such as a quarterly report; or it may be reflected by one or more of the following: credit ratings, share of customers, volume growth, cuts in capital spending, figure net loss, pre-tax charge, analysts' or MCI's own forecast about how well they will be doing, or MCI's response to price cuts that AT&T makes at its own initiative or under orders from the Federal Communications Commission (FCC), such as price reductions, layoffs of employees out of a perceived need to cut costs, etc. Daily OTC trading stock market and monthly short interest reports are NOT relevant; the inventory must be longer term, at least quarterly.
</top>

| term | idf | weight |
|---|---|---|
| *deposit+financial* | 16.185341 | 0.192091 |
| *result+breakup* | 16.185341 | 1.000000 |
| *restrict+action* | 16.185341 | 0.441260 |
| *number+factor* | 16.185341 | 1.000000 |
| *deposit+financial* | 16.185341 | 0.192091 |
| *layoff+job* | 15.600377 | 0.199677 |
| *report+health* | 15.185340 | 1.000000 |
| *share+cut* | 15.185340 | 1.000000 |
| *benefit+price* | 14.863412 | 0.263889 |
| *share+annual* | 14.600377 | 0.249365 |
| *document+relevant* | 14.377985 | 1.000000 |
| *growth+custom* | 14.185340 | 1.000000 |
| *need+perceive* | 14.185340 | 1.000000 |
| *report+short* | 14.185340 | 1.000000 |
| *earnings+cash* | 14.015415 | 1.000000 |
| *report+interest* | 13.725908 | 1.000000 |
| *share+benefit* | 13.484900 | 0.263889 |
| *restrict+place* | 13.185340 | 1.000000 |
| *share+growth* | 13.097878 | 1.000000 |
| *layoff+employee* | 13.015415 | 1.000000 |
| *breakup+system* | 12.661778 | 1.000000 |
| *bell+alarm* | 12.484900 | 0.303537 |
| *number+adjust* | 12.377985 | 0.421292 |
| *hoard+cash* | 12.278449 | 0.253572 |
| *rebate+cash* | 12.015415 | 0.112116 |
| *growth+volume* | 11.661778 | 1.000000 |
| *flow+earnings* | 11.512915 | 1.000000 |
| *pre+tax* | 11.430452 | 1.000000 |
| *report+monthly* | 11.056057 | 1.000000 |
| *flow+capital* | 10.827788 | 0.434692 |
| *report+quarterly* | 10.827788 | 1.000000 |
| *bell+baby* | 10.541484 | 1.000000 |
| *reduce+price* | 10.314976 | 1.000000 |
| *health+financial* | 10.005431 | 1.000000 |
| *hoard* | 9.956521 | 0.253572 |
| *infusion+cash* | 9.717734 | 0.332548 |
| *ensue* | 9.505860 | 1.000000 |
| *stock+trade* | 9.327359 | 1.000000 |
| *boost+price* | 9.225338 | 0.280143 |
| *breakup* | 9.135491 | 1.000000 |
| *disposable* | 9.108524 | 0.216766 |
| *rebate* | 8.918553 | 0.112116 |
| *fcc* | 8.810301 | 1.000000 |
| *cut+price* | 8.763275 | 1.000000 |
| *mci* | 8.555984 | 1.000000 |

**Table A.1.** Partial search query for topic 057

OCLC Online Computer Library Center, Inc.
Roger Thompson

## Introduction

We are interested in determining the extent to which the effects of syntactic phrase indexing scales up to large databases. Previous investigations of the hypothesis that syntactic phrase indexing leads to improvements in retrieval performance were conducted on databases ranging from 250 records to a few thousand records (Dillon and Gray 1983, Fagan 1987, Lewis 1991, Burgin and Dillon 1992). The results have been conflicting or equivocal. However, we believe that the issue isn't settled. Technologies for phrase extraction and thesaurus construction have been evolving, and the arguments favoring syntactic phrase indexing have not been convincingly dispelled.

The major argument in favor of phrase indexing is that it is a precision enhancer because words in context are less ambiguous than isolated terms, and because documents represented in indexes as key phrases indicative of their content instead of the sum of their individual terms have undergone considerable noise reduction. Nevertheless, we believe that there are upper limits to the effectiveness of syntactic phrase indexing. One of our goals in this project is to understand and elucidate these limits.

In keeping with our interest in answering the simple question of whether syntactic phrase indexing scales up, we tested the effectiveness of an existing program for phrase extraction, FASIT (described in Dillon and Gray 1983, Dillon and McDonald 1983 and Burgin and Dillon 1992), in the SMART retrieval environment. The primary advantages of FASIT are that the phrase extraction process is fully automatic, the parse is shallow and time-efficient, and the logic is table-driven and easily modified. Thus, FASIT represents a kind of lower-bound estimate of what is necessary to enhance retrieval performance through automatic indexing.

## FASIT Description

FASIT identifies noun phrases appropriate for indexing by determinining the part of speech for each word in the input text. This is done by looking up the word in a dictionary created by assigning tags derived from the Brown Corpus to all entries in the Oxford Advanced Learner's Dictionary. If the word is not found in the dictionary, its part of speech is determined from the word's suffix. Words with more than one part speech have multiple tag assignments and are eventually disambiguated by examining the tags of the words in the surrounding context.

Once tagging is complete, the concept selection module consults a template to identify index phrases. Concepts in FASIT are a subset of the noun phrases encountered in the input text which are judged by syntactic criteria to be useful for indexing. These include all proper nouns, adjective-noun combinations such as "federal agency"; noun-noun combinations such as "metals technology"; or noun-prepositional phrase combinations such as "maker of furniture", which might be paraphrased as the noun-noun construction "furniture maker". The selected concepts are normalized by eliminating determiners and pronouns, and the head noun is stemmed.

Table I shows a portion of a sentence as it passes through the major phases of FASIT processing.

Table I -- Stages of FASIT Processing

| Input | Tagging | Disambiguation | Selected Concepts |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Most | AP | | |
| of | OF | | |
| these | DTS | | |
| homocides | NNS VBZ | NNS | homocides |
| have | HV | | |
| been | BEN | | |
| related | VBD VBN | VBD | |
| to | RI TO | | |
| the | AT | | |
| city's | NN$ | | |
| burgeoning | VBG | | |
| drug | NN VB | NN | drug |
| trade | NN VB | NN | trade |

In previously reported work on FASIT (Dillon and Gray 1983, Dillon and McDonald 1983), additional processing was done on the extracted concepts. Nouns and adjectives which might be judged not to be indicative of the document's content, such as "current account" or "little flexibility", and might therefore be expected to reduce precision, were filtered out. Secondly, the earlier work reported an algorithm which might increase precision by performing a rudimentary degree of phrase clustering. Phrases such as "life insurance" and "life insurance policy" were collected to form a type of thesaurus entry because they share many stems. Both improvements were eliminated from the present study because of our interest in establishing a baseline performance for FASIT. However, both are active areas of research.

## Data Preparation

All data is processed automatically; there is no hand-guiding. Queries are in the narrative-concept format.

Queries and documents are represented as FASIT output. In addition, the component words of the phrases identified by FASIT are represented as terms. All words and phrases are stemmed.

Because we are Plan B participants, we are working with the 350M subset of the Wall Street Journal data. A SMART database with ATC weighting was built from the FASIT output and submitted to the TREC sponsors for evaluation.

We have also performed extensive testing and failure analysis of a 46449-record subset of the TREC training database which consists of all Wall Street Journal articles from 1987. We chose this subset because all of the documents relevant to the training queries for the Plan B participants were drawn from the 1987 subset. A baseline database for these experiments was created using SMART, ATC weighting and stem indexing.

## Results

The results reported here are from the 1987 subset of the Wall Street Journal database because the results of the full database were not available at the time of this writing. Table II shows precision scores for eleven levels of recall in the baseline database; Table III shows results for a test database constructed with FASIT processing of queries and documents.

Table II

Precision Results for Eleven Levels of Recall with Stem Indexing

```
Num_queries:          25
Total number of documents over all queries
    Retrieved:      5000
    Relevant:        245
    Rel_ret:         166
    Trunc_ret:      4432
Recall - Precision Averages:
    at 0.00        0.3590
    at 0.10        0.3031
    at 0.20        0.2118
    at 0.30        0.1641
    at 0.40        0.1444
    at 0.50        0.1269
    at 0.60        0.0946
    at 0.70        0.0819
    at 0.80        0.0599
    at 0.90        0.0322
    at 1.00        0.0292
Average precision for all points
   11-pt Avg:      0.1461
Average precision for 3 intermediate points (0.20, 0.50, 0.80)
    3-pt Avg:      0.1329
```

Table III

```
---------------------------------------------
Precision Results for Eleven Levels of Recall
with FASIT Indexing
---------------------------------------------

Num_queries:          25
Total number of documents over all queries
    Retrieved:      5000
    Relevant:        245
    Rel_ret:         164
    Trunc_ret:      4382
Recall - Precision Averages:
    at 0.00        0.3867
    at 0.10        0.3092
    at 0.20        0.2369
    at 0.30        0.1918
    at 0.40        0.1449
    at 0.50        0.1034
    at 0.60        0.0816
    at 0.70        0.0666
    at 0.80        0.0501
    at 0.90        0.0458
    at 1.00        0.0337
Average precision for all points
   11-pt Avg:      0.1501
Average precision for 3 intermediate points (0.20, 0.50, 0.80)
    3-pt Avg:      0.1301
```

The higher precision scores at the lowest levels of recall are a replication of the experiments with FASIT reported in Dillon and McDonald (1983) and Burgin and Dillon (1992), and support the hypothesis that phrase indexing serves primarily as a precision enhancer. In the higher levels of recall, the effects of phrase indexing and stem indexing are essentially the same, indicating that retrieval performance is not jeopardized by the reduced document representation. We expect larger gains in precision with phrase indexing as we fine-tune our phrase-extraction algorithms.

References

Burgin and Dillon, M (1992). Improving Disambiguation in FASIT.

Journal of the American Society for Information Science, 43, 101-114.

Dillon and Gray, A.S. (1983). FASIT: A fully automatic syntactically based indexing system. Journal of the American Society for Information Science, 35, 3-10.

Dillon and McDonald, L.K. (1983). Fully automatic book indexing. Journal of Documentation, 39, 135-154.

Fagan, J.L. (1987). Experiments in automatic phrase indexing for document retrieval: A comparison of syntactic and non-syntactic methods. (Ph.D. dissertation, Cornell University.) Technical Report No. 87-868. Ithaca, NY: Cornell University.

Lewis, D. (1991). Representation and learning in information retrieval. (Ph.D. dissertation, University of Massachusetts, Amherst.) COINS Technical Report 91-93.

# A Single Language Evaluation of a Multi-lingual Text Retrieval System

*Ted Dunning*

*Mark Davis*

Computing Research Laboratory
New Mexico State University
Las Cruces, NM 88003

## 1. Introduction

The goal of the participation by the Computing Research Laboratory at New Mexico State University in the Text Retrieval Evaluation Conference was to evaluate our multi-lingual text retrieval system in a mono-lingual setting and in the context of a large set of documents. This system is currently being developed for use in a multi-lingual setting, but we felt that there were novel aspects to the system which should prove useful in retrieval from a large text base.

In particular, the system is able to find and make use of phrases in queries which will aid in retrieval. Marking which phrases are significant was done using a likelihood ratio test which is more reliable than previously used measures.

## 2. System Structure

Our Multi-Lingual Text-Retrieval system consisted of a fairly conventional system based around an inverted index structure. The positions of individual words are recorded in terms of which document they appeared in, and where in the document they appeared, both in terms of bytes, as well as in terms of words. In order, to maintain language independence as well as to improve modularity, tokenization during indexing can be done as a separate process. This modularity allows most of the system to remain unchanged when changing languages from English to Japanese, or when changing the indexing structure to support fast access to, say, a traditional dictionary of word definitions where only the various spelling forms of the head word might be indexed.

All postings in the final database are constant length structures in order to facilitate and simplify vector oriented operations. Document numbers and locations are uniformly stored as 4 byte integers in order to avoid for all practical purposes any limitations on the number or size of documents and tokens. Since full positional information is kept, it is relatively fast and simple

to search for phrases, or to find instances of words appearing near each other. We plan in the near future to alter one of the position indices to store sentence position instead of word position relative the beginning of a file. In other CRL systems, the ability to determine whether words appear in the same sentence has proved very useful.

## 3. Indexing Operations

Indexing for the Multi-Lingual Text-Retrieval system consists of a 4-pass process. The passes include tokenization, relabelling, sorting and stripping.

### 3.1. Tokenization

Tokenization in English consists of putting each word in the input text on a single line along with positional and document information. Kill list processing is done at this stage, as well, as creation of a word and document table. Lemmatization and suffix stripping can be done at this stage, but no savings result since this merely increases the average length of posting vectors during retrieval. Further, early lemmatization or suffix stripping makes it impossible to later avoid lemmatization or suffix stripping.

In Japanese, tokenization down to the level of strings of consecutive kanji, or katakana is supported as well as tokenization down to individual kanji and strings of katakana. Since strings of hiragana typically are used for inflection, they are ignored. This results in words being ignored, since there are words in Japanese (mostly verbs) which are written in hiragana. Using kanji strings results in indexing a large number of phrases, while indexing each kanji ideograph separately results in an increasing number of phrase searches later in the retrieval process with an accompanying drop in performance.

Preprocessing to mark and normalize company, place and person names and dates is best included into the tokenization process. We plan to incorporate elements of CRL's Tipster Extraction software into this program at some time to investigate the impact this has on performance, but we have not done so at the present time.

Indices for fielded data can be created by using a conventionalized word location such as 0. This approach allows all such data to be handled in a uniform manner.

### 3.2. Relabelling

The output of the tokenization process is still ASCII or some form of extended ASCII such as Unicode or JIS. The next step is the conversion to a uniform binary form which assists

in subseqent sorting. This step is also where word and document tables are created which allow bi-directional conversion between strings and word numbers, or file references and document numbers. The output of the relabelling consists of three files,

1) the string table, in which all strings (including file names) are kept.

2) the document table, in which the file names, starting offset and lengths of all documents is kept.

3) a vector of sortable posting structures.

In order to improve performance in cases where a large amount of data must be indexed, the first two passes are often integrated. This has several benefits which primarily include a radical decrease in the number of times the word and document tables must be reread, and the substitution of function call/return instead of Unix pipes for the transfer of data between the tokenization and relabeling passes.

### 3.3. Sorting

Next, these sortable posting are sorted using a version of MSB radix sort combined with a heap merge pass for very large files. Our radix sort uses a radix of 65536 and is able to sort at a rate of nearly 1MB per second. Total time for the radix sort is linear in the number of elements to be sorted, up to the limit of main memory. Merging then takes $\Theta(n \log m)$ where $m$ is the number of internal sorts needed. In practice, sorting is so fast that tokenization and relabeling are the bottleneck even when these first two passes are integrated.

The major disadvantage of this technology is that a considerable amount of temporary disk space is required. Ultimately, at least 100% overhead in temporary disk space is required. Furthermore, this overhead is needed even if all that is being done is to update the indices. We believe that paying a time penalty to avoid this overhead is probably very much warranted if a system is to be used on large files.

### 3.4. Stripping

The output of the sort phase consists of a very large vector of postings. In this vector, there are long runs in which the word number is constant. The stripping process consists of creating and index so that the boundaries of these long runs can be found, and deleting the word number from the postings. This change in structure results in about a 25-50% decrease in posting file size (depending on how much positional information is kept), and the index allows very fast access to the vector of postings for any particular word.

## 4. Primitive Document Access

The posting vector for any particular word can be retrieved from the database with a single procedure call. The posting vectors obtained in this way can be combined using a set of functions which combine perform the customary boolean operations, as well as functions which allow adjacent words to be found, as well as when specified words appear within a specified neighborhood of each other. In addition, the basic query routines include a procedure which, given a document number from within a posting, will read the contents of this document into memory and return the resulting string.

On top of this primitive layer, another level is built which supports scoring of documents based on a query. Given a vector of strings which represent query terms, procedures in this level return a sorted score vector whose contents contain documents and scores. All scoring is done based on inverse document frequency weighting. We plan to convert to a system based on Bayesian decision rules if time permits in this original contract.

## 5. Query Formulation

In the CRL TREC effort, conversion from topics to queries was done entirely automatically. The retrieval topics were reduced to lists of one, two, three and four word phrases. Each such phrase was included in the query if it met a statistical test based on generalized likelihood ratios. On the average, this resulted in about 80 terms being retained for the retrieval.

Had time permitted, we would have included a manual relevance feedback operation in the process of query formulation. We expect that this would have greatly improved the utility of the multi-word phrases.

## 6. Document Scoring

Retrieval was accomplished by using the `or_posting` procedure to compute a posting vector which contained references to all documents which potentially had a non-zero score. These documents were then scored using a conventional inverse document frequence weighting scheme and the results were sorted. Only the first 200 documents in the sorted document vector were printed.

In the submitted results, the fact that there was a very significant difference in average document length was partially compensated for by accumulating scores in quadrature rather than simply summing them. There are much better methods available to normalize for document length in a principled way.

## 7. Overall Results

Analysis of the results as returned by the automatic TREC scoring program shows that the CRL entry performed much better than might be expected, given the severe problems encountered in actually running the tests. Even though the system was not able to complete a clean index of the AP textbase in time, and even though the merging process was severely flawed, and even though no relevance feedback was employed, the system was able to perform on a creditable level.

Many of the last minute problems can be backed out of the results by examining the results for only the WSJ and Ziff textbases. The other primary databases are less interesting because:

1) The FR texts were significantly longer and thus caused severe problems in merging results.

2) The index of the AP textbase was not completed by the revised program in time.

3) Excluding the DOE textbase had little effect on the results (only three queries among the first 50 had relevant documents in the DOE textbase).

When these exclusions are made, and a composite recall-precision plot is made it can be seen that, on the average, our system provides a maximum precision of about 60%, a minimum precision of about 40% and a maximum recall of about 40%. Unfortunately, there is a wild variability so that in a strong sense, no single average is a terribly valid picture of the systems performance.

# The QA System

James Driscoll, Jennifer Lautenschlager, Mimi Zhao
Department of Computer Science
University of Central Florida
Orlando, Florida 32816 USA

**Abstract**

In the QA system, semantic information is combined with keywords to measure similarity between natural language queries and documents. A combination of keyword relevance and semantic relevance is achieved by treating keyword weights and semantic weights alike using the vector processing model as a basis for the combination. The approach is based on (1) the database concept of semantic modeling and (2) the linguistic concept of thematic roles. Semantic information is stored in a lexicon built manually using information found in Roget's Thesaurus.

**Keywords:** vector processing model, semantic data model, semantic lexicon, thematic roles, entity attributes.

## 1. Introduction

The QA system is based on the semantic approach to text search reported in [9]. The QA system accepts natural language queries against collections of documents. The system uses keywords as document identifiers in order to sort retrieved documents based on their similarity to a query. The system also imposes a semantic data model upon the "surface level" knowledge found in text (unstructured information from a database point of view).

The intent of the QA System has been to provide convenient access to information contained in the numerous and large public information documents maintained by Public Affairs at NASA Kennedy Space Center (KSC). During a launch at KSC, about a dozen NASA employees access these printed documents to answer media questions. The planned document storage for NASA KSC Public Affairs is around 300,000 pages (approximately 900 megabytes of disk storage).

Because of our environment, the performance of our system is measured by a count of the number of documents one must read in order to find an answer to a natural language question. Consequently, the traditional precision and recall measures for IR have not been used to measure the performance of the QA System.

We have had success using semantics to improve the ranking of documents when searching for an "answer" to a query in a document collection of size less than 50 megabytes. However, it is important to note that our success has been demonstrated only in a real-world situation where queries are the length of a sentence and documents are either a sentence or, at most, a paragraph [8,9].

Our reasons for participating in TREC have been to (1) learn how our semantic approach fares when traditional IR measures of performance are used, and (2) test our system on larger collections of documents. In this paper, we describe our system, the experiments we performed, our results, and failure analysis.

## 2. Overview of the QA System Modified for TREC

The QA System has been restricted to an IBM compatible PC platform running under the DOS 5.0 operating system and without the use of any other licensed commercial software such as a DOS extender. The DOS version of the QA System is available at nominal cost from [3]. About 2,000 hours of programming have been used to develop the current software which includes a pleasant user interface; just as many hours have been used testing the basic keyword operation of the system. In addition, approximately 1,000 hours have been used performing experiments involving the semantic aspect of the QA System. The SQL/DS relational data base system has been used to carry out some of these semantic experiments.

The QA System is implemented in C and uses B+ tree structures for the inverted files. We felt the speed of the system and its storage overhead was not efficient enough to appear reasonable for TREC, so we designed a separate system without a pleasant user interface which uses a hashing scheme to establish codes for strings. This was done to cut down on storage space and eliminate the use of B+ trees. Approximately 400 hours of programming and debugging effort was used to modify the system for the TREC experiments. We kept the DOS environment.

This work has been supported in part by NASA KSC Cooperative Agreement NCC 10-003 Project 2, Florida High Technology and Industry Council Grants 4940-11-28-721 and 4940-11-728, and DARPA Grant 4940-11-28-808.

For the TREC experiments, we used nine IBM PS/2 Model 95 computers. These were 50 MHz 486 computers, each with 8 megabytes of RAM and one 400 megabyte hard drive. Two of the machines had 16 megabytes of RAM and another one had two 400 megabyte hard drives. A 33 MHz 486 PC was used to distribute text to the nine IBM machines, and then collect information for merging and redistribution.

Our plan was to put each of the nine Vol. 1 and Vol. 2 document collections on a separate machine, determine the document frequency for each term $t_j$ on each machine, and then merge the results and determine the inverse document frequency for each term in the entire collection. Next, we would index and query each document collection separately, and finally merge the retrieval results.

The vector processing model is the basis for our approach. Terms used as document identifiers are keywords modified by various techniques such as stop lists, stemming, synonyms, and query reformulation.

The calculation of the weighting factor ($w$) for a term in a document is a combination of term frequency ($tf$), document frequency ($df$), and inverse document frequency ($idf$). The basic definitions are as follows:

$tf_{ij}$ = number of occurrences of term $t_j$ in document $D_i$

$df_j$ = number of documents in a collection which contain $t_j$

$idf_j$ = $\log(N/df_j)$, where $N$ = total number of documents

$w_{ij} = tf_{ij} * idf_{ij}$

When the system is used to query a collection of documents with $t$ terms, the system computes a vector $Q$ equal to $(w_{q1}, w_{q2}, \ldots, w_{qt})$ representing the weights for each term in the query. The retrieval of a document with vector $D_i$ equal to $(d_{i1}, d_{i2}, \ldots, d_{it})$ representing the weights of each term in the document is based on the value of a similarity measure between the query vector and the document vector. For the NIST experiments, we used the Jaccard similarity coefficient [7] to retrieve documents for the September deadline.

Jaccard Coefficient

$$sim(Q, D_i) = \frac{\sum_{j=1}^{t} w_{qj} d_{ij}}{\sum_{j=1}^{t} (d_{ij})^2 + \sum_{j=1}^{t} (w_{qj})^2 - \sum_{j=1}^{t} w_{qj} d_{ij}}$$

## 3. Semantic Approach

Although the basic IR approach has shown some success in regard to natural language queries, it ignores some valuable information. For instance, consider the following query:

How long does the payload crew go through training before a launch?

The typical IR system dismisses the following words in the query as useless or empty: "how", "does", "the", "through", "before", and "a". Some of these words contain valuable semantic information. The following list indicates some of the semantic information triggered by a few of these words:

how long → Duration, Time
through → Location/Space, Motion with Reference to Direction, Time
before → Location/Space, Time

The database concept of semantic modeling and the linguistic concept of thematic roles can help in this regard.

### 3.1 Semantic Modeling

Semantic modeling was an object of considerable database research in the late 1970's and early 1980's. A brief overview can be found in [2]. Essentially, the semantic modeling approach identified concepts useful in talking informally about the real world. These concepts included the two notions of entities (objects in the real world) and relationships among entities (actions in the real world). Both entities and relationships have properties.

The properties of entities are often called attributes. There are basic or surface level attributes for entities in the real world. Examples of surface level entity attributes are General Dimensions, Color, and Position. These properties are prevalent in natural language. For example, consider the phrase "large, black book on the table" which indicates the General Dimensions, Color, and Position of the book.

In linguistic research, the basic properties of relationships are discussed and called thematic roles. Thematic roles are also referred to in the literature as participant roles, semantic roles and case roles. Examples of thematic roles are Beneficiary and Time. Thematic roles are prevalent in natural language; they reveal how sentence phrases and clauses are semantically related to the verbs in a sentence. For example, consider the phrase "purchase for Mary on Wednesday" which indicates who benefited from a purchase (Beneficiary) and when a purchase occurred (Time).

The goal of our approach is to detect thematic information along with attribute information contained in natural language queries and documents. When the information is present, our system uses it to help find the most relevant document. In order to use this additional information, the basic underlying concept of text relevance as presented earlier needs to be modified. The major modifications include the addition of a lexicon with thematic and attribute information, and a modified computation of the similarity coefficient. We now discuss our semantic lexicon.

### 3.2 The Semantic Lexicon

Our system uses a thesaurus as a source of semantic categories (thematic and attribute information). For example, Roget's Thesaurus contains a hierarchy of word classes to relate word senses [6]. For our research, we have selected several classes from this hierarchy to be used for semantic categories. We have defined thirty-six semantic categories as shown in Figure 1.

| *Thematic Role Categories* |
|---|
| Accompaniment |
| Amount |
| Beneficiary |
| Cause |
| Condition |
| Comparison |
| Conveyance |
| Degree |
| Destination |
| Duration |
| Goal |
| Instrument |
| Location/Space |
| Manner |
| Means |
| Purpose |
| Range |
| Result |
| Source |
| Time |

| *Attribute Categories* |
|---|
| Color |
| External and Internal Dimensions |
| Form |
| Gender |
| General Dimensions |
| Linear Dimensions |
| Motion Conjoined with Force |
| Motion in General |
| Motion with Reference to Direction |
| Order |
| Physical Properties |
| Position |
| State |
| Temperature |
| Use |
| Variation |

Figure 1. Thirty-Six Semantic Categories.

Prior to TREC, there were 3,000 entries in the lexicon established by manual examination of roughly 6,000 of the most frequent words occurring in NASA KSC text. For TREC, we made 1,000 new entries by examination of 1,700 frequent words occurring in the training text and the 52 training topics. Since the 1911 edition of Roget's Thesaurus has become public domain, we also created software which automatically extracted approximately 20,000 lexicon entries. However, we did not have enough time to explore the use of these entries.

In order to explain the assignment of semantic categories to a given term using Roget's Thesaurus, consider the brief index quotation for the term "vapor":

| vapor | | |
|---|---|---|
| n. | fog | 404.2 |
| | fume | 401 |
| | illusion | 519.1 |
| | spirit | 4.3 |
| | steam | 328.10 |
| | thing imagined | 535.3 |
| v. | be bombastic | 601.6 |
| | bluster | 911.3 |
| | boast | 910.6 |
| | exhale | 310.23 |
| | talk nonsense | 547.5 |

The eleven different meanings of the term "vapor" are given in terms of a numerical category. We have developed a mapping of the numerical categories in Roget's Thesaurus to the thematic role and attribute categories given in Figure 1. In this example, "fog" and "fume" correspond to the attribute State; "steam" maps to the attribute Temperature; and "exhale" is a trigger for the attribute Motion with Reference to Direction. The remaining seven meanings associated with "vapor" do not trigger any thematic roles or attributes. Since there are eleven meanings associated with "vapor," we indicate in the lexicon a probability of 1/11 each time a category is triggered. Hence, a probability of 2/11 is assigned to State, 1/11 to Temperature, and 1/11 to Motion with Reference to Direction. This technique of calculating probabilities is being used as a simple alternative to a corpus analysis. It should be pointed out that we are still experimenting with other ways of calculating probabilities.

Figure 2 shows lexicon entries for prepositions as a sample of the lexicon used in our experiments. These entries are somewhat misleading. Most prepositions trigger too many semantic categories to be of real use. The prepositions "during" and "until" are examples of useful prepositions.

### 3.3 Extended Computation of the Similarity Measure

The probabilistic details of a semantic lexicon and the computation of semantic weights can be found in [9]. A detailed explanation of the manner in which we combine semantic weights and keyword weights can be found in [8].

Essentially we treat semantic categories like indexing terms, and the probabilities introduced by a semantic lexicon mean that the frequency of a category in a document becomes an expected frequency and the presence of a category in a document becomes a probability for the category being present. This means that the document frequency for a category becomes an expected document frequency, and this enables an inverse document frequency to be calculated for a category.

| | |
|---|---|
| after: | Time(3), None(3), External and Internal Dimensions(2), Motion with Respect to Direction, Order, Location/Space |
| above: | Amount(2), Location/Space, None(2), Linear Dimensions(2), Order, External and Internal Dimensions, Position |
| as: | Condition, Comparison, Manner, None(2), Amount |
| at: | Condition, Location/Space, Manner, Time, Position, External and Internal Dimensions, Duration |
| atop: | Location/Space, Position |
| before: | Location/Space, Time(4), External and Internal Dimensions, Motion with Respect to Direction, Order, None(2), Position |
| below: | Motion with Respect to Direction, None(2), Amount, State, Linear Dimensions, Location/Space, Position |
| between: | Comparison, Duration, External and Internal Dimensions, Location/Space, Position, Amount |
| by: | Amount, Conveyance, Location/Space, Time, Position, External and Internal Dimensions, Means, Order, Motion with Respect to Direction, Duration, None |
| during: | Duration, Time |
| except: | Condition, None(2), Order, Amount(2) |
| for: | Duration, Goal, Purpose, Variation, Destination, Beneficiary, Amount, Range |
| from: | Cause, Source, Time, Motion with Respect to Direction(2), Amount, Location/Space, Position, Instrument |
| in: | Instrument, Location/Space, Purpose, Time, Motion with Respect to Direction(5), External and Internal Dimensions(2), Position, Condition, Goal, Means |
| into: | Condition, Location/Space, Time, Motion with Respect to Direction, External and Internal Dimensions, Position, None |
| like: | Comparison(3), Amount(2), Condition, Manner, None(7) |
| of: | Cause, Location/Space, Source, None, Time, Duration, Position |
| on: | Condition, Conveyance, Location/Space, Source, Time, Linear Dimensions(2), Motion with Respect to Direction, General Dimensions, Position, Means, External and Internal Dimensions, Purpose |
| over: | Duration, Location/Space, Time(2), Order, Linear Dimensions(3), Amount(4), General Dimensions, Motion with Respect to Direction, External and Internal Dimensions, Position, Degree, Condition |
| per: | Means, Order(2) |
| through: | None, Order, Goal, Linear Dimensions, Means, Time, Location/Space, Position, Motion with Respect to Direction |
| to: | Accompaniment, Beneficiary, Condition, Degree, Location/Space, Purpose, Result, Time, General Dimensions, Position, Motion with Respect to Direction(2), Comparison |
| under: | Condition, Location/Space, Position, Order, Degree, None, Amount, Linear Dimensions(3) |
| until: | Condition, Time |
| upon: | Condition, Conveyance, Source, Time, None, General Dimensions, Linear Dimensions, Means, External and Internal Dimensions, Motion with Respect to Direction, Purpose |
| with: | Accompaniment(2), Comparison, Manner, Result, Amount(3), Means, None, Order, Location/Space, Position, Time |
| within: | Range, External and Internal Dimensions(2), Time, Degree, Position, Location/Space |
| without: | External and Internal Dimensions, Order, Amount, None(2) |

Figure 2. Prepositions and Their Semantic Categories
(Reprinted by Permission of [3]).

For TREC semantic experiments performed after the September deadline (refer to Section 5.2), we treated semantic categories like keywords and used the following normalized similarity coefficient:

$$sim(Q,D_i) = \frac{\sum_{j=1}^{t+s} w_{qj} d_{ij}}{\sqrt{\sum_{j=1}^{t+s} (d_{ij})^2}}$$

where $s = 36$ is the number of semantic categories. It should be pointed out that we are still performing experiments to determine a proper blend of keyword and semantic information.

## 4. System Details

The following is an overview of the system we used to perform the TREC experiments. As pointed out earlier, many hours of programming and debugging effort were used to create a system for the TREC experiments.

### 4.1 The Scanning Process

Under the QA scanning procedure, scanning a document for indexing terms is a three-step process:

A. A token is scanned.

B. The token is analyzed. It is compared to a list of 166 stopwords, and these words (along with any numbers) are later discarded. Dates are transformed into a generic format, to allow for the matching of dates in a variety of different formats. Non-valid hyphenated words are separated into multiple tokens. Words that can be abbreviated (as determined by a list of 122 abbreviations) are replaced by their abbreviated form. Acronyms of multiple words are detected, and replaced by their respective acronyms.

C. The remainder of the words are stemmed according to a modified version of the J. B. Lovins stemming algorithm [5]. In some cases, prefixes are also removed.

The scanning procedure for the TREC experiments is a modification of the QA scanning procedure, in which only the words found in the text fields of the documents are tokenized. In order to speed up text processing, the amount of analyzing in step B is reduced. Dates are no longer transformed, and abbreviations and acronyms are not created. The stemming algorithm and removal of prefixes in step C remains unchanged.

For the TREC scanning procedure, an additional step is added. During this extra step, a hashing algorithm is used to assign each indexing term a unique 32-bit integer value, which we call a stem code.

### 4.2 Data Files

After a document is processed by the TREC version of the QA System, four primary data files are created. These are the document weight file, the inverted index file, the inverted data file, and the document name file.

The document weight file is a binary file containing a list of floating point numbers. These numbers are ordered sequentially by document number, and represent the summation of the weight in the query squared for a particular document's keywords.

The inverted index file is ordered by stem code. Each code has two file pointers, one pointing to the first block of data in the data file, and the second pointing to the last block of data. The inverted data file then consists of blocks of data, containing pairs of document numbers that the code is found in, and the code's frequency within that document. The blocks are linked together to form a list.

The document name file consists of a list of pointers into the original text file.

For Vol. 1, both the document weight file and the inverted index file were two megabytes. The inverted data file was approximately 385 megabytes, and the document name file was two megabytes.

### 4.3 Basic Procedure

For the TREC experiments, we did the following:

Step 1: This step involves matching every legitimate stem in the document collection with a unique integer value. This is done with a linear hashing function. A table containing this mapping, along with the number of documents each code is found in, is temporarily saved for use in Step 2.

Step 2: This step creates the four data files described above. The entire database is scanned, with the four files being created on the fly. Once this is accomplished, the table from Step 1 is no longer necessary, and is discarded.

Step 3: Relevant documents for each query are selected using the Jaccard similarity coefficient. The top 200 documents for each query are then determined.

The above three steps were followed to create the results for the September deadline. In the next section we present our "official" experiments and results, and some "unofficial" experiments and results.

## 5. Experiments and Results

Our experiments were intended to be Category A experiments with two results submitted for each ranking task. One ranking result would be for just keywords, the other ranking result would be for keywords combined with semantics. All query construction was automatic, and the treatment of ad-hoc routing queries was identical.

We also performed experiments concerning the expected behavior of string hashing functions and the use of part-of-speech tagging to improve retrieval performance. These experiments are not reported here.

As an example of our automatically built ad-hoc and routing queries, consider Topic 004 reproduced in Figure 3. Figure 4 indicates the keyword and semantic information generated by the QA System for this topic. The first part of Figure 4 indicates the stems along with their frequencies found in the query. The second part of Figure 4 indicates the semantic categories also found in the query along with their expected frequencies and probability present.

It is important to note that the topic represented in Figure 3 and Figure 4 has generated many semantic categories and the probability present for most of them is close to, or at, 100%. This is mainly due to the length of the text involved. We discovered that, for the TREC document collection, each document generated many semantic categories with high probability present. Because we treated semantic categories like keywords, this caused semantic weights to be essentially useless.

For the TREC September deadline, we were only able to submit a routing experiment using a keywording approach. The results of this experiment, computed with the aid of Chris Buckley's SMART evaluation program [1], are shown in Figure 5. The results are not good. In Section 6, we discuss what impeded our experiments.

Further "unofficial" experiments were designed to test the use of semantics. The main goal of our experiments was to demonstrate that our original routing results could be improved through the use of semantic analysis. In order to do this, we made two modifications to our approach. The first change involved dividing the original TREC documents into paragraphs. The second change involved a semantic analysis when calculating the list of relevant documents.

Our experiments involved the use of only six routing queries (for topics 001, 002, 003, 007, 017, and 022). These topics were selected because our original results for them were poor. Through the use of semantic analysis, we hoped to significantly improve our results. Figure 6 shows the precision-recall statistics for the six "poor" queries using the retrieval results which created the statistics in Figure 5.

When analyzing our results, we computed all precision-recall tables through the use of Chris Buckley's SMART evaluation program [1]. The relevancy lists used were those produced before November 1 (the original qrels for the routing queries). We did not use the modified results that were distributed later for Query 017. This should not affect our results, though, because our experiments were aimed only at improving our precision-recall averages, and the relevancy results used were consistent from one experiment to the other.

### 5.1 Re-ranking of Documents

In an effort to demonstrate that semantics could affect retrieval in the TREC environment, we used the original QA System with a semantic lexicon containing TREC words as described in Section 3.2. We created a separate database for each query we considered, for a total of six databases. Each database contained only the documents that we originally judged in the top 250 for each query. Because of this, when we computed new relevancy lists we were simply rearranging the order of the same 250 documents, not bringing in new documents.

Figure 7 reveals the precision-recall statistics when originally retrieved documents for a query are used as a document collection and re-ranked by imposing the query again. There is a 25.8% increase when comparing the 11-pt average here to the 11-pt average of the originally retrieved text (Figure 6).

To determine the ranking of a particular document with paragraph divisions, we defined the similarity coefficient of a document to be equal to the highest coefficient associated with one of its corresponding paragraphs. The paragraph divisions were automatically constructed from the original text. The precision-recall statistics for paragraphs being used as documents are shown in Figure 8. There is an 18.8%

<top>

<head> Tipster Topic Description

<num> Number: 004

<dom> Domain: International Finance

<top> Topic: Debt Rescheduling

<desc> Description: Document will discuss a current debt rescheduling agreement between a developing country and one or more of its creditor(s).

<narr> Narrative:

A relevant document will discuss a current debt rescheduling agreement reached, proposed, or being negotiated between a debtor developing country and one or more of its creditors, commercial and/or official. It will identify the debtor country and the creditor(s), the repayment time period requested or granted, the monetary amount requested or covered by the accord, and the interest rate, proposed or set.

<con> Concept(s):

1. rescheduling agreement, accord, settlement, pact
2. bank debt, commercial debt, foreign debt, trade debt, medium-term debt, long-term debt
3. negotiations, debt talks
4. creditor banks, creditor countries/governments, Paris Club
5. debtor countries, developing countries
6. debt package
7. debt repayments
8. restructuring, rescheduling existing loans
9. lower interest-rate margin, easier terms, more lenient terms

<fac> Factor(s):

<nat> Nationality: Developing country

<time> Time: Current

</fac>

<def> Definition(s):

Debt Rescheduling - agreement between creditors and debtor to provide debt relief by altering the original payment terms of an existing debt. This is most often accomplished by lengthening the original schedule for principal and interest payments, and deferring interest payments. Done most publicly by developing countries and their bankers, but often less publicly by other willing creditors and debtors, e.g., governments, banks and companies. Much in vogue in the early 1980s, the road to rescheduling for countries in crisis runs as follows: when a country borrows so much that its lenders grow nervous, the banks start lending for shorter and shorter maturities. Eventually the country, though still paying interest on its debt, is unable to make payments on the principal. The country is then forced to request a rescheduling, which means that it is able to escape its immediate repayment commitments by converting short-term loans into longer-term ones. A country wishing to reschedule its official debt tal

</top>

Figure 3. Topic 004.

| Stems | Frequency | Stems | Frequency |
|---|---|---|---|
| tal | 1 | wish | 1 |
| on | 1 | longer | 1 |
| short | 1 | convers | 1 |
| commitm | 1 | immedi | 1 |
| escap | 1 | abl | 1 |
| mean | 1 | forc | 1 |
| un | 1 | pay | 1 |
| though | 1 | eventu | 1 |
| matur | 1 | shorte | 2 |
| lend | 1 | start | 1 |
| nerv | 1 | grow | 1 |
| lender | 1 | borrow | 1 |
| follow | 1 | run | 1 |
| cris | 1 | road | 1 |
| ear | 1 | vogu | 1 |
| compan | 1 | e.g. | 1 |
| wil | 1 | less | 1 |
| banker | 1 | public | 2 |
| defer | 1 | princip | 2 |
| length | 1 | accompl | 1 |
| origin | 2 | alter | 1 |
| relief | 1 | prov | 1 |
| leni | 1 | easxxx | 1 |
| margin | 1 | lower | 1 |
| loan | 2 | exist | 2 |
| structur | 1 | pack | 1 |
| club | 1 | par | 1 |
| governm | 2 | talk | 1 |
| long-term | 1 | term | 6 |
| med | 1 | trad | 1 |
| foreign | 1 | bank | 4 |
| pact | 1 | settl | 1 |
| rat | 2 | interest | 5 |
| accord | 2 | cover | 1 |
| amount | 1 | monet | 1 |
| grant | 1 | request | 3 |
| period | 1 | paym | 7 |
| identif | 1 | offic | 2 |
| commerc | 2 | negoti | 2 |
| propos | 2 | reach | 1 |
| relev | 1 | credit | 7 |
| countr | 13 | develop | 5 |
| agre | 4 | cur | 3 |
| discus | 2 | docum | 2 |
| schedl | 10 | debt | 22 |
| fin | 1 | intern | 1 |

| Semantics | Expected Frequency | Probability Present |
|---|---|---|
| Linear Dimensions | 0.300000 | 0.300000 |
| Motion with Reference to Direction | 2.000000 | 1.000000 |
| Accompaniment | 0.333333 | 0.333333 |
| Condition | 2.000000 | 0.937500 |
| External and Internal Dimensions | 0.333333 | 0.333333 |
| Time | 5.744589 | 1.000000 |
| Duration | 1.291667 | 0.819444 |
| Purpose | 2.000000 | 1.000000 |
| Color | 2.333333 | 0.941472 |
| Position | 6.071428 | 0.999654 |
| Location/Space | 6.071428 | 0.999654 |
| Variation | 5.933333 | 1.000000 |
| Amount | 53.412807 | 1.000000 |
| Order | 6.833333 | 1.000000 |

Figure 4. Automatically Generated Query for Topic 004.

Top ranked evaluation

| | | |
|---|---|---|
| Run number: | UCFQA1 | (category B) |
| Num_queries: | 19 | |
| Total number of documents over all queries | | |
| Retrieved: | 3800 | |
| Relevant: | 2056 | |
| Rel_ret: | 517 | |
| Recall - Precision Averages: | | |
| at 0.00 | 0.5146 | |
| at 0.10 | 0.2645 | |
| at 0.20 | 0.1735 | |
| at 0.30 | 0.1006 | |
| at 0.40 | 0.0693 | |
| at 0.50 | 0.0107 | |
| at 0.60 | 0.0069 | |
| at 0.70 | 0.0000 | |
| at 0.80 | 0.0000 | |
| at 0.90 | 0.0000 | |
| at 1.00 | 0.0000 | |
| Average precision for all points | | |
| 11-pt Avg: | 0.1037 | |
| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | | |
| 3-pt Avg: | 0.0614 | |
| Recall: | | |
| at 5 docs: | 0.0162 | |
| at 15 docs: | 0.0362 | |
| at 30 docs: | 0.0692 | |
| at 100 docs: | 0.2156 | |
| at 200 docs: | 0.2899 | |
| Precision: | | |
| at 5 docs: | 0.2737 | |
| at 15 docs: | 0.2211 | |
| at 30 docs: | 0.2088 | |
| at 100 docs: | 0.1932 | |
| at 200 docs: | 0.1361 | |

Figure 5. Results for Experiment Completed by September Deadline.

Top ranked evaluation

| | |
|---|---|
| Run number: | 1 |
| Num_queries: | 6 |
| Total number of documents over all queries | |
| Retrieved: | 1200 |
| Relevant: | 863 |
| Rel_ret: | 147 |
| Recall - Precision Averages: | |
| at 0.00 | 0.3486 |
| at 0.10 | 0.1676 |
| at 0.20 | 0.0749 |
| at 0.30 | 0.0279 |
| at 0.40 | 0.0000 |
| at 0.50 | 0.0000 |
| at 0.60 | 0.0000 |
| at 0.70 | 0.0000 |
| at 0.80 | 0.0000 |
| at 0.90 | 0.0000 |
| at 1.00 | 0.0000 |
| Average precision for all points | |
| 11-pt Avg: | 0.0563 |
| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
| 3-pt Avg: | 0.0250 |
| Recall: | |
| Exact: | 0.1997 |
| at 5 docs: | 0.0033 |
| at 10 docs: | 0.0062 |
| at 30 docs: | 0.0382 |
| at 100 docs: | 0.1212 |
| at 200 docs: | 0.1997 |
| Precision: | |
| Exact: | 0.1225 |
| at 5 docs: | 0.1000 |
| at 10 docs: | 0.0833 |
| at 30 docs: | 0.1556 |
| at 100 docs: | 0.1550 |
| at 200 docs: | 0.1225 |

Figure 6. Results for Six Poorly Performing Queries using Experiment Completed by September Deadline.

Top ranked evaluation

| | |
|---|---|
| Run number: | 1 |
| Num_queries: | 6 |
| Total number of documents over all queries | |
| Retrieved: | 1200 |
| Relevant: | 863 |
| Rel_ret: | 152 |
| Recall - Precision Averages: | |
| at 0.00 | 0.4585 |
| at 0.10 | 0.2113 |
| at 0.20 | 0.0775 |
| at 0.30 | 0.0315 |
| at 0.40 | 0.0000 |
| at 0.50 | 0.0000 |
| at 0.60 | 0.0000 |
| at 0.70 | 0.0000 |
| at 0.80 | 0.0000 |
| at 0.90 | 0.0000 |
| at 1.00 | 0.0000 |
| Average precision for all points | |
| 11-pt Avg: | 0.0708 |
| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
| 3-pt Avg: | 0.0258 |
| Recall: | |
| Exact: | 0.2109 |
| at 5 docs: | 0.0072 |
| at 10 docs: | 0.0128 |
| at 30 docs: | 0.0469 |
| at 100 docs: | 0.1374 |
| at 200 docs: | 0.2109 |
| Precision: | |
| Exact: | 0.1267 |
| at 5 docs: | 0.2000 |
| at 10 docs: | 0.1833 |
| at 30 docs: | 0.1944 |
| at 100 docs: | 0.1683 |
| at 200 docs: | 0.1267 |

Figure 7. Results of Keywording with Document Divisions.

Top ranked evaluation

| | |
|---|---|
| Run number: | 1 |
| Num_queries: | 6 |
| Total number of documents over all queries | |
| Retrieved: | 1200 |
| Relevant: | 863 |
| Rel_ret: | 157 |
| Recall - Precision Averages: | |
| at 0.00 | 0.4096 |
| at 0.10 | 0.2089 |
| at 0.20 | 0.0832 |
| at 0.30 | 0.0340 |
| at 0.40 | 0.0000 |
| at 0.50 | 0.0000 |
| at 0.60 | 0.0000 |
| at 0.70 | 0.0000 |
| at 0.80 | 0.0000 |
| at 0.90 | 0.0000 |
| at 1.00 | 0.0000 |
| Average precision for all points | |
| 11-pt Avg: | 0.0669 |
| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
| 3-pt Avg: | 0.0277 |
| Recall: | |
| Exact: | 0.2167 |
| at 5 docs: | 0.0121 |
| at 10 docs: | 0.0167 |
| at 30 docs: | 0.0462 |
| at 100 docs: | 0.1392 |
| at 200 docs: | 0.2167 |
| Precision: | |
| Exact: | 0.1308 |
| at 5 docs: | 0.2667 |
| at 10 docs: | 0.1833 |
| at 30 docs: | 0.1833 |
| at 100 docs: | 0.1650 |
| at 200 docs: | 0.1308 |

Figure 8. Results of Keywording with Paragraph Divisions.

increase when comparing the 11-pt average here to the 11-pt average of the originally retrieved text (Figure 6).

The re-ranking of documents using keywords was done in order to establish a baseline for the semantic experiments reported in the next subsection. The increase in retrieval performance because of the re-ranking was a surprise. According to Sparck Jones' criteria, the increases of 25.8% and 18.8% would each be categorized as "significant" (greater than 10.0%) [4].

### 5.2 Semantic Experiments

Essentially, what we are trying to show in this section is that semantics can be useful if documents do not get larger than a paragraph.

Figure 9 displays results when semantic similarity is combined with keyword similarity for the case that originally retrieved TREC documents are used. These results can be compared to those in Figure 7 (the same documents with a strictly keywording approach). Comparing the 11-pt average for the two methods shows a 0.7% decrease when going from the strictly keywording results to the combined semantic with keywording results. According to Sparck Jones' criteria [4], this is not a noticeable decrease. But, certainly, semantics did not help.

The explanation for this is that when the size of a document is large, a greater number of semantic categories are triggered in the document. Also, the probability present for each category in a document is often very close to 100%. Consequently, almost every semantic category becomes present in every document causing the semantic category weights to become very low and useless.

To remedy this problem, we used the database that was constructed with paragraph divisions, and computed relevancy lists using the combined semantic and keywording approach explained in Section 3.3. The results are shown in Figure 10. The statistics there can be compared to those in Figure 8 (the same documents with a strictly keywording approach). When going from a keywording approach to a combined semantic and keywording approach, the 11-pt average increased by 7.9%. According to Sparck Jones' criteria, this change would be classified as "noticeable" (greater than 5.0%) [4].

The two semantic experiments reported here demonstrate the main thing that we have learned. Our semantic approach to text retrieval is only useful when documents are no larger than a paragraph.

Top ranked evaluation

| | |
|---|---|
| Run number: | 1 |
| Num_queries: | 6 |
| Total number of documents over all queries | |
| Retrieved: | 1200 |
| Relevant: | 863 |
| Rel_ret: | 155 |
| Recall - Precision Averages: | |
| at 0.00 | 0.4370 |
| at 0.10 | 0.2250 |
| at 0.20 | 0.0789 |
| at 0.30 | 0.0329 |
| at 0.40 | 0.0000 |
| at 0.50 | 0.0000 |
| at 0.60 | 0.0000 |
| at 0.70 | 0.0000 |
| at 0.80 | 0.0000 |
| at 0.90 | 0.0000 |
| at 1.00 | 0.0000 |
| Average precision for all points | |
| 11-pt Avg: | 0.0703 |
| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
| 3-pt Avg: | 0.0263 |
| Recall: | |
| Exact: | 0.2133 |
| at 5 docs: | 0.0072 |
| at 10 docs: | 0.0128 |
| at 30 docs: | 0.0508 |
| at 100 docs: | 0.1378 |
| at 200 docs: | 0.2133 |
| Precision: | |
| Exact: | 0.1292 |
| at 5 docs: | 0.2000 |
| at 10 docs: | 0.1833 |
| at 30 docs: | 0.2000 |
| at 100 docs: | 0.1700 |
| at 200 docs: | 0.1292 |

Figure 9. Results of Semantics with Document Divisions.

Top ranked evaluation

| | |
|---|---|
| Run number: | 1 |
| Num_queries: | 6 |
| Total number of documents over all queries | |
| Retrieved: | 1176 |
| Relevant: | 863 |
| Rel_ret: | 158 |
| Recall - Precision Averages: | |
| at 0.00 | 0.4556 |
| at 0.10 | 0.2231 |
| at 0.20 | 0.0845 |
| at 0.30 | 0.0294 |
| at 0.40 | 0.0000 |
| at 0.50 | 0.0000 |
| at 0.60 | 0.0000 |
| at 0.70 | 0.0000 |
| at 0.80 | 0.0000 |
| at 0.90 | 0.0000 |
| at 1.00 | 0.0000 |
| Average precision for all points | |
| 11-pt Avg: | 0.0720 |
| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
| 3-pt Avg: | 0.0282 |
| Recall: | |
| Exact: | 0.2159 |
| at 5 docs: | 0.0096 |
| at 10 docs: | 0.0171 |
| at 30 docs: | 0.0513 |
| at 100 docs: | 0.1456 |
| at 200 docs: | 0.2159 |
| Precision: | |
| Exact: | 0.1332 |
| at 5 docs: | 0.2333 |
| at 10 docs: | 0.2000 |
| at 30 docs: | 0.2167 |
| at 100 docs: | 0.1767 |
| at 200 docs: | 0.1317 |

Figure 10. Results of Semantics with Paragraph Divisions.

It should be noted that the semantic experiments reported here were crude. Our lexicon did not have enough TREC words, and we used a blend of keyword and semantic weights that was not the best. So we expect that better semantic results for paragraphs can be achieved (refer to Section 6).

## 6. Failure Analysis

In general our participation in the TREC experiments was impeded by the following:

PC DOS Platform. This platform has a serious memory addressing restriction which results in memory page swapping and this seriously affects the speed of processing, especially during creation of inverted files and index structures. We can solve this problem by moving to an OS/2 or UNIX platform.

Extra Semantic Processing Time. Our semantic probabilistic and statistical calculations more than double the processing time for indexing and statistical ranking of retrieved documents. Again, we can solve this problem by moving to an OS/2 or UNIX platform.

Time to Build Semantic Lexicon. We were only able to incorporate 1000 frequently occurring words in the training text within our semantic lexicon. We did not have enough time to process the test text for the ad-hoc queries. This problem can be solved by having archival data distributed earlier. We suspect that by having more TREC words in our semantic lexicon, better results could have been achieved in Section 5.2 when paragraphs are used as the basis for retrieval.

Unknown Blend for Semantic and Keyword Weights. There are three main aspects to our blend of semantic and keyword weights within the vector processing model:

(i) The Proper Probabilities to Use for the Semantics Triggered by a Word. For example, we let the word "vapor" trigger State with 18% probability, Temperature with 9% probability, and Motion with Reference to Direction with 9% probability. We have several techniques for determining probabilities such as these.

(ii) The Scaling of Keyword Weights and Semantic Weights. For example, in a Question/Answer environment where queries are the length of a sentence and documents are either a sentence or at most a paragraph, we have been successful by forcing semantic similarity to be approximately 1/3 of keyword similarity when the two are combined in processing small document collections (less than 1000 documents). There was no scaling for the experiments reported in Section 5.2; we suspect better results could have been achieved.

(iii) Independent Semantic Weights and Keyword Weights. A valid criticism of our research has been that the semantic contribution from a word in a document should be kept independent of the word's own similarity contribution if the word is a keyword in common with the query.

The overall problem of proper blend can be solved by spending more time using TREC test documents, test topics, and good test relevance judgments to run many retrieval experiments to establish the correct blend.

Number of Semantic Categories. Another way to solve the problem of long documents causing semantic weights to be of little value is to have more semantic categories. A large number of "semantic" categories could be obtained (for example) by using all the categories and/or subcategories found in Roget's Thesaurus, instead of the 36 semantic categories we use. This would be a deviation from database semantic modeling but it probably should be examined.

Block-split Tree Structured Files. The QA System used B+ tree structures for implementing inverted files and this actually slowed the system in our DOS environment. The QA System also had severe storage overhead due to storage of character strings in the B+ trees. We have solved these two problems by implementing a separate system using a hashing function to establish codes for strings.

TREC Document Length. Semantic experiments like those reported in Section 5.2 have shown that documents larger than a paragraph cause our semantic approach to be of little value. This problem can be corrected by considering paragraphs as a basis for document retrieval.

Finally, we spent too much time on work that was never incorporated in our experiments. We originally designed an efficient method of inverting data files, but it could not be used for routing queries. Also, trying to do semantic part-of-speech tagging experiments using SQL/DS slowed us down.

## References


[1] C. Buckley, SMART Evaluation Program (for TREC), *Cornell SMART Group*, Cornell University.

[2] C. Date, *An Introduction to Database Systems*, Vol. I, Addison Wesley, 1990.

[3] Hello Software, P. O. Box 494, Goldenrod, FL 32733.

[4] K. Sparck Jones and R. Bates, "Research on Automatic Indexing 1974-1976," *Technical Report*, Computer Laboratory, University of Cambridge, 1977.

[5] J. Lovins, "Development of a Stemming Algorithm," *Mechanical Translation and Computational Linguistics*, Vol. 11, No. 1-2, pp. 11-31, March and June, 1968.

[6] *Roget's International Thesaurus*, Harper & Row, New York, Fourth Edition, 1977.

[7] G. Salton, *Automatic Text Processing*, Addison-Wesley, 1989.

[8] D. Voss and J. Driscoll, "Text Retrieval Using a Comprehensive Semantic Lexicon," *Proceedings of ISMM First International Conference on Information and Knowledge Management*, Baltimore, Maryland, November 1992.

[9] E. Wendlandt and J. Driscoll, "Incorporating a Semantic Analysis into a Document Retrieval Strategy," *Proceedings of the Fourteenth Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval*, Chicago, Illinois, pp. 270-279, October 1991.

# CLASSIFICATION TREES FOR DOCUMENT ROUTING

## A Report on the TREC Experiment

*Richard Tong, Adam Winkler, Pamela Gage*
Advanced Decision Systems
(a division of Booz•Allen & Hamilton)
1500 Plymouth Street, Mountain View, CA 94043

### 1. Introduction

In this paper we describe an approach to document routing on the TREC corpus that employs a technique for the automatic construction of classification trees. The approach makes use of the Classification and Regression Trees (CART) algorithm that has seen application in various areas of machine learning [1,2].

Our initial work with this algorithm [3] has demonstrated that probabilistic structures can be automatically acquired from a training set of documents with respect to a single target concept, or a set of related concepts. These structures can then be applied to individual documents to derive a posterior probability that the document is about a particular target concept. In addition, CART provides the user with a mechanism for explicitly controlling the precision and recall performance trade-offs.

In the CART approach, the task is to provide a direct assessment of the probability that a given concept is in a document given all possible combinations of evidence. Classification trees are used as the probabilistic structure for representing this direct assessment. When used to perform the document routing task, the CART algorithm takes as input a collection of example documents (the training set), each of which consists of a known class assignment and a vector of features (e.g., whether or not the document is about the topic of interest, together with a list of the feature words found in the document). In the general case, the feature values may be boolean, integer, real or nominal and, in addition, may be both noisy and incomplete. The output from CART is a binary decision tree that can be used to classify subsequent documents for which the class is not known. When the training set is noisy, trees of maximal size are invariably too large in that a smaller tree will perform better in terms of classification accuracy. This is the standard statistical "overfitting" problem. CART addresses the overfitting problem via tree pruning and error estimation algorithms that locate the "right sized" tree, ensuring a parsimonious, accurate classifier.

The remainder of the paper is divided into a number of sections. In Section 2 we briefly describe the processing steps performed by our system, first to generate the optimal classification tree and then to use the tree for classification. In Section 3 we illustrate how the TREC data is processed. In Section 4 we discuss our "official" TREC results.

Then finally in Section 5 we offer some observations on the overall value of using CART as the basis of a document routing system.

## 2. The CART Algorithm

CART has been shown to be useful when one has access to datasets describing known classes of observations, and wishes to obtain rules for classifying future observations of unknown class—exactly as in the document routing problem. CART is particularly attractive when the dataset is "messy" (i.e., is noisy and has many missing values) and thus unsuitable for use with more traditional classification techniques. In addition, and particularly important for the document routing application, if it is important to be able to specify both the misclassification costs and the prior probabilities of class membership then CART has a direct way of incorporating such information into the tree building process. Finally, CART can generate auxiliary information, such as the expected misclassification rate for the classifier as a whole and for each terminal node in the tree, that is useful for the document routing problem.

Figure 1 shows how the CART algorithm is used first to construct the optimal classification tree and then to generate a classification decision. The upper part of the diagram

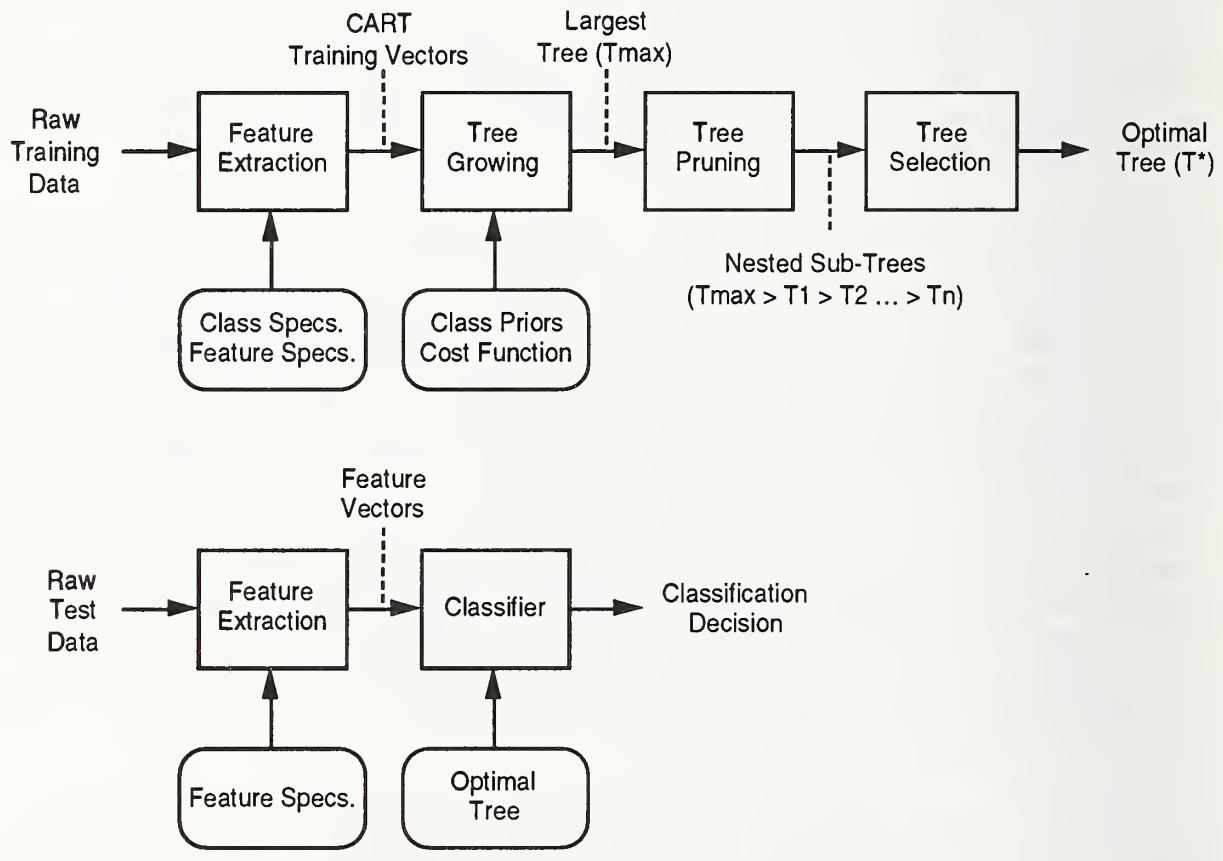



Figure 1: CART Processing Flow

shows the four sub-processes used to generate the optimal tree, T*. The "raw" training data (i.e., the original texts of the articles), together with the class specifications (i.e., the training data relevance judgments) and the feature specifications (i.e., the words defined to be features), are input to the Feature Extraction Module. The output is a set of vectors that record the class membership and the features contained in the training data. These vectors, together with class priors and the cost function (these are optional), are input to the Tree Growing Module which then constructs the maximal tree ($T_{max}$) that characterizes the training data. Since this tree overfits the data, the next step is to construct a series of nested sub-trees by pruning $T_{max}$ to the root. The sequence of sub-trees ($T_{max} > T_1 > \ldots > T_n$) are input to the Tree Selection Module which performs a cross-validation analysis on the sequence and selects that tree with the lowest cross-validation error[1]. This is T*.

The lower part of the diagram shows how T* is used to classify documents of unknown class. As in tree building, the "raw" test data (i.e., the unprocessed texts of the test documents) are passed, together with the feature specifications, to the Feature Extraction Module, which in turn generates a feature vector for each text. These vectors are passed to the Classifier Module which uses the optimal tree to make the classification decision (i.e., whether the document is relevant or not).

### 3. Working with the TREC Corpus

Since the application of the CART algorithm to document routing is a relatively unexplored area of research (see [3] for some of our preliminary results with a small test corpus), we chose to participate in TREC as a Category B group. That is, we worked only with the Wall Street Journal articles and used only the first 25 query topics. In addition, since CART is intended to be used with training data, it most readily lends itself to the document routing problem. Thus we did not address the *ad hoc* retrieval problem in our experiments.

Our primary goal in working with the TREC data was to develop a totally automatic approach to generating document classification trees, working only with the information need statements and the training data provided. That is, we wanted to test the hypothesis that CART would be a valuable tool in situations where the user can formulate an information need in some detail and can provide examples of documents that are relevant and examples of those that are not. Such a scenario is common in organizations that monitor large volumes of real-time electronic documents.

To illustrate how we used the information need statements, consider the following topic description:

---

1. The algorithm actually minimizes with respect to both the cross-validation error and the tree complexity. So that if two trees have statistically indistinguishable error rates, then the smaller of the two trees will be selected as optimal.

```
<top>
<head> Tipster Topic Description
<num> Number: 001
<dom> Domain: International Economics
<title> Topic: Antitrust Cases Pending
<desc> Description:
Document discusses a pending antitrust case.
<narr> Narrative:
To be relevant, a document will discuss a pending antitrust case
and will identify the alleged violation as well as the government
entity investigating the case. Identification of the industry and
the companies involved is optional. The antitrust investigation
must be a result of a complaint, NOT as part of a routine review.
<con> Concept(s):
1.  antitrust suit, antitrust objections, antitrust investigation,
antitrust dispute
2.  monopoly, bid-rigging, illegal restraint of trade, insider
trading, price-fixing
3.  acquisition, merger, takeover, buyout
4.  Federal Trade Commission (FTC), Interstate Commerce Commission
(ICC), Justice Department, U.S. Securities and Exchange Commission
(SEC), Japanese Fair Trade Commission
5.  NOT antitrust immunity
<fac> Factor(s):
<def> Definition(s):
Antitrust - Laws to protect trade and commerce from unlawful
restraints and monopolies or unfair business practices.
Acquisition - The taking over by one company of a co trolling
interest in another, also called a takeover. The action may be
friendly or unfriendly.
Merger - The acquisition by one corporation of the stock of
another. The acquiring corporation then retires the other's stock
and dissolves that corporation. Therefore, only one corporation
retains its identity in a merger.
</top>
```

Since this is a very comprehensive description it contains many topic-specific words. Recognizing this, our approach is to:

- extract the `<desc>`, `<narr>`, `<con>`, and `<def>` fields from the topic specification,
- concatenate them, removing the SGML-style tags and the field labels (i.e., the `Description:`, `Narrative:`, `Concept(s):`, and `Definition(s):` strings),
- map all the words into lowercase, remove duplicates and stop words[2] from the resulting description, and
- use the remaining list of words as the set of features.

2. We used a stop word list published by Fox [4] that contains 421 words.

Note that we do no further processing of the topic text. We do no phrasal analysis, no stemming, no term expansion and, in particular, we do nothing with the NOT-clauses.

Applying this procedure to the topic above results in the following list of feature words:

```
acquisition: antitrust: commerce: commission: department:
exchange: ftc: fair: federal: icc: identification: interstate:
japanese: justice: laws: merger: sec: securities: trade: acquir-
ing: action: alleged: bid: business: buyout: called: companies:
company: complaint: controlling: corporation: discuss: dispute:
dissolves: document: entity: fixing: friendly: government: iden-
tify: identity: illegal: immunity: industry: insider: investigat-
ing: investigation: involved: merger: monopolies: monopoly:
objections: optional: pending: practices: price: protect: rele-
vant: restraint: restraints: result: retains: retires: review:
rigging: routine: stock: suit: takeover: taking: trading: unfair:
unfriendly: unlawful:
```

This feature specification is used as input to the Feature Extraction module (see upper part of Figure 1) along with the original training data and the relevance judgements associated with the topic.

A key characteristic of the training data provided for TREC is that there were very few relevance judgments for each topic. Thus we were only able to generate a limited number of training vectors to be used by the CART algorithm. For each document in the training set for which there is a relevance judgement, we determine the number of occurrences of each feature (using only the <text> fields). Thus for each topic we get a set of training vectors that looks like those shown below (augmented here, for presentation purposes, with the topic and document ids):

```
001 Q0 WSJ870320-0188 (0 (0,1,0,5,0,0,3,…,6))
001 Q0 WSJ870320-0069 (0 (2,0,9,1,0,4,1,…,0))
…
001 Q0 WSJ870424-0084 (1 (1,3,0,5,2,0,0,…,2))
```

Where in each vector the first element indicates whether the document is relevant to the topic (0 mean no, 1 mean yes) and the remainder of the elements represent counts of the occurrence of the features in the document. Thus document WSJ870320-0188 is not-relevant to Topic 1 and contains zero occurrences of the first feature, one occurrence of the second feature, etc.

This training data is input to the Tree Growing module (see Figure 1) along with the cost function and class priors[3]. CART then grows the largest tree, prunes this into a set of nested sub-trees and uses cross validation to suggest the optimal tree. For the example

3. In our "official" TREC experiments we set the priors to be P(rel)=0.01 and P(non-rel)=0.99, and the ratio of the cost of misclassifying a relevant document to the cost of misclassifying a non-relevant document to be 100:1. That is, we assumed that there are relatively few relevant documents in the collection and that we want to emphasize recall significantly over precision.

above, the optimal tree is:

```
          class 0 (0.000)
company<=0.50
                                    class 1 (0.981)
                               federal<=9.00
                                    class 0 (0.000)
                          takeover<=1.50
                               class 0 (0.000)
                     securities<=1.50
                          class 0 (0.000)
               company<=4.50
                    class 0 (0.000)
          fair<=0.50
               class 0 (0.000)
```

This is a the binary classification tree against which new documents can be tested. The tree is read from left to right. Branches upwards correspond to "yes" answers to the test at the node; branches downwards correspond to "no" answers. `Class 0` terminals correspond to the "not relevant" decision; `Class 1` terminals correspond to the "relevant" decision. The numbers in parentheses are estimates of the probability that the document is relevant[4].

Thus in this example if a document has fewer than 0.5 occurrences of the word `company` (i.e., the word does not appear) then the document is not relevant. On the other hand if the document has more than 0.5 occurrences (i.e., the word does appear) then additional features are tested to lead to a classification decision. Note that a test often requires multiple occurrences of the feature (e.g., for the word `federal`). Also note that this tree has only one Class 1 terminal, and that the pattern of features that indicate a relevant document can be written as a logical expression of feature counts. Thus we have:

```
company>0 & !fair & company<5
& securities<2 & takeover<2 & federal<10
```

as a description of a relevant document.

Other overall system features of note are:

- The exact word is used as the basis of the feature, although our implementation for TREC the actual test is case insensitive and is a substring match—so the feature `fair` would match with `Fair` or `FAIR` or `fairs` as well as with `fairground`. We do not, though, perform any stemming *per se*.
- No additional data structures are needed by CART. That is, we did not use any external data (e.g., thesauri or knowledge-bases) to help with tree construction.
- We spent approximately one-person month of effort in developing the

4. These are generated by CART from re-substitution error estimates and are, therefore, overly optimistic.

system infrastructure needed to handle the TREC data and to produce the official results. We used an "off-the-shelf" version of CART (written in C).

- The data were stored in compressed form and only uncompressed as needed. This was a satisfactory strategy for tree construction since the training sets involved relatively few documents. However, the time overhead was unacceptable when we came to do the actual test classifications.

## 4. Official Results and Performance Analysis

ADS submitted two sets of results for the routing queries. In the first set (denoted adsba1) we used the classification trees generated by exactly the information provided by the training data. In the second set (denoted adsba2) we used trees generated using an augmented training data. To generate this additional data we randomly selected an additional block of 50 Wall Street Journal articles from the training corpus and then one of us made the relevance judgements with respect to the 25 topics. This data was then added to the original collection of relevance judgements to provide a larger training set from which the second set of trees were grown. As noted above, we used a set of priors that reflect the low density of relevant documents, together with a cost function that encourages recall over precision. We also performed a number of auxiliary tests to help with our interpretation of the official results. These are all described in the following sections.

### 4.1 The Baseline Experiment

The baseline experiment (adsba1) was designed to explore how well our approach could do with absolutely no manual intervention and with the minimum of training data. So for this experiment we used just those documents in the training set for which there were relevance judgments.

Table 1 shows the performance on the baseline experiment together with the performance from the other Category B systems. We have chosen to show only the number of relevant-retrieved documents at the 200 document cut-off point since we believe that this gives a more accurate picture of the ability of the system to perform document routing than do the precision and recall numbers[5].

Table 1: Performance on Baseline Experiment

| Topic # | # Rel | Rel-Ret @ 200 | | | |
|---|---|---|---|---|---|
| | | adsba1 | Max | Median | Min |
| 1 | 131 | 2 | 67 | 32 | 2 |
| 2 | 172 | 15 | 33 | 21 | 9 |
| 3 | 304 | 3 | 130 | 48 | 3 |
| 4 | 20 | 1 | 18 | 7 | 1 |

Table 1: Performance on Baseline Experiment

| Topic # | # Rel | Rel-Ret @ 200 | | | |
|---|---|---|---|---|---|
| | | adsba1 | Max | Median | Min |
| 5 | 55 | 4 | 45 | 29 | 4 |
| 6 | 68 | 4 | 46 | 20 | 4 |
| 7 | 92 | 1 | 46 | 30 | 1 |
| 8 | 133 | 4 | 37 | 16 | 2 |
| 9 | 157 | 13 | 41 | 29 | 13 |
| 10 | 149 | 94 | 109 | 88 | 46 |
| 11 | 61 | 15 | 55 | 26 | 9 |
| 12 | 82 | 14 | 56 | 15 | 3 |
| 13 | 93 | 7 | 93 | 26 | 7 |
| 14 | 156 | 38 | 73 | 52 | 23 |
| 15 | 515 | 29 | 74 | 49 | 23 |
| 16 | 58 | 2 | 44 | 17 | 2 |
| 17 | 69 | 23 | 53 | 23 | 9 |
| 18 | 95 | 38 | 49 | 38 | 14 |
| 19 | 664 | 74 | 147 | 99 | 56 |
| 20 | 274 | 111 | 179 | 121 | 56 |
| 21 | 16 | 12 | 16 | 14 | 0 |
| 22 | 106 | 28 | 79 | 40 | 8 |
| 23 | 30 | 2 | 27 | 7 | 2 |
| 24 | 253 | 37 | 96 | 41 | 29 |
| 25 | 13 | 1 | 12 | 9 | 1 |

Overall performance of the baseline case is mixed and our analysis does not reveal any obvious correlation between performance and factors such as:

- the number of features extracted from the information need statements (the maximum number was 161, the minimum was 17, with the median being 39),
- the complexity of the topics (some are straightforward—such as Topic 13 "Mitsubishi Heavy Industries Ltd.", whereas others involve complex conditionals—such as Topic 1 "Antitrust Cases Pending"),

5. We return to this point in the final section of the paper.

- the number of training examples (Table 2 shows the size of the training sets—notice that for adsba1 the set size was approximately 30 for each topic with approximately 10 relevant instances for each topic), and
- the size of the optimal tree (Table 2 also shows the size of the optimal tree generated for each topic—this varied from a tree with only one terminal node to a tree with 10 terminal nodes, with the median being 2 terminal nodes).

In fact, given the paucity of information actually used to generate the classification trees, perhaps the most surprising aspect of these results is how well some of the trees perform. There were many instances in which the adsba1 trees generated the minimum response. However, there were several results around the median score and even one significantly above the median.

As an example of a tree that performed moderately, consider the optimal tree for Topic 22 "Counternarcotics." This had only one decision node—a split on the word coca[6]. The actual tree is:

```
     class 0 (0.050)
coca<=0.50
     class 1 (0.862)
```

That is, the classification is based in the presence of absence of the word coca. If it is present, then the document is marked as "relevant" and the estimate of the probability of it being relevant is 0.862; if it is not present, then the document is marked as "non-relevant" but in fact still has a small probability (0.050) of being relevant. As we see from the table, this tree identifies 28 out of a possible 106 relevant documents, by the 200 document cut-off point.

A tree that performed poorly is the one for Topic 9 "Candidate Sightings." Here the optimal tree had three decision nodes:

```
     class 0 (0.000)
camp<=0.50
               class 0 (0.000)
          ran<=1.50
               class 1 (0.948)
     sen<=9.00
          class 0 (0.000)
```

Thus a relevant document is one which contains the word camp, the word sen (an abbreviation for Senator) nine times or less, and has a least two occurrences of the word ran. This tree only identified 13 of 157 relevance documents and was the worst performing of the Category B systems.

On the other hand, the tree for Topic 10 "AIDS Treatments" performed very well. It

6. Recall that our feature extraction algorithm would match coca with any word with coca as a substring. So coca, cocaine and Coca-Cola would all match.

also had only one decision node, but correctly identified 94 of 149 relevant documents:

```
        class 0 (0.126)
azt<=0.50
        class 1 (1.000)
```

Thus relevant documents are those that contain the word `azt` (the name of a drug for treating AIDS patients).

### 4.2 The Effect of Additional Training Data

Our second set of official results was designed to investigate the sensitivity of system performance to the size of the training sets. To provide additional training samples we randomly selected a block of 50 Wall Street Journal articles[7] for which we generated relevance judgements for the first 25 topics. In practice this gave us some additional relevant articles, but mostly contributed to the non-relevant examples. Table 2 shows the effect of adding the additional documents—notice that some articles in the new set were already included in the original training data.

Table 2: Size of Training Sets and Optimal Trees

| Topic # | Size of Optimal Tree | | Size of Training Set | | | |
|---|---|---|---|---|---|---|
| | adsba1 | absba2 | adsba1 | | absda2 | |
| | | | Total | Rel | Total | Rel |
| 1 | 7 | 7 | 34 | 7 | 83 | 7[a] |
| 2 | 8 | 14 | 31 | 7 | 81 | 8 |
| 3 | 4 | 5 | 30 | 9 | 80 | 9 |
| 4 | 3 | 5 | 29 | 12 | 79 | 12 |
| 5 | 3 | 3 | 30 | 18 | 79 | 18[a] |
| 6 | 3 | 2 | 28 | 15 | 78 | 15 |
| 7 | 2 | 3 | 28 | 10 | 78 | 10 |
| 8 | 1 | 19 | 32 | 7 | 82 | 8 |
| 9 | 4 | 4 | 23 | 4 | 73 | 4 |
| 10 | 2 | 4 | 20 | 12 | 69 | 12[b] |
| 11 | 2 | 2 | 21 | 12 | 71 | 12 |
| 12 | 8 | 11 | 35 | 8 | 85 | 8 |
| 13 | 2 | 2 | 12 | 8 | 62 | 8 |
| 14 | 10 | 13 | 30 | 5 | 80 | 6 |

7. The articles were in the block starting with WSJ870311-0102 and ending with WSJ870324-0001.

Table 2: Size of Training Sets and Optimal Trees

| Topic # | Size of Optimal Tree | | Size of Training Set | | | |
|---|---|---|---|---|---|---|
| | adsba1 | absba2 | adsba1 | | absda2 | |
| | | | Total | Rel | Total | Rel |
| 15 | 3 | 13 | 27 | 7 | 77 | 10 |
| 16 | 2 | 3 | 36 | 3 | 86 | 3 |
| 17 | 2 | 2 | 30 | 12 | 80 | 12 |
| 18 | 2 | 3 | 18 | 14 | 68 | 14 |
| 19 | 2 | 3 | 30 | 12 | 80 | 15 |
| 20 | 2 | 3 | 30 | 14 | 80 | 14 |
| 21 | 2 | 2 | 33 | 12 | 82 | 12[b] |
| 22 | 2 | 2 | 28 | 10 | 78 | 10 |
| 23 | 2 | 2 | 29 | 10 | 79 | 10 |
| 24 | 2 | 2 | 48 | 10 | 98 | 10 |
| 25 | 2 | 2 | 39 | 6 | 89 | 6 |

a. Some relevant documents in the augmented training set already in the original training set.
b. Some non-relevant documents in the augmented training set already in the original training set.

The new training data did not have a significant impact on the size of optimal tree, except in those cases where there were additional relevant documents—that is, for topics 6, 8, 14, 15 and 19. The changes here were quite dramatic. For example, in the case of Topic 8 "Economic Projections" the addition of just one more relevant article changed the optimal tree from one with only one terminal node to one with 19! This suggests, of course, that for this topic the training data do not provide a very representative sample of texts.

Of more interest, however, is whether these additional training data had any effect on the overall performance of the system. Table 3 shows the official results for absda2. The results for adsba1 are retained for comparison, as are the results for the other Category B systems.

Table 3: Performance with Additional Training Data

| Topic # | # Rel | Rel-Ret @ 200 | | | | |
|---|---|---|---|---|---|---|
| | | adsba1 | absba2 | Max | Median | Min |
| 1 | 131 | 2 | 25 | 67 | 32 | 2 |
| 2 | 172 | 15 | 9 | 33 | 21 | 9 |

Table 3: Performance with Additional Training Data

| Topic # | # Rel | Rel-Ret @ 200 | | | | |
|---|---|---|---|---|---|---|
| | | adsba1 | absba2 | Max | Median | Min |
| 3 | 304 | 3 | 15 | 130 | 48 | 3 |
| 4 | 20 | 1 | 1 | 18 | 7 | 1 |
| 5 | 55 | 4 | 29 | 45 | 29 | 4 |
| 6 | 68 | 4 | 10 | 46 | 20 | 4 |
| 7 | 92 | 1 | 22 | 46 | 30 | 1 |
| 8 | 133 | 4 | 2 | 37 | 16 | 2 |
| 9 | 157 | 13 | 25 | 41 | 29 | 13 |
| 10 | 149 | 94 | 69 | 109 | 88 | 46 |
| 11 | 61 | 15 | 9 | 55 | 26 | 9 |
| 12 | 82 | 14 | 3 | 56 | 15 | 3 |
| 13 | 93 | 7 | 25 | 93 | 26 | 7 |
| 14 | 156 | 38 | 23 | 73 | 52 | 23 |
| 15 | 515 | 29 | 49 | 74 | 49 | 23 |
| 16 | 58 | 2 | 17 | 44 | 17 | 2 |
| 17 | 69 | 23 | 53 | 53 | 23 | 9 |
| 18 | 95 | 38 | 14 | 49 | 38 | 14 |
| 19 | 664 | 74 | 56 | 147 | 99 | 56 |
| 20 | 274 | 111 | 63 | 179 | 121 | 56 |
| 21 | 16 | 12 | 0 | 16 | 14 | 0 |
| 22 | 106 | 28 | 8 | 79 | 40 | 8 |
| 23 | 30 | 2 | 2 | 27 | 7 | 2 |
| 24 | 253 | 37 | 29 | 96 | 41 | 29 |
| 25 | 13 | 1 | 1 | 12 | 9 | 1 |

As for adsba1, the results are mixed. For 10 topics performance improved; for 13 topics performance got worse; and for 2 topics performance was unchanged. We also do not see any strong correlation between change in performance and change in the number of positive instances in the training set, or change in the optimal tree size. There is some indication that the while the extra positive instances tended to produce significant changes in optimal tree size, these new trees also tend to have poorer performance—suggesting, as we should expect, that the tree construction process is highly sensitive to local changes in these small training sets. Nevertheless there are some interesting results.

For example, the tree for Topic 22 "Counternarcotics" becomes:

```
     class 0 (0.000)
drug<=0.50
     class 1 (0.905)
```

Thus although the tree size is unchanged, the test is now on the word drug instead of coca. As we might expect this turns out to be a much less generally useful test and the tree only identifies 8 of the relevant articles—actually the minimum retrieved by a Category B system.

The tree for Topic 15 "CEO" is one which shows marked change from the adsba1 version, growing from two decision nodes to twelve, and retrieving 49 relevant documents instead of 29. The tree is:

```
          class 0 (0.000)
     chief<=0.50
                    class 0 (0.000)
               executive<=0.50
                                        class 1 (1.000)
                                   company<=0.50
                                        class 0 (0.000)
                              executive<=1.50
                                             class 0 (0.000)
                                        name<=0.50
                                             class 1 (1.000)
                                   company<=1.50
                                             class 0 (0.000)
                                        resign<=0.50
                                                  class 0 (0.000)
                                             chief<=1.50
                                                  class 1 (1.000)
                         name<=3.50
                              class 0 (0.000)
                    executive<=9.00
                         class 0 (0.000)
               appoint<=0.50
                    class 0 (0.000)
          ceo<=0.50
               class 0 (0.000)
```

This is a much more complex structure than the other trees illustrated so far. Note that there are three terminal nodes that lead to a document being classified as relevant. Although they are in the same sub-tree defined by the expression:

```
chief>0 & !ceo & !appoint & executive>0 & executive<10 & name<4
```

they make minor distinctions based on the words company, name, resign and chief. Thus we have three further tests:

```
executive<2 & !company
executive>1 & company>1 & resign>0 & chief>1
executive>1 & company<2 & name>0
```

to separate relevant from non-relevant documents.

Finally, the tree for Topic 17 "Measures to Control Agrochemicals" does the best of all the Category B systems detecting 53 of the 69 relevant documents. The tree has one decision node:

```
        class 0 (0.042)
pesticide<=0.50
        class 1 (0.920)
```

That is, a test for the presence of the word `pesticide`—a surprisingly simple structure given it performance.

### 4.3 Sensitivity to the Choice of Optimal Tree

Since the choice of optimal tree from the sequence of nested sub-trees is dependent on the cross-validation error estimates, and since the number of training samples is rather small, we might expect that there is a possibility that the tree selection process is in fact in error. To explore the sensitivity of the system's performance to errors in selecting the optimal tree we ran an auxiliary experiment for Topic 22 "Counternarcotics".

As in the official experiments, we used the adsba1 dataset to generate a sequence of sub-trees, but instead of selecting just one (i.e., T*), we saved all the trees. We then used each tree to classify the test data and tabulated the results. These are show in Table 4. The

Table 4: Performance as a Function of Tree Size

| Tree No. | Tree Size | $R_{cv}$ | Rel-Ret @ 200 | Recall @ 200 | Precision @ 200 |
|---|---|---|---|---|---|
| 1 | 6 | 0.3932 | 12 | 0.1132 | 0.0600 |
| 2 | 4 | 0.5004 | 1 | 0.0094 | 0.0050 |
| 3 | 3 | 0.3931 | 24 | 0.2264 | 0.1200 |
| 4 (T*) | 2 | 0.4645 | 28 | 0.2642 | 0.1400 |
| 5 | 1 | 0.7866 | – | – | – |

columns of the table record the tree identifier (Tree No.), the size of the tree in terms of the number of terminal nodes in the tree (Tree Size), the cross-validation error estimate ($R_{cv}$), the number of relevant documents retrieved (Rel-Ret @ 200), and the recall and precision at the cut-off (Recall @ 200 and Precision @ 200).

For this topic, the maximum tree ($T_1$) has six terminal nodes and an estimated error rate of 39%. The minimum tree ($T_5$) has only one terminal node—which in this case classifies all document as non-relevant—and an estimated error rate of 79%. The optimal tree ($T_4$) has two terminal nodes but an estimated error rate of 46%.

We see that the optimal tree did in fact generate the best result, increasing our confi-

dence that the tree selection process is working as intended despite the small sample sizes.

### 4.4 The Use of Surrogate Split Information

A basic problem with the use of classification trees in the TREC experimental environment is that although we use the probability of being relevant as a mechanism for ordering the system's output, this is a very weak method for generating a ranking. In fact, since we typically have very few "output bins" for any given tree, for most topics we effectively generated no ordering that would discriminate over the top 200 documents. In this situation we need to look for a mechanism for generating a "finer-grained" ranking.

Our second auxiliary experiment was designed to explore the use of the surrogate split information generated by the CART experiment as a way of ordering the output. Surrogate splits are a feature of CART used too deal with the problem of missing data. They define alternative splitting criteria in the case that the primary feature measurement is not available[8]. In TREC trees these appear as additional tests on the frequency of occurrence of words. So, for example, in adsba1 the optimal tree for Topic 22 "Counternarcotics" is:

```
        class 0 (0.050)
coca<=0.50
        class 1 (0.862)
```

and the alternatives to the split on the word coca are:

```
cocaine<=0.50   [1.00]
colombia<=0.50  [0.84]
drug<=0.5       [0.78]
```

where the number in brackets indicate how correlated the surrogate split is with the optimal split. Thus in this example a split on coca and cocaine are identical in term of their ability to classify the documents. We also show the next two highly correlated splits.

To use this information for output ranking we took the output from the adsba1 tree for Topic 22 (i.e., the one shown above) and then for each document classified as relevant by this tree we gave it a weighted score based on the number of surrogate split tests it also satisfied. Thus if a document contained only the word coca it received a score of 1.00. If a document contained coca and colombia then it received a score of 1.84. In general a document received a score that was the sum of the correlation coefficients for those tests that were satisfied[9].

---

8. Of course in the document routing problem addressed by TREC we do not have the notion of a missing measurement—a word is either present or not. However, it is easy to imagine document routing scenarios in which this does apply—for example, when working with noisy transmissions—so that the surrogate split information would be extremely useful.

The effect of using this information is shown in Table 5. The table demonstrates the impact on system performance for Topic 22 of adsba1. Without any ordering we detected

Table 5: Performance of T* as a Function of Output Ordering

| Tree/Output Ordering | Rel-Ret @ 200 | Recall @ 200 | Precision @ 200 |
|---|---|---|---|
| T*: no additional ordering | 28 | 0.2642 | 0.1400 |
| T*: ordering based on surrogate splits | 43 | 0.4057 | 0.2150 |

28 relevant documents in the first 200, but with the ordering scheme just described we were able to improve this to 43 in the first 200. This in turn translated into an increase of 14 points of recall and 7 points of precision at the 200 document point.

The surrogate splits also give us some insights into the overall behavior of the tree selection process as the number of training samples change. Thus although the optimal tree for Topic 22 in adsba2 was:

```
     class 0 (0.050)
drug<=0.50
     class 1 (0.862)
```

with worse performance than the optimal tree for adsba1, when we look at the top three surrogate splits we see that they are:

```
cocaine<=0.50      [0.89]
coca<=0.50         [0.88]
government<=0.5    [0.86]
```

That is although the optimal split for the augmented training set changed, we still see the importance of the same set of word features, which in turn indicates are certain stability in the underlying feature space. We might predict that as the training set increases in size that we would see the splits also becoming more stable.

### 4.5. Commentary

The TREC corpus represents a significant challenge for our system. Our previous results with a small corpus, while encouraging, did not allow us to evaluate how well the technique might do with realistically sized document collections. Our conclusion based on the results we have from TREC is that CART does exhibit some interesting behaviors on a realistic corpus, and that, despite the small size of the training sets and the restricted choice of features, for some topics it produces competitive results. So although the overall performance is moderate (relative to the better performing systems at TREC), we believe that the absolute performance (given that the system is totally auto-

9. As yet, there is no theoretical justification for this algorithm. It does however have the intuitive property that documents that satisfy additional splits get a higher score in proportion to the "power" of those splits.

matic) is at least encouraging and definitely acceptable in several instances.

Some specific observations on the performance of the current implementation of the CART algorithm are:

- Relying on the re-substitution estimates for the terminal nodes is a very weak method for producing an output ranking. The estimates themselves are not very good and when combined with optimal trees that emphasize recall over precision give a largely undifferentiated output. As we noted above, a scheme that makes use of surrogate split information to generate a *post hoc* ranking shows much promise as a technique for improving our scores in the TREC context.
- While our approach is totally automatic, it is restricted to using as features only those words that appear in the information need statement. This is obviously a limitation since the use of even simple query expansion techniques (e.g., stemming and/or a synonym dictionary) is likely to provide a richer and more effective set of initial features.
- Using words as features is possibly too "low-level" to ever allow stable, robust classification trees to be produced. At a minimum, we probably need to consider working with concepts rather than individual words. Not only would this reduce the size of the feature space but would probably result in more intuitive trees. The disadvantage of this that it is not clear where the concepts would come from, other than from a manually constructed knowledge-base of some sort.
- We need to work with much bigger and more representative training sets. Our preliminary experiment in this area shows, not surprisingly, that adding more training examples can lead to dramatic changes in the classification trees.

As a final comment, we would like to suggest that the overall evaluation paradigm used in TREC does not properly assess the performance of systems on the routing task. Although *ad hoc* retrieval and routing are similar when viewed in terms of the basic technology, systems designed and built to support these two applications have significantly different requirements. In particular, operational routing systems do not usually emphasize output ordering but instead focus on optimizing the trade-off between detection and false alarm rate. In this respect, at least, we believe that recall and fallout are better indicators of routing performance than recall and precision. Furthermore, artificially limiting reported output to the first 200 documents automatically discriminates against those routing systems that actually do attempt to perform the recall/fallout trade-off. A fairer set-up for the routing component of TREC would be to allow systems to report exactly those documents marked as relevant. Comparison of systems would be more complex since different systems will produce different numbers of documents, but individual scores would give a better picture of routing performance.

## 5. CART as the Kernel of a Document Routing System

Users today are faced with ever increasing amounts on real-time text that must be sifted for relevant information. This information space is: massive—both in terms of the number of sources and the volume of material available within each source; dynamic—sources and their contents are changing constantly and especially within the time horizon of any specific analysis problem; and heterogeneous—each source represents information in different ways and independently of the others.

In this environment, users require tools that can perform document filtering and selection that are: easy to learn and use, easily adaptable to changing and ill-defined information needs, portable across sources and analysis domains, and give good precision and recall. With our TREC results in hand, we are in a position to briefly consider the potential role of CART as a central component in such an operational document routing system.

Figure 2 illustrates how a such a system might be organized. In our scenario, we

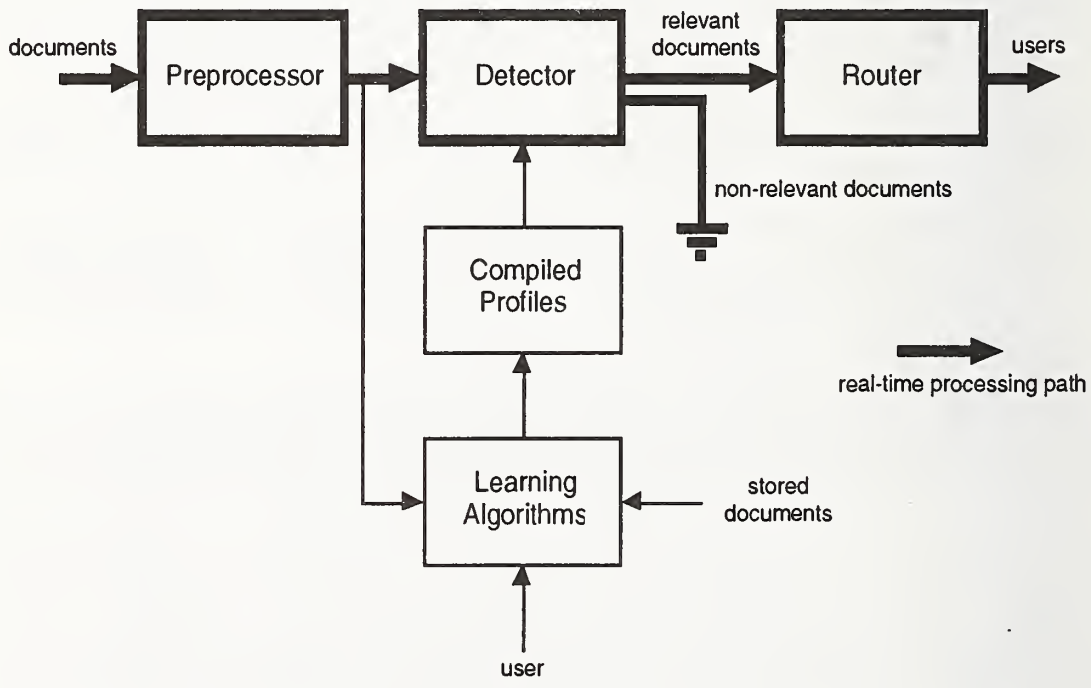



Figure 2: A Document Routing System Concept

imagine that documents enter the system in real-time and after preprocessing to extract features are passed to the detection module where they are either rejected as being non-relevant to any of the user profiles stored in the system, or are marked as relevant and passed to the router for transmission to users. Note that in our proposed architecture, learning takes place in parallel to actual operation of the real-time routing system so that the classification trees can be updated as necessary when new training instances become

available or when users interests change.

While we would make no claim that CART alone is sufficient to guarantee high-performance detection and routing, we do believe that its ability to work automatically with any size data set and with any set of specified features means that it can be a very cost effective component of such a system. Indeed we believe that it is probably best used as an initial filter to screen out non-relevant documents and that CART's output might then be fed to a more language-oriented algorithm to decrease the false alarm rate.

To summarize, we consider CART to be an important tool in our arsenal of effective and efficient document detection and routing technologies. While the results from TREC are preliminary, we believe that they do demonstrate that CART has a number of advantages over other approaches, namely:

- classification trees are constructed automatically from specifications of features and a set of examples,
- the learning algorithm generates an "optimal" classifier together with useful auxiliary data and statistics such as the misclassification probability,
- prior class probabilities can be used if known,
- specification of the misclassification cost function provides for direct control of the fallout and recall of the classifier, and
- the classification trees are easily understood and interpreted by end users.

In addition, the CART algorithm is completely language independent, in the sense that it make no assumptions about the inherent features of the source language of the document—all that it requires are "features" and training examples. Further, the features themselves can be extracted from document externals as well as document internals.

In TREC-2 we will explore some of the extensions discussed in Section 4 to show how CART can indeed be integrated into a system to help users who are faced with the task of searching through the "information wilderness."

**References**

[1] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth & Brooks, Pacific Grove, CA. 1984.

[2] S. L. Crawford. Extensions to the CART Algorithm. *International Journal of Man-Machine Studies*, 31:197-217, 1989.

[3] S. L. Crawford, R. M. Fung, L. A. Appelbaum, and R. M. Tong. Classification Trees for Information Retrieval. *Proceedings of the Eighth International Workshop on Machine Learning (ML91)*. Morgan Kaufmann, San Mateo, CA. 1991.

[4] C. Fox. A Stop List for General Text. *SIGIR Forum*, 41:19-35, Fall/Winter 1989/90.

# Compression, Fast Indexing, and Structured Queries on a Gigabyte of Text

Alan Kent
Alistair Moffat
Ron Sacks-Davis
Ross Wilkinson
Justin Zobel

Collaborative Information Technology Research Institute
Departments of Computer Science
RMIT and The University of Melbourne
723 Swanston St, Carlton, Melbourne 3053
Australia
{ajk, alistair, rsd, ross, jz}@kbs.citri.edu.au

## 1 Introduction

Large document collections present many problems for practical information retrieval. To avoid unnecessary accesses to the text of the collection during query evaluation, comprehensive indexes are required. It must be possible to create and access these indexes in a reasonable amount of time, and to store them, and the data itself, in a reasonable amount of space. Moreover, although advanced indexes permit efficient evaluation of conventional ranking measures, they make only limited use of any structure inherent in the query.

Here we describe two separate streams of research that address these problems. In the first stream we have over the last two years developed indexing and compression techniques that allow the text to be stored compressed [2, 8]; provide fast access to document collections via compressed indexes [2, 9, 16]; create indexes with a fast inversion technique [7]; and permit fast ranking of large document collections [17]. On the small document collections initially available to us (the largest was 132 Mb), compressed text and index typically required less than 35% of the space required for the source text; query response times were a few seconds; and peak memory requirements were less than a Megabyte during query processing. In Section 2 we describe the application of these techniques to the TREC data.

In the second stream of research we used the compression and indexing techniques discussed above, as well as signature file indexing techniques developed previously by the group [5, 11], to investigate the extent to which the structure of queries aids the retrieval process. In Section 3 we report on a series of experiments that examine how this structure may be used in both generation of Boolean queries and in ranking. As part of these experiments we are able to test the advantage of using adjacent pairs in retrieval.

## 2 Compression and Inverted File Indexing

We first describe the structure of text databases with inverted file indexes, and how documents are ranked with regard to a query. Then, in Section 2.3, we describe the compression of the text of the documents, and in Section 2.4 describe the representation of the inverted file and

associated information. The result of applying these techniques to the TREC data is described in Section 2.5.

## 2.1 Text databases

Text databases provide access to text documents on the basis of their content. We assume that access is via the inverted file indexing scheme we have described elsewhere [16]. In this scheme, each distinct word in the database is held in a *vocabulary*, which may be an array, or may be a search structure such as a B-tree. For each word in the vocabulary the *inverted index* stores an *index entry*, a list of the identifiers of documents containing the word. Interleaved with the identifiers are the number of occurrences of each word in each document. Since the index entries contain ordinal document identifiers rather than disk addresses, a *document mapping* is needed to convert identifiers into addresses. This will require either a non-trivial amount of memory space, or must be stored on disk in an auxiliary file. Similarly, an *index mapping* is required to turn an ordinal word number into the address of the corresponding index entry.

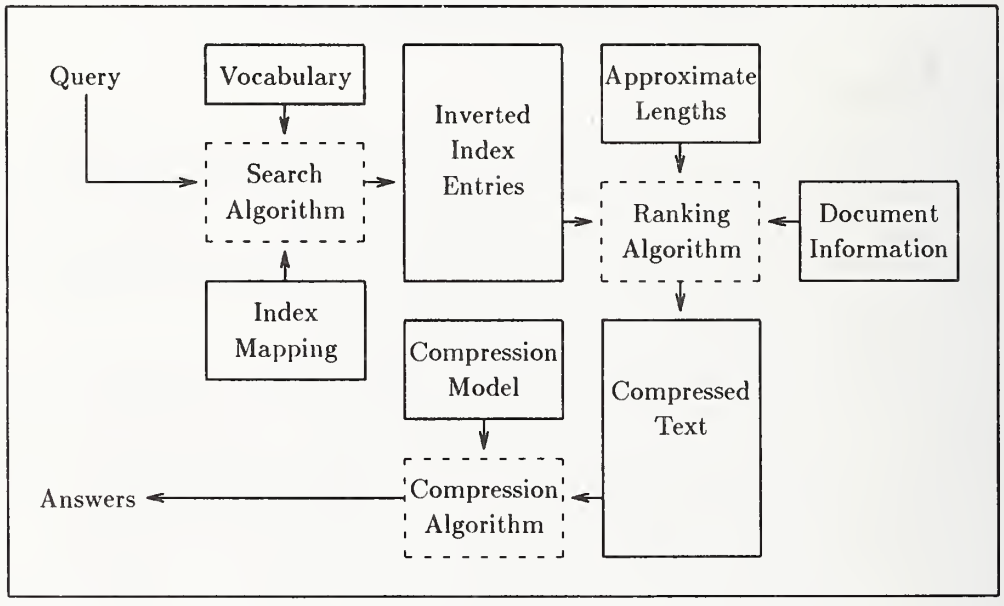



Figure 1: Organisation of text databases

Other components of the database are the *compression model*, used for text compression, and the *document lengths*, used for ranking. These are discussed later. In our experiments, the vocabulary and compression model were held in memory, together with an array of *approximate lengths*, also described below; all of the other structures were stored on disk. Figure 1 shows the relationship between these various components.

## 2.2 Query evaluation

Given an inverted file index of the format described, it is straightforward to rank the documents in a database with respect to an informally phrased query, returning, say, the top ten documents as answers. As an example, consider the cosine measure, one of the most effective ranking

techniques [1, 13, 14]. The cosine measure evaluates the relevance of a document to a query via the function

$$cosine(q,d) = \frac{\sum_t w_{q,t} \cdot w_{d,t}}{\sqrt{(\sum_t w_{q,t}^2)} \cdot \sqrt{(\sum_t w_{d,t}^2)}} ,$$

where $q$ is the query, $d$ is the document, and $w_{x,t}$ is the weight of word $t$ in document or query $x$. A common function for assigning weights to words in document or query $x$ is to use the frequency-modified inverse document frequency, described by

$$w_{x,t} = tf_{x,t} \cdot \log_e(N/f_t) ,$$

where $tf_{x,t}$ is the number of occurrences of word $t$ in $x$, $N$ is the number of documents in the collection, and $f_t$ is the number of documents containing $t$. This is commonly called the *tf·idf* weighting.

The information required to compute the sum $\sum_t w_{q,t} \cdot w_{d,t}$ is held in the inverted index. To answer the query, an accumulator with initial value zero is created for each document in the database; the index entry for each word in the query is retrieved; and, for each 'word-number, document-number, frequency-in-document' triple, a cosine contribution is added into the corresponding accumulator. This technique is also used by Harman and Candela [4].

For each document in a static database the *document length*, $\sum_t w_{d,t}^2$, is a constant, and can be pre-computed and stored at the time the database is created. Thus, to compute the final ranking after the accumulators have been calculated, the document length for each document with a non-zero accumulator value must be fetched, the accumulator value divided by this length.

For the queries supplied with TREC, over 90% of the documents contained one or more of the query terms, meaning that over 90% of the document lengths needed to be accessed. Moreover, at over 2 Mb, the table of document lengths is large, and in practice space limitations might prevent it being held in memory. We have shown, however, that keeping limited precision *approximate lengths* in memory can largely eliminate the cost of fetching document lengths from disk [17]. Suppose we are prepared to allow $b$ bits of main memory storage for each document length $x$, where $L \leq x \leq U$. Then if $B = [(U + \epsilon)/L]^{2^{-b}}$, where $\epsilon$ is a very small number, the function $f(x) = \lfloor \log_B(x/L) \rfloor$ yields integers $c$ between 0 and $2^b - 1$ for all legal values of $x$, and $g(c) = L \cdot B^c$ can be used to compute approximations to $x$.

The $c$ values stored can then be used in either of two ways. First, the approximate lengths can be used to determine which exact document lengths should be retrieved from disk. Second, *cosine* can be computed with $g(c)$, yielding an approximate ranking. With as few as two bits per length, there was only a small decline in retrieval performance in the five small document collections for which we had relevance judgments [17]. The effects of these techniques on TREC are discussed in Section 2.5.

## 2.3 Text compression

Using a word-based model of the text, the space required to store the documents comprising a database can be reduced to less than 30% of the original size [2, 8, 15]. Each word occurrence in the text is replaced by a canonical Huffman code, the length of which is dependent on the frequency of the word, and the intervening 'non-words' are similarly coded against a vocabulary of non-words. Thus, by alternately decoding a word, then a non-word, and so on down the compressed text, the exact form of the input document can be reconstructed. For further details of the scheme used the reader is referred to our previous work [15].

The decompression process is very fast. The canonical Huffman code used is particularly amenable to a table-based 'look up and copy' implementation, and each such decoding step generates several output bytes. As a result, the compression regime has only limited impact on retrieval time. Moreover, the small amounts of time needed for decompression are partially (and sometimes fully) offset by reduced disk traffic.

Because every word and non-word decompressed is retrieved from the compression model, it is crucial that this model be held in main memory. If the compression model was stored on disk, two or more disk accesses would be required for each word in each document retrieved. On the smaller databases, the compression model did not exceed about 500 Kb. We were very interested to see how much space would be required to hold the model for the TREC collection.

### 2.4 Inverted file compression

The inverted file index entries should also be compressed, since without compression the inverted file might take as much space as the original text itself [2, 9, 16]. Good compression of index entries can be achieved by numbering documents sequentially from 1, sorting index entries, representing each sequence of identifiers as a sequence of differences or gaps, and then using compact representations of the generally small integers that result.

For example, a word might appear in the first, fourth, seventeenth, ..., documents, with an index entry of

$$1, 4, 17, 91, 113, \ldots$$

This can be represented by the sequence of differences

$$1, 3, 13, 74, 22, \ldots ,$$

which can then be compressed.

We have experimented with several different techniques for compressing these runlengths, in two broad classes [8, 9]. *Global* methods use the same encoding for all entries, and so have the advantage of being more general, but are insensitive to the frequency of each term. *Local* methods code each entry using a code that is adjusted to take into account the frequency of the term.

Use of a variable length prefix code allows the frequent small runlengths to be represented more succinctly that the less frequent long runlengths, and is a marked improvement over the standard binary encoding. The further advantage of using a local code arises because of the high variability of word frequency, and the effect this has on the average runlength within the entry. For example, the word 'the'[1] can be expected to have a sequence of runlengths where each difference is very small, usually one. On the other hand, a rare word such as 'palimpsest' will have gaps that are often much larger, but also sometimes small, since rare words will tend to occur in clusters of related documents within the collection. Hence, a local compression method that adjusts its codes according to word frequency can be expected to perform better than a global method, and this is indeed the case [9]. Term 'frequency-within-document' values are efficiently represented by use of the $\gamma$ code [16].

After compression, the index entries for a text database will typically occupy less than 10% of the space required for the text itself. Decompression is fast, with about 100,000 document identifiers decoded per second on a 25 MIP Sun SPARC 2. None of these compression methods require any significant memory resources, once the index entry has been retrieved.

[1] We did not 'stop' any words, and also indexed all numbers. We did, however, stem all words before indexing, using Lovins's algorithm [6].

We have also used these inverted file encodings in the inversion process required during database construction, again capitalising on the large main memory offered by current workstations and the ability to perform two passes over a static database [7]. An alternative approach is the disk-based method described by Harman and Candela [4]; they report that the inversion of an 806 Mb database required 313 hours and 163 Mb of disk work space to produce an inverted index of 112 Mb. We were thus very interested to see whether the in-memory approach would be successful for the TREC collection.

## 2.5 Experimental results

This section describes the performance of our techniques on the 2055 Mb TREC collection. All results are for a 25 MIP Sun SPARC 2 with 256 Mb of main memory and no other active user processes.

### Compression

The two components of the compression process took about three hours each, generating a compression model and a compressed database of 4.2 Mb and 605.1 Mb respectively (Table 1). The second pass also generated a file containing the document mapping, a little under 3 Mb. That is, including both of these auxiliary files, the document collection was reduced from 2055 Mb to 612 Mb, or 29.8%, a disk saving of 1400 Mb.

The need to store the coding model meant that the compression passes were expensive on memory. With more than 900,000 unique symbols, and an additional storage requirement of about 12 bytes per word for the search structure, each pass required about 25 Mb of data structures. These figures reinforce our contention that database creation needs to be performed on a reasonably well configured machine, even if query processing is performed elsewhere.

| Pass | Output files | Size | | CPU Time | Memory |
|---|---|---|---|---|---|
| | | Mb | % | (Hr:Min) | (Mb, peak) |
| First pass: | | | | | |
| Compression | Compression model | 4.2 | 0.2 | 2:37 | 25.6 |
| Inversion | Vocabulary | 6.4 | 0.3 | 3:02 | 18.7 |
| Overhead | | | | 0:19 | 2.5 |
| Total | | 10.6 | 0.5 | 5:58 | 46.8 |
| Second pass: | | | | | |
| Compression | Compressed text | 605.1 | 29.4 | 3:27 | 25.6 |
| | Document mapping | 2.8 | 0.1 | | |
| Inversion | Inverted index | 132.2 | 6.4 | 5:25 | 162.1 |
| | Inverted index mapping | 2.1 | 0.1 | | |
| | Document lengths | 2.8 | 0.1 | | |
| | Approximate lengths | 0.7 | 0.0 | | |
| Overhead | | | | 0:23 | 2.5 |
| Total | | 745.8 | 36.3 | 9:15 | 190.2 |
| Overall Total | | 756.4 | 36.8 | 15:13 | 190.2 |

Table 1: Files in compressed TREC database ($b = 8$). Percentages are relative to the size of the original document collection (2055.3 Mb)

## Inversion

The inversion process was performed as two passes in tandem with the two compression passes. The first stage, counting term frequencies, had similar time and memory requirements to the first component of the compression process, but could not actually share data structures because of differing definitions of 'words' and the need, in the index, for stemming.

The second pass of the inversion process was where we expected to encounter problems. The in-memory technique requires the allocation of memory a little larger than the final size of the compressed inverted file, and our previous experiments had indicated that this would be roughly 10% of the size of the input text [7]. That is, we expected to use more than 200 Mb of the 256 Mb memory available. However, the long TREC documents—averaging 470 words rather than the 90 for our previous largest collection—meant that the inverted file was smaller than anticipated, and, as it turned out, the inversion process ran smoothly to completion in 162 Mb, and contributed five and a half hours to the time of the combined second pass. One would not want to do this every day, but these resource requirements are well within the capabilities of a typical $50,000 workstation.

Using the 'local $V_G$' code [8, 9], the final inverted file required 132 Mb, or 6.4% of the input text, of which about 40 Mb was by $\gamma$ coded 'frequency-within-document' values. Hence, if only Boolean queries were to be supported, the inverted file could be reduced to under 100 Mb.

The second pass of the inversion process also produced three other files—the index mapping for the inverted file, giving the bit address for each entry; the file of exact document lengths (later merged with the document mapping to form the document information file); and a file of approximate document lengths. The sizes of these, plus all of the other components of the compressed database, are shown in Table 1.

## Performance on ranked queries

The 50 test queries were transformed, by removal of SGML tags, stop words, and punctuation, into lists of words to which the cosine measure could be applied. The top 200 ranked documents were then retrieved and decompressed for each query.

| Operation | File | Disk Accesses | Data (Kb) | CPU Time (sec) | Elapsed (sec) |
|---|---|---|---|---|---|
| Searching | Index mapping | 43 | 170 | | |
| Ranking | Inverted Index | 43 | 1708 | | |
| | Document Information | 219 | 7 | 16.6 | 18.2 |
| Compression | Compressed text | 200 | 203 | | |
| | Answers | — | 729 | 3.7 | 7.2 |
| Total | | 505 | 2817 | 20.3 | 25.4 |

Table 2: Ranked queries on TREC ($b = 8$)

Processing of these queries was remarkably quick. Table 2 shows, for the three stages of processing each query, the average amount of data involved and the average time required. (We did not separate the time taken by the searching algorithm and the ranking algorithm.) As can be seen, on average the queries were processed in about 20 seconds of CPU time. Given that each query involved approximately 2.1 million document numbers and 600,000 non-zero accumulators, we were very pleased with this result. Note that the times listed in

Table 2 include the cost of writing the retrieved documents to disk for later analysis by a post-processing program, but do not include the time required by the post-processing.

On average the queries required about 500 disk accesses and the transfer of 2.8 Mb of data. At (say) 100 accesses per second on the disk[2], and a transfer rate of 1 Mb/sec, the input/output operations contribute about 5–8 seconds to the elapsed query time, accounting for the observed difference between CPU times and elapsed times.

For small queries response was even more impressive. For a query involving four terms ('Australia wheat tariff subsidy') and retrieval of the top ten documents, 2.6 seconds of CPU time were required; and, if output was directed into a file, the total elapsed time for the query was under 4 seconds.

The use of approximate lengths in-memory (using $b = 8$) meant that on average only 219 exact weights were required to identify the top 200 documents for the 50 TREC queries. Depending on whether the exact lengths would have been stored in memory or on disk, with the use of approximate lengths we have either reduced the memory required by over 2 Mb, at the expense of 19 disk operations; or, alternately, at a cost of 725 Kb of memory, we have avoided the need to sequentially read the entire exact lengths file, a saving of several seconds per query.

We also tested using the limited precision lengths to provide the final ranking, ignoring the exact lengths altogether. This means that the cosine measure is approximate, and some variation in recall effectiveness can be expected. In our previous experiments on smaller collections this degradation was minimal for even very low values of $b$. We used the 47 partial judgments supplied, and compared the number of relevant answers using the exact ranking and the approximate ranking technique. These results are shown in Table 3. Note that relevance judgements were not provided for all documents, hence the range for precision—most queries returned some documents that were un-judged. If we pessimistically assume that these documents were not relevant, the precision should be taken to be the lower value. Clearly, although the approximate ranking is finding different documents, the use of approximate ranking has, down to $b = 4$, little effect on precision.

| $b$ | Number of answers | | | | |
|---|---|---|---|---|---|
| | 5 | 15 | 30 | 100 | 200 |
| exact | 55.7–56.2 | 52.1–52.3 | 48.8–49.6 | 40.7–44.1 | 34.2–43.3 |
| 12 | 55.3–55.7 | 51.9–52.2 | 48.8–49.6 | 40.7–44.1 | 34.1–43.3 |
| 10 | 55.3–55.7 | 51.8–52.1 | 48.9–49.7 | 40.6–44.1 | 34.1–43.2 |
| 8 | 54.9–55.7 | 51.2–51.5 | 48.8–49.5 | 40.6- 44.1 | 34.2–43.3 |
| 6 | 55.7–56.6 | 50.8–51.3 | 47.5–48.4 | 40.6–44.5 | 34.1–44.0 |
| 4 | 57.9–59.1 | 50.2–50.9 | 46.5–48.6 | 38.0–45.6 | 31.2–48.0 |
| 2 | 42.6–46.4 | 32.6–42.4 | 29.9–46.6 | 21.3–57.7 | 15.7–65.8 |

Table 3: Effect of varying $b$—percentage precision, 47 queries

[2]One hundred accesses per second for the file containing the compressed text is optimistic for purely random accesses, since a transfer of several Kb from a random point in a large file in the Unix file system will typically require three or more seeks, each costing perhaps 10 to 20 milliseconds. However, the accesses to the document information file are in sequential order, and the accesses into the compressed text were batched into groups that were performed in sequential order.

### Memory requirements

In total, the peak memory requirement during query processing is about 10.9 Mb (not all components are required to be co-resident, and, in particular, the accumulators and answer buffer are never simultaneously present). If necessary, this could be further reduced by writing the output text directly to disk rather than into an answer buffer. However, we do not feel that this requirement is excessive given the large size of the document collection, and to date the use of approximate weights is the only area where we have concentrated on reducing memory usage.

## 3 Structured Queries

We focused in this experiment on structured queries. They would be transformed in two ways: into Boolean queries that would return a given number of documents, and into a vector of weights for the purposes of ranking. The system in which these experiments were carried out was a nested relational database, Atlas, that uses the multi-organisation signature file scheme [10]. The bulk of these experiments were carried out on the second part of the Tipster database. At the end of this section, we will report on some of the experiments that we have carried out on the whole database.

### 3.1 Boolean Query Generation

There are good reasons for preferring ranked retrieval over Boolean retrieval. Nevertheless, there are two key reasons that Boolean retrieval might be used. The first is that Boolean retrieval is computationally simpler, and the second is that the retrieval system may not support ranked retrieval. Since both factors were relevant to the set of experiments described here—we had been working with a Boolean retrieval system rather than the system described in Section 2—it was desirable to form a Boolean query and rank the answers it returned.

There were several requirements that a Boolean query generation method needed to satisfy. The first is that it should return a fixed number of documents. This allowed control over query time by fixing the number of documents to be subsequently ranked. Secondly, the query should be in disjunctive normal form, and have relatively few disjuncts. This is because query time is almost linear in the number of disjuncts in signature file retrieval systems. Thirdly the query generation mechanism should be automatic. These are the requirements that have previously been used as criteria for Boolean query generation algorithms [12]. However, the queries that we are considering in the TREC experiments have additional structure that may be used to advantage. Despite the limited amount of text to be processed, no natural language processing was performed; all processing was statistically and structurally based.

There were many possible approaches to Boolean query generation. A starting point might be to use one of the Boolean query generation algorithms previously developed [12] on the <narrative>. However, this would not adequately use the other available structures. We considered three approaches: base the query on the <concepts>, base the query on the <description>, augmented by the <narrative>, and base the query on the <topic>, <description>, and <narrative>, using adjacent pairs in the disjuncts as well.

The first approach used a list of concepts. The words within the concept were ordered with the most frequent word in the full text of the query first, and the least frequent last. Starting at the first concept, a disjunct was formed by creating a conjunction of terms from the concept, starting at the first word, until the conjunct returned fewer documents than the desired limit.

If the conjunct had fewer words than was required to meet this termination condition, other words were added to the concept by finding words in the remaining text that were adjacent to the first word in the concept. Finally, words that occurred rarely in the database were added as simple disjuncts if the query was still estimated to return fewer than the required number of documents. For example, to return 500 documents for query 82 on genetic engineering, we generated the Boolean query:

(genetic AND engineering) OR (manipulation AND molecular)
OR (biotechnology AND genetic AND manipulation)
OR (animal AND drug AND plant)

The second approach started by forming a simple disjunction of rare words, up to 10% of the required number of documents. Next, two key descriptors were identified, using the text of the description primarily, but also using the narrative. The key descriptors were created as a set of three words each, where each key descriptor is intended to capture a principal characteristic of the query. The frequency of the terms in the query and the adjacency of words were used to generate the key descriptors. Sometimes the algorithm would form two variations of the one concept, sometimes two distinct concepts. For instance query 82 formed the key descriptors:

(genetic, engineering, developed)
(genetic, engineering, manipulation)

to generate the query:

(genetic AND engineering) OR (genetic AND manipulation)

The third approach was quite similar to the second, except that more emphasis was placed on the title in generating the concepts, and, by using pairs of words in the query, finer gradation of the Boolean queries was possible. The concepts for queries 82 and 87 were the same. The Boolean query for query 82 was the same as for the second algorithm, however the query generated for 87 was:

institutions_failed OR (actions_criminal AND officers)

When evaluating these approaches, the key consideration is whether or not relevant documents are identified. For a fair comparison, each algorithm should return the same average number of documents. In Table 4, each algorithm has returned an average of about 400 documents. The recall and precision results are all based on the same ranking formula. Note that since these algorithms identified documents that had not been assessed as relevant or irrelevant, we made the (pessimistic) assumption that all such documents were irrelevant.

| Algorithm | Concepts | Description | Description with Pairs |
|---|---|---|---|
| Docs. Requested | 200 | 700 | 600 |
| Docs. Returned | 453 | 354 | 360 |
| Disjuncts | 3.5 | 1.8 | 4.4 |
| Recall | 0.155 | 0.170 | 0.146 |
| Precision | 0.150 | 0.144 | 0.121 |

Table 4: Comparison of Boolean query generation algorithms

To show how increasing the number of documents requested affects these algorithms, in Table 5, we show how many documents were retrieved and how many disjuncts were used when each of the algorithms were requested to return 1,000 documents.

| Algorithm | Concepts | Description | Description with Pairs |
|---|---|---|---|
| Docs. Returned | 1221 | 543 | 676 |
| Disjuncts | 4.01 | 1.94 | 6.03 |

Table 5: Further comparison of Boolean query algorithms

¿From these tables we are able to see that the least costly algorithm is based on finding two key descriptors. If pairs are added, there are more rare terms, which each contribute a single disjunct.

## 3.2 Ranking

Ranking documents using the vector space model has usually treated both documents and queries as a flat structures—lists of words. However, it is very simple to combine different representations of a query in ranking. If each form of a query is represented as a unit vector over the same vector space, we may combine the representations by vector addition. We are thus able to combine structural information and statistical information in the same measure.

The first series of ranking experiments were used to determine the relative usefulness of some of the query fields. The results are given in Table 6. The third Boolean query generation algorithm was used to generate a set of approximately 1,000 documents for each query. These were ranked using each field successively, and then combined, ignoring structure. In all these experiments we give first the results in terms of recall and precision, and then in terms of precision in terms of the number of viewed documents.

Let $V_t$ stand for the title vector, $V_d$ stand for the description vector, $V_n$ stand for the narrative vector, $V_c$ stand for the concepts vector, $V_f$ stand for the definition vector, $V_a$ stand for the vector for all of the text, and $V_p$ stand for vector of all adjacent pairs in the query.

| Recall | 10% | 20% | 30% | 40% | 50% | 60% | 70% | 80% | 90% | 100% | Av. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $V_d$ | 0.197 | 0.114 | 0.032 | 0.014 | 0.015 | 0.012 | 0.000 | 0.000 | 0.000 | 0.000 | 0.038 |
| $V_n$ | 0.192 | 0.112 | 0.034 | 0.014 | 0.015 | 0.012 | 0.000 | 0.000 | 0.000 | 0.000 | 0.038 |
| $V_a$ | 0.191 | 0.114 | 0.034 | 0.018 | 0.015 | 0.012 | 0.000 | 0.000 | 0.000 | 0.000 | 0.038 |

Table 6: Ranking using individual fields

Table 7 shows what happens if each of the five fields are added with the same weight, and then what happens if adjacent pairs are added as well. By way of comparison, we show the effect of adding pairs to the full text vector, and by adding pairs to the narrative vector.

$$
\begin{aligned}
V_1 &= V_t + V_d + V_n + V_f + V_c \\
V_2 &= V_t + V_d + V_n + V_f + V_c + V_p \\
V_3 &= V_a \\
V_4 &= V_a + V_p \\
V_5 &= V_n \\
V_6 &= V_n + V_p
\end{aligned}
$$

| Recall | 10% | 20% | 30% | 40% | 50% | 60% | 70% | 80% | 90% | 100% | Av. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $V_1$ | 0.209 | 0.112 | 0.034 | 0.014 | 0.015 | 0.012 | 0.000 | 0.000 | 0.000 | 0.000 | 0.040 |
| $V_2$ | 0.209 | 0.112 | 0.034 | 0.014 | 0.015 | 0.012 | 0.000 | 0.000 | 0.000 | 0.000 | 0.040 |
| $V_3$ | 0.191 | 0.114 | 0.034 | 0.018 | 0.015 | 0.012 | 0.000 | 0.000 | 0.000 | 0.000 | 0.038 |
| $V_4$ | 0.198 | 0.115 | 0.037 | 0.018 | 0.015 | 0.012 | 0.000 | 0.000 | 0.000 | 0.000 | 0.039 |
| $V_5$ | 0.192 | 0.112 | 0.034 | 0.014 | 0.015 | 0.012 | 0.000 | 0.000 | 0.000 | 0.000 | 0.038 |
| $V_6$ | 0.198 | 0.111 | 0.045 | 0.014 | 0.015 | 0.012 | 0.000 | 0.000 | 0.000 | 0.000 | 0.039 |

Table 7: All field ranking

Using this information, we tried to combine the various fields in "creative" ways. In Table 8 we show the results of creating a combined vector described by

$$V_7 = 1V_t + 2V_d + 3V_n + 0.5V_f + 3V_c + 2V_p$$
$$V_8 = 1V_t + 1V_d + 1V_n + 0.5V_f + 3V_c + 3V_p$$

| Recall | 10% | 20% | 30% | 40% | 50% | 60% | 70% | 80% | 90% | 100% | Av. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $V_7$ | 0.208 | 0.112 | 0.034 | 0.014 | 0.015 | 0.012 | 0.000 | 0.000 | 0.000 | 0.000 | 0.039 |
| $V_8$ | 0.205 | 0.112 | 0.033 | 0.014 | 0.015 | 0.012 | 0.000 | 0.000 | 0.000 | 0.000 | 0.039 |

Table 8: Combining the fields

The first thing to note is how similar the results are. Since a limited number of documents are being examined using an imperfect Boolean algorithm, many relevant documents are being missed altogether, so the ranking formulas have no possibility of giving them high scores. An alternative technique for evaluating ranking formulas is to determine precision after fixed numbers of documents have been examined. This has the advantage that large numbers of relevant documents not identified by the Boolean algorithm do not flatten out recall/precision results. Table 9 gives results at twenty document intervals for all previous experiments.

| Documents | 5 | 15 | 30 | 100 | 200 | Av. |
|---|---|---|---|---|---|---|
| $V_d$ | 0.374 | 0.321 | 0.298 | 0.223 | 0.158 | 0.275 |
| $V_n$ | 0.319 | 0.295 | 0.279 | 0.225 | 0.159 | 0.255 |
| $V_a$ | 0.349 | 0.316 | 0.305 | 0.224 | 0.169 | 0.273 |
| $V_1$ | 0.387 | 0.352 | 0.336 | 0.237 | 0.166 | 0.296 |
| $V_2$ | 0.391 | 0.352 | 0.336 | 0.237 | 0.166 | 0.296 |
| $V_a$ | 0.349 | 0.316 | 0.305 | 0.224 | 0.169 | 0.273 |
| $V_4$ | 0.349 | 0.305 | 0.304 | 0.230 | 0.171 | 0.272 |
| $V_n$ | 0.319 | 0.295 | 0.279 | 0.225 | 0.159 | 0.255 |
| $V_6$ | 0.336 | 0.304 | 0.284 | 0.229 | 0.164 | 0.263 |
| $V_7$ | 0.404 | 0.367 | 0.334 | 0.236 | 0.166 | 0.302 |
| $V_8$ | 0.370 | 0.350 | 0.327 | 0.238 | 0.165 | 0.290 |

Table 9: Comparison of ranking formula for fixed no. of documents returned

There are two observations that one might make after examining both forms of evaluation. The first is that there appears to be a small gain obtained by treating the text in the query in a structured way using the methodology proposed here. The gain is less than 10%. The second result is both surprising and has significant ramifications. The use of adjacent pairs in previous experiments on small collections with relatively short queries provided substantial improvement in recall/precision [3, 12]. We see no such improvement in these experiments.

Finally, we may compare the three Boolean algorithms in their ability to identify appropriate subsets for ranking. In this comparison, we use $V_4$ for ranking each of the subsets. Let $A_c$ represent the algorithm based on concepts, $A_d$ represent the algorithm based on the description, and $A_p$ represent the algorithm that uses adjacent pairs. Table 10 shows recall/precision figures and Table 11 shows precision for fixed number of documents returned.

| Recall | 10% | 20% | 30% | 40% | 50% | 60% | 70% | 80% | 90% | 100% | Av. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $A_c$ | 0.185 | 0.089 | 0.054 | 0.031 | 0.028 | 0.012 | 0.000 | 0.000 | 0.000 | 0.000 | 0.040 |
| $A_d$ | 0.220 | 0.133 | 0.048 | 0.017 | 0.018 | 0.017 | 0.000 | 0.000 | 0.000 | 0.000 | 0.045 |
| $A_p$ | 0.198 | 0.115 | 0.037 | 0.018 | 0.015 | 0.012 | 0.000 | 0.000 | 0.000 | 0.000 | 0.039 |

Table 10: Comparison of Boolean algorithms

| Documents | 5 | 15 | 30 | 100 | 200 | Av. |
|---|---|---|---|---|---|---|
| $A_c$ | 0.328 | 0.319 | 0.306 | 0.221 | 0.169 | 0.269 |
| $A_d$ | 0.349 | 0.332 | 0.311 | 0.222 | 0.149 | 0.272 |
| $A_p$ | 0.349 | 0.305 | 0.304 | 0.230 | 0.171 | 0.272 |

Table 11: Comparison of Boolean Algorithms for fixed no. of documents returned

Using both methods of evaluation, the algorithm that uses key descriptors, without adjacent pairs outperformed augmenting the algorithm with pairs, and the algorithm using concepts. This is despite the fact that $A_c$ returned an average of 1190 documents per query, $A_p$ returned an average of 667 documents and $A_d$ returned only an average of 530 documents.

### 3.3 The Atlas System

The Boolean query generation experiments were performed using the Atlas database system [5]. This system is a nested relational database system designed for text applications. The system was not adapted or tuned in any way to support the TREC experiments. There are a number of signature file organisations that are supported by Atlas—bit slice, two level schemes, and multi-organisation schemes. We implemented an algorithm that analyses the data and selects the most appropriate scheme and optimal parameters for this scheme [11]. For the TREC collection, the multi-organisation scheme was selected.

It took 23 hours to load the database and build the index, which required 313 Mb of disk space. This indexed all stopped and stemmed terms, including adjacent pairs. The text stored in the database was compressed using the mechanism described in Section 2.3.

### 3.4 Combined Database Results

Due to our inability to hold the whole of TREC in the form used for the compression experiments, and the form used for the Atlas system, we ran a further set of experiments using the compressed database system, but using the Boolean algorithm based on concepts to test some of the rank formulas. We performed 4 experiments where we repeated experiments using $V_d$, the description vector, $V_n$ the narrative vector, $V_a$ the vector of all text with structure ignored, $V_1$ where all text was used but structure was taken into consideration, and $V_2$, a modification of $V_1$ where pairs were added as an extra vector. The results are given in Table 12 using recall and precision and then at various intervals based on the number of documents retrieved in Table 13.

| Recall | 10% | 20% | 30% | 40% | 50% | 60% | 70% | 80% | 90% | 100% | Av. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $V_d$ | 0.324 | 0.196 | 0.136 | 0.090 | 0.083 | 0.034 | 0.022 | 0.005 | 0.000 | 0.000 | 0.089 |
| $V_n$ | 0.325 | 0.218 | 0.128 | 0.084 | 0.080 | 0.034 | 0.022 | 0.005 | 0.000 | 0.000 | 0.090 |
| $V_a$ | 0.298 | 0.170 | 0.116 | 0.074 | 0.067 | 0.040 | 0.026 | 0.005 | 0.000 | 0.000 | 0.080 |
| $V_1$ | 0.372 | 0.235 | 0.129 | 0.092 | 0.083 | 0.048 | 0.022 | 0.005 | 0.000 | 0.000 | 0.099 |
| $V_2$ | 0.372 | 0.236 | 0.129 | 0.092 | 0.083 | 0.048 | 0.022 | 0.005 | 0.000 | 0.000 | 0.099 |

Table 12: Ranking all data

| Documents | 5 | 15 | 30 | 100 | 200 | Av. |
|---|---|---|---|---|---|---|
| $V_d$ | 0.417 | 0.431 | 0.404 | 0.349 | 0.294 | 0.379 |
| $V_n$ | 0.370 | 0.408 | 0.407 | 0.353 | 0.301 | 0.368 |
| $V_a$ | 0.383 | 0.391 | 0.379 | 0.329 | 0.279 | 0.352 |
| $V_1$ | 0.447 | 0.472 | 0.438 | 0.377 | 0.312 | 0.409 |
| $V_2$ | 0.447 | 0.475 | 0.438 | 0.375 | 0.312 | 0.409 |

Table 13: Ranking all data

The results we obtained for the smaller collection are again observed with the full 2 Gb of text. However the advantage of using the structure of the queries is slightly more pronounced.

## 4 Summary

In the first line of investigation, we built a compressed retrieval system for around 2 Gb of text that required under 37% of the size of the input text, and was created in just over 15 CPU-hours on a typical well-configured workstation. Retrieval performance for simple techniques such as the cosine measure is fast, and queries can be processed effectively on low-end workstations in seconds. The techniques developed during this first thread of investigation have thus altered the status of 2 Gb information retrieval systems from being 'unpleasantly expensive' to being 'eminently practical'.

In our second thread of research we have seen that the use of the structure of queries has enabled Boolean queries to be generated that have few disjuncts and perform better than more complicated ones. Given the large number of documents that are relevant to many of the queries that have been examined, such algorithms must be subject to reduced performance

when compared to techniques that rank the entire collection. In the light of the parallel experiments carried out using the full collection, it must be questionable as to whether such techniques will survive.

We have seen a minor improvement in ranking as a result of taking the structure of queries into account. A more surprising result however is that pairs do not provide the performance gains that we have seen with much smaller collections. As a result, the vocabulary for a collection would appear to be more manageable than if pairs need to be explicitly described.

## Acknowledgements

We are grateful to Lachlan Andrew, Daniel Lam, and Neil Sharman for assisting with the implementation. This work was supported by the Australian Research Council.

## References

[1] S. Al-Hawamdeh and P. Willett. Comparison of index term weighting schemes for the ranking of paragraphs in full-text documents. *International Journal of Information and Library Research*, pages 116–130, 1990.

[2] T.C. Bell, A. Moffat, C.G. Nevill, I.H. Witten, and J. Zobel. Data compression in full-text retrieval systems. *Journal of the American Society for Information Science.* To appear.

[3] J. Fagan. Automatic phrase indexing for document retrieval: An examination of syntactic and non-syntactic methods. In *Proc. 10'th ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 91–108, 1987.

[4] D. Harman and G. Candela. Retrieving records from a gigabyte of text on a minicomputer using statistical ranking. *Journal of the American Society for Information Science*, 41(8):581–589, 1990.

[5] A.J. Kent, R. Sacks-Davis, and K. Ramamohanarao. A signature file scheme based on multiple organisations for indexing very large text databases. *Journal of the American Society for Information Science*, 41(7):508–534, 1990.

[6] J.B. Lovins. Development of a stemming algorithm. *Mechanical Translation and Computation*, 11(1-2):22–31, 1968.

[7] A. Moffat. Economical inversion of large text files. *Computing Systems*, 5(2):125–139, 1992.

[8] A. Moffat and J. Zobel. Coding for compression in full-text retrieval systems. In *Proc. 2'nd IEEE Data Compression Conference*, pages 72–81, Snowbird, Utah, March 1992. IEEE Computer Science Press.

[9] A. Moffat and J. Zobel. Parameterised compression for sparse bitmaps. In *Proc. 15'th ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 274–285, Copenhagen, Denmark, June 1992. ACM Press.

[10] R. Sacks-Davis, A. Kent, R. Kotagiri, J. Thom, and J. Zobel. A nested relational database for text applications. Technical Report 92-52, Collaborative Information Technology Research Institute, Melbourne, Australia, 1992.

[11] R. Sacks-Davis, A.J. Kent, and K. Ramamohanarao. Multi-key access methods based on superimposed coding techniques. *ACM Trans. on Database Systems*, 12(4):655–696, 1987.

[12] R. Sacks-Davis, P. Wallis, and R. Wilkinson. Using syntactic analysis in a document retrieval system that uses signature files. In *Proc. 13'th ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pages 179–192, September 1990.

[13] G. Salton. *Automatic Text Processing.* Addison Wesley, Massachusetts, 1989.

[14] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, 1983.

[15] J. Zobel and A. Moffat. Adding compression to a full-text retrieval system. In *Proc. 15'th Australian Computer Science Conference*, pages 1077–1089, Hobart, Australia, January 1992.

[16] J. Zobel, A. Moffat, and R. Sacks-Davis. An efficient indexing technique for full-text database systems. In *Proc. 18'th Conference on Very Large Databases*, pages 352–362, Vancouver, Canada, August 1992.

[17] J. Zobel, A. Moffat, and R. Sacks-Davis. Memory-efficient ranking of document collections. Technical Report TR-92-53, Collaborative Information Technology Research Institute, Melbourne, Australia, August 1992.

# Application of the Automatic Message Router to the TIPSTER Collection

by

Richard L Jones,
Sek Kit Leung
and D Lewis Pape

Centre for Electronic Document Research
c/- Computer Power
P O Box 7126
Canberra Mail Centre ACT 2610
AUSTRALIA

**Abstract**

The two experiments undertaken (CPGHC and CPGCN) investigated the applicability of technology developed in the Automatic Message Router Project (AMR) to the TIPSTER data. AMR inverts the queries rather than the data; this is claimed to be an appropriate way of handling document routing given the ephemeral nature of documents in an electronic network. AMR's techniques of automatically identifying good discriminating terms and using the relative position of terms in documents as a means of developing a single relevance score proved to be very effective in producing good results in the experiment.

## Background

The Centre for Electronic Research (CEDR) is based in Canberra, Australia. It is s laboratory in the Australian Computers and Communications Institute (ACCI), an organisation whose primary mission is to act as a link between Australian research in I.T. and the market-place. CEDR is the principal contribution to ACCI by Computer Power Group (CPG), Australia's largest computer services company. It also continues an eight year R and D program by CPG into document analysis and retrieval techniques.

The technology being tested in the TREC experiments has been developed in the Automatic Message Router Project, (AMR). AMR was developed over the past three years extending techniques called IQ, developed earlier by CPG for document retrieval, and incorporated in the STATUS document retrieval product since 1988.

## The AMR Project

The AMR Project set out to develop viable techniques that operate in an electronic mail or wire service environment. In these environments, the roles of query and document are reversed, compared with document retrieval. Users have a relatively long term interest in a topic and wish to receive documents that are relevant to that topic passed to them as soon as possible. However, documents are of short term interest and lose their value rapidly with time, unless they have been routed to someone with a specific interest in them. Of course all documents may be routed to a document retrieval system for more general historical access. AMR reflects this exchange of role of query and document, by inverting the queries (referred to as filters) and passing the documents one at a time against them.

AMR allows filters to be prepared in a structured form, where each filter term represents a set of synonymous terms, or in a plain English statement of the information that is desired. The former technique was used for all the filters used in the experiment. The filters are inverted into memory for performance reasons.

AMR computes the relevance of each document by utilising a set of heuristics that take into account the number of different terms in a filter, and their relative positioning at a paragraph level. Each term is automatically weighted by estimating its effectiveness as a discriminator, i.e. its ability to divide the universe of documents into two groups, those that are relevant and those that are not. From the model of routing defined above a decision on the fate of a document must be made immediately. Thus the universe of documents is not constant but changes as each document is filtered. To handle this dynamic environment, AIDA keeps statistics of the discriminating power of each terms, both with respect to the most recent documents seen and the average over the life-time of the filter. In practice, the weights stabilise after some 40 documents that have some degree of relevance to the filter have been processed. Thereafter, the weights are only changed by a group of documents that predominantly discuss a few aspects of a filter.

## The TREC Experiments

The experiments conducted for TREC had four major objectives:

- to obtain an objective evaluation of AMR on a large document collection;
- to develop as many different filters as possible for each topic to see what sensitivity there was in the AMR heuristic to widely differing filters;
- to investigate AMR's performance and robustness;
- to perform tuning and relevance normalisation on the AMR heuristics.

Four sets of filters were run, though because of time constraints, only two were submitted to the formal experiment set. These two were:

- CPGHC - Hand-crafted structured form of filters. These were written by an experienced staff member over a period of a week.

- CPGCN - Automated structured form, generated directly from the Concept field of each topic. These were generated by a simple LEX program that converted each entry directly into an AMR term.

**Experimental Procedure**

The filters were run against a portion of the Disk 1 data. The primary purpose of this run was to tune the internal weights of the terms in each filter. Because of hardware problems, the filters had been submitted to TREC before these experiments were conducted so no changes were made to the filters in the light of these runs. However, some changes to the AMR algorithms were made to stabilise the dynamic weight modification process described above.

The two sets of filters submitted, together with two other sets generated from the Disk 1 data (200 filters in all) were run against the Disk 2 collection over a weekend.

Though AMR provides a measure of the relevance of each document in the range 1 to 100, no cut-off was applied to the results submitted. If a filter returned 200 documents, then these were submitted, regardless of the scores obtained.

**Results Analysis**

The results of the experiments were very encouraging, with both the manually generated and automatic sets appearing in the top four routing results presented at TREC, as measured by the 11-point average precision vs recall scores.

The manually generated filters performed better than the automatically generated ones though not markedly so. This is probably due to the fact that the Concepts were well described and provided an adequate range of synonyms for most topics. Four of the manually generated filters presented very poor results, due to the insertion of a mandatory term in the filter.

As measured by the precision figures, the filters performed better on the second 25 topics than they did on the first 25. This may be due to the different nature of the topics; the first half of the topics being more fact specific, and the last being more generic and 'information retrieval' oriented. However, the 11 point average recall precision scores do not reflect any significant difference.

The system achieved a throughput of one document a second against 200 filters. This was especially pleasing since the experiments were run on a relatively modest platform, an 80486/33 processor with 8 Mbytes of memory under SCO UNIX.

As measured against the original objectives described above, three of the four have been achieved: AMR techniques have proved to be as accurate as any other in current use, The tests certainly proved that AMR is robust in running a very large collection, its performance was also quite adequate. The system was tuned during the first set of runs, and the experience has proved to be very valuable. The only major objective that remains to be achieved is to analyse AMR's sensitivity to widely differing filter types. Work is proceeding on this.

## References

(1) Jones R.L., Enhanced Retrieval Mechanisms for Free Text Data Bases, Proceedings First Pan Pacific Computer Conference, (1985), pp 134-143

**Centre for Electronic Document Research**

**TREC Routing Experiments CPGHC and CPGCN**

## Systems Summary and Timing

I Construction of Indexes

The software does not invert the text. It inverts the queries (or filters) and passes the text through the combined index formed from the queries.

II Query Construction

D Automatically Built Queries (Routing)

1 Concept field used

2 Time to build query < 5 seconds

3 (a) Terms selected from topic

(b) Terms weighted with weights based on terms from documents with relevance judgements, and dynamically modified through the training set and the test set.

c) Phrases extracted from topics

j) Automatic addition of Boolean connectors and proximity operators from topics.

E Manually constructed queries (routing)

1 All topic fields used

2 Average time to build query 30 minutes

3 Query builder system expert

4 Data used to build query from topic

5 The creation of the query uses Boolean operators, and proximity operators.

III Searching

A Total Computer Time

One message through 200 filters per second. This includes searching and ranking.

B Method

Software uses a fuzzy AND and Proximity measure to rank documents.

C Factors included in ranking

1 Term frequency

5 Position in document

7 Proximity of terms

IV Machine

The experiments were run on an HP 486/33 with 8 Mbytes under SCO UNIX. The CD ROM drive was accessed via NFS.

V AMR Software

AMR is commercial strength software developed by Computer Power Group. Its commercialisation software engineering phase took some three person years.

# CLARIT TREC Design, Experiments, and Results

David A. Evans, Robert G. Lefferts, Gregory Grefenstette,
Steven K. Handerson, William R. Hersh, Armar A. Archbold

Laboratory for Computational Linguistics
Department of Philosophy
Carnegie Mellon University

## 1 Introduction

This report presents an abbreviated description[1] of the approach and the results of the CLARIT team in completing the tasks of the "Text Retrieval Conference" (TREC) organized by the National Institute of Standards and Technology (NIST) and the Defense Advanced Research Projects Agency (DARPA) in 1992.[2]

### 1.1 A Characterization of the TREC Tasks

TREC activities required participants to 'retrieve' 200 documents for each of 100 different 'topics' from a large database of full-text documents. Each topic was given as a one-page description of an item of interest. This feature of the TREC tasks was somewhat unusual, at least compared to many traditional 'bibliographic'-retrieval evaluations, in which the topic or 'query' is a minimal, often telegraphic, single-phrase statement of a 'subject' or 'an interest'.[3] However, the principal distinguishing features of the TREC tasks were (1) their scale—involving a total of approximately 2 gigabytes of text, representing approximately 750,000 full-text documents of varying length—and (2) the careful attention of the organizers in evaluating the results submitted by each participating group.

More specifically, TREC tasks were designed to simulate two general types of information 'retrieval' situations, "routing" and "ad-hoc" querying. "Routing" corresponds to situations in which a topic is possibly well documented (e.g., with examples) and the user desires to find more similar documents. In the case of TREC tasks, 50 topics were designated as "routing" topics; each was accompanied by a set of documents judged to be "relevant" to the topic.[4] The first installment of the full set of documents, representing approximately 1.1-gigabytes of text, was available to each team for use in identifying possible other relevant documents for each

[1] A more complete and detailed description of the CLARIT–TREC activities and results is available as a technical report, [Evans et al. in preparation].

[2] The TREC activities were organized at the end of 1991. Data was made available in the Spring of 1992. All processing results were submitted by September 1, 1992, to NIST. The "Conference" itself—a Workshop involving the approximately two dozen groups that submitted partial or full processing results—took place on November 4-6, 1992, in Rockville, MD.

[3] The longer statements of 'topics' in TREC were arguably more interesting as a test of systems and more representative of many contemporary information-seeking situations. See Figure 9 for a sample topic statement.

[4] The number of sample relevant documents varied greatly from topic to topic. Some topics had almost 100 sample relevants; other had only about ten.

routing topic. The rules of the exercise required each group to submit 'models' for each routing topic (e.g., a set of procedures or a 'query vector'), which then were 'on record' and had to be used in the final evaluation of the routing task. That evaluation required that each group retrieve documents from the second installment of documents, approximately 0.9-gigabytes of text. "Ad-hoc" querying corresponds to situations in which a topic is presented to a system and appropriate documents must be found; no example documents are available. In TREC, the second 50 topics were designed as "ad-hoc-query" topics. The rules required each group to use the full 2-gigabyte database as the search space for ad-hoc queries. All results were reported as a ranked list of the 200 'top' documents in response to each topic, whether a routing topic or an ad-hoc-query topic.

### 1.2 Notes on CLARIT Team Participation

The CLARIT team submitted results, labeled "A" and "B",[5] representing the top 200 documents at the end of each of two sequential steps in the processing of topics. Since the actual processing of topics was designed to give 'best' results only after both stages of processing were completed, the "A" results are known to be suboptimal; the "B" results represent the true test of the CLARIT-TREC design.

The large scale of the tasks challenged the resources that were available to the CLARIT team. Storage for the source data and topics alone required 2 gigabytes of space. The research-prototype version of the CLARIT system, which was used in the task, generates various secondary and intermediate resources in the course of processing. Such intermediate files also require temporary storage. In all, approximately 8 gigabytes of disk space was used for the process. The system-engineering work required to manage the data represented a significant effort for the team; more than 75% of the team effort was devoted to (a) re-implementing critical CLARIT processes to deal with larger volumes of data and limited space and (b) monitoring and directing the use of resources and the sequence of processes when making actual 'runs' over the data.

Data for the final tests was made available from NIST only after preliminary processing results were submitted. The CLARIT team submitted its preliminary results (the 'frozen' forms of the routing queries) on Friday, August 21, 1992. NIST express-mailed the new test data to Carnegie Mellon on the same day, but the package was misaddressed and did not arrive. A second mailing finally did arrive on Tuesday, August 25, one week before the deadline for final results. Thus, all final processing took place in seven days.

The CLARIT team utilized, variously, six machines (including a DECsystem 5820 and DECstation 5000s and 3100s) and the approximately 8 gigabytes of dedicated storage for TREC-processing tasks. Actual processing occurred in batch mode over several machines and across a network (as some storage was remote).

## 2 Background Description of Basic CLARIT Processing in TREC

Basic CLARIT processing is described elsewhere.[6] A schematic representation of the 'standard' CLARIT process for *document indexing* is given in Figure 1. A representation of the simplified CLARIT process that was employed in the case of CLARIT-TREC document indexing is given in Figure 2.

---

[5] The Conference provided a special category ("Category B") for groups that intended to work only with a subset (100 megabytes) of the TREC data. This should not be confused with what we call the CLARIT "A" and "B" results: all CLARIT processing involved the full set of TREC data.

[6] Cf. [Evans 1990], [Evans et al. 1991a,b,c].

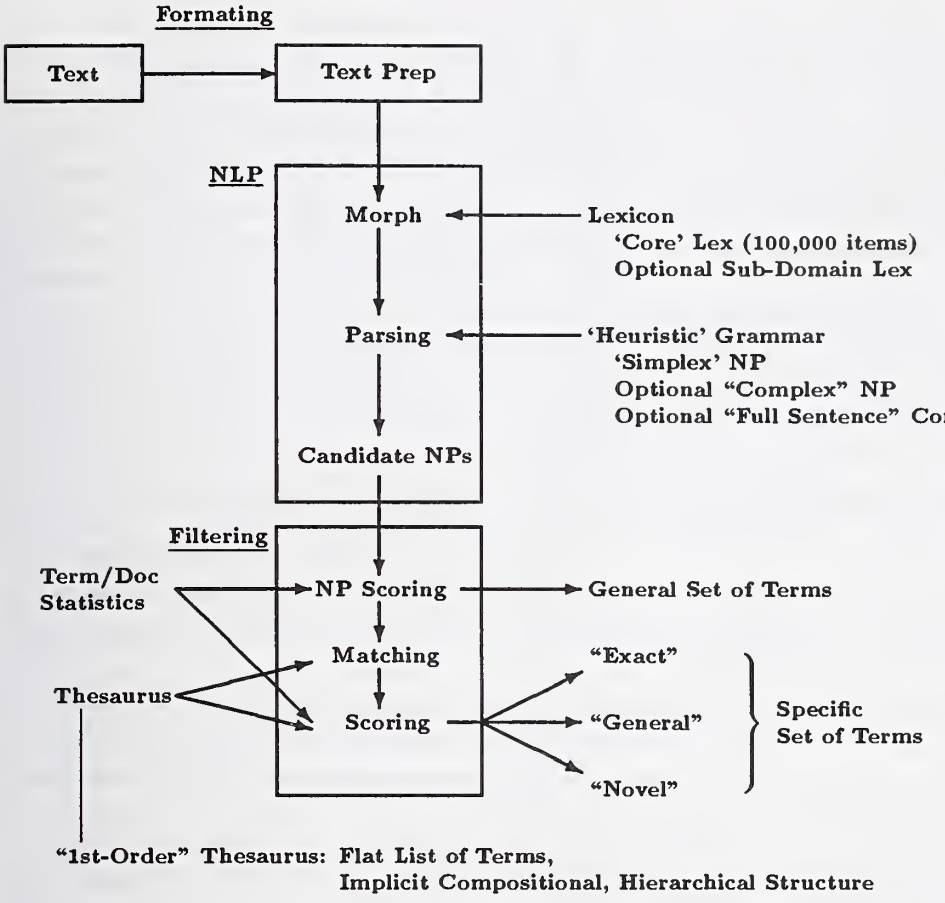



Figure 1: 'Standard' CLARIT Indexing Overview

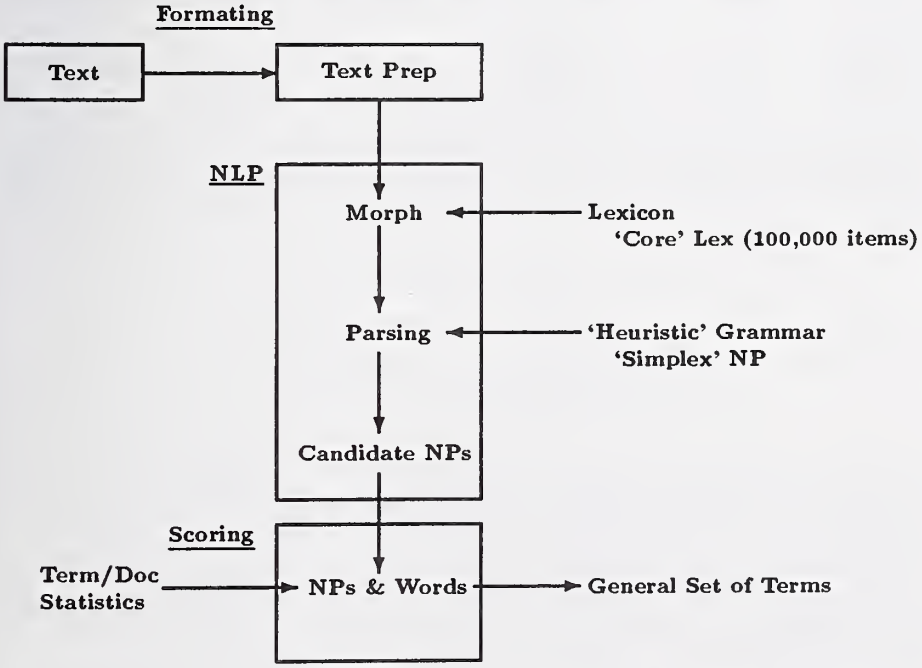



Figure 2: Modified CLARIT Indexing in TREC

### 2.1 Selective NLP to Nominate Information Units

In brief, the CLARIT indexing process as shown in Figure 1 involves several steps, one of which utilizes selective natural-language processing (NLP) to identify noun phrases (NPs) in texts, which are taken as the relevant information units in all further processing. Subsequent steps take advantage of several statistical measures of 'importance' to evaluate NPs as potential index terms. One special feature of CLARIT processing is the use of an automatically-generated 'first-order' thesaurus for a domain to support the selection of appropriate terms. The standard CLARIT process returns three categories of index terms, corresponding to terms that (1) occur in the document and exactly match terms in the thesaurus, (2) terms that are in the thesaurus and are more general than near-matching terms in the document, and (3) terms that are 'novel' to the document and not found in the thesaurus. In addition to being categorized as an *exact*, *general*, or *novel* index term, each term is given a numerical relevance weight deemed to reflect its relative value in characterizing the contents of the document.

### 2.2 'Thesaurus Discovery' to Nominate Sets of Terms for Collections

First-order thesauri are 'discovered' via another CLARIT process, distinct from indexing. The process requires a sample of documents representing a 'domain'. The sample must be moderately large (e.g., minimally 2 megabytes of text) and must be composed of documents that are more or less 'about' the topic of the domain.[7]

In general, CLARIT 'thesaurus discovery' comprises algorithms and techniques for clustering phrases in collections of documents to construct first-order thesauri that optimally 'cover' an arbitrary percentage of all the terminology in the domain represented by the document collection. 'Normal' thesaurus discovery involves (1) decomposition of candidate NPs from the documents to build a term lattice in which nodes are organized hierarchically from words to phrases based on the number of phrases subsumed by the term associated with each node and (2) selection of nodes that have high subsumption scores and that also satisfy certain structural and statistical characteristics (such as being legitimate NPs, well distributed in the corpus, and relatively uncommon in general English). Terms thus selected represent a subset of vocabulary that accurately characterizes the domain. Thesaurus discovery is quite fast[8] and typically yields a subset of terminology that represents less than 5% of all the available terms in the corpus.[9]

Since the TREC experiments involved a heterogeneous collection of documents and since it was not possible to identify specific subsets of documents in the database as 'about' one or another topic, it was not possible to discover and use relevant thesauri in TREC tasks. Thus, as shown in Figure 2, the simplified CLARIT indexing process in TREC tasks did not involve 'matching' of terms against a first-order thesaurus and did not result in three-way-categorized index terms.

---

[7] An example of an appropriate sample might be 50 full-text articles involving "AIDS Research"; or 2,000 abstracts about "Silicon Engraving"; or even one's personal file of recent e-mail correspondence, provided it is sufficiently large and topically coherent.

[8] At present, using the CLARIT research system, a thesaurus can be found for a 3-megabyte corpus in less than 10 minutes on a DECstation 5000/200.

[9] In fact, the *number* of terms returned will vary depending on parameters the user selects when generating the thesaurus.

### 2.3 Vector-Space 'Similarity' Measures

The principal method used by the CLARIT system in comparing 'information objects' (e.g., in retrieval, in routing) is vector-space distance.[10] The basic metric is that of 'similarity' of terms. 'Similarity' is determined by different procedures in different contexts. Partial or 'fuzzy' matching of terms is facilitated by noting whether terms share words or attested subphrases. For example, in vector-space modeling of documents, the contained words of all terms (in the document vector as well as the query vector) are broken out, giving, in effect, the possibility of matching parts of terms, though, technically, the individual words are realized as independent dimensions of the term space.[11]

### 2.4 Notes on the Limited Version of CLARIT Processing in TREC

Because of the time and space limitations in the task, the CLARIT team did not utilize several features of CLARIT processing that normally produce enhanced results. One of the features—the automatic 'tokenization' or identification of proper names—would certainly have assisted processing of some topics. Another feature—the identification of equivalence classes of terms—also would have aided the task.

In addition, no attempt was made to establish 'uniform-length' documents or sub-documents (e.g., by setting a maximum word count or sentence length for such units). Though CLARIT processing supports the treatment of documents as sub-document collections, that feature of CLARIT processing was not utilized in the experiments.

All topic statements were treated uniformly and simply: no attempt was made to handle implicit or explicit quantification, time intervals, satisfaction conditions, etc., except as literally encoded in the topics.

Though CLARIT NLP modules can produce full sentence analyses or complex-NP analyses, neither of these features was utilized in TREC processing. All documents were processed only for simplex NPs; inevitably, some non-NP information was lost.

In indexing TREC documents, term weights were based on a general IDF–TF score[12] for topic 'domains'. In the case of multi-word terms (the norm), the full terms are assigned an independent IDF–TF score, and each word in the term was broken out and assigned an independent IDF–TF score.

While all CLARIT processing is designed to be fully automatic, we did not employ fully automatic processing in TREC tasks. In particular, there were two steps in the CLARIT–TREC process that required non-automatic processing: (1) initial review and weighting of the index terms automatically-nominated and derived from each topic statement and (2) review of first-pass retrieved documents to identify 5–10 relevant ones for 'feedback'. The two steps involved minimal user intervention (and, in fact, required very little time and effort); however, they do qualify the CLARIT–TREC system as a "manual process".

In general, we regard the CLARIT–TREC system as a minimal system for purposes of evaluation. The results of CLARIT–TREC processing are useful in helping us establish *baseline* performance for core but abbreviated CLARIT functions.

---

[10] Cf. [Salton & McGill 1983] for background on vector-space modeling in information retrieval applications.

[11] Cf. [Evans et al. 1992] and [Hersh et al. 1992] for an evaluation of CLARIT vector-space 'similarity' measures.

[12] "IDF–TF" represents the standard *inverse document frequency* × *intradocument term frequency* score for terms.

## 3 Overview of CLARIT–TREC Processing

There were three major phases of processing for the CLARIT–TREC retrieval experiments. Initially, the entire corpus, along with the topic statements, was parsed to extract candidate NPs via CLARIT NLP. In the special case of topics, the candidate NPs were manually reviewed and evaluated to produce weighted query terms. Second, the entire corpus (in noun phrase form) was passed through a quick, and somewhat rough, ranking procedure that was designed to nominate a large subset of documents for further analysis. This step is referred to as "partitioning". A "partitioning thesaurus", or list of weighted, representative terminology was automatically created for each topic. In the final phase of processing, referred to as "querying", a "query vector" was produced for each topic. The query vector was used to retrieve (= rank) documents in the selected partition for the topic using a vector-space 'similarity' metric. The details of these phases of processing are presented below, along with a discussion of different techniques used for "routing" and "ad-hoc" queries.

### 3.1 Design Philosophy—"Evoke" and "Discriminate"

In approaching the principal TREC task of returning 200 ranked documents for each topic, we used a two-stage processing strategy, illustrated in Figure 3. The first stage of processing was designed to identify candidate documents that seemed likely to contain information related to a topic. Of course, since the topic was represented as a set of weighted terms, this step involved scoring each document based on the set of terms. Because this step involved scoring every document in the database against every topic, it was important to design the scoring procedure so that it was not computationally expensive. In fact, it was based on summing the value and number of 'hits' between the topic's set of terms and the terms (NPs) in each document and was expected to result in an over-generated set of candidate documents. The highest-scoring documents were retained as a candidate 'partition' of the database with respect to the topic. The second stage was designed to find the subset of documents in each partition that best matched the topic. In theory, greater (= more discriminating) processing resources could be devoted to this second-stage task, as the total number of documents involved was small compared to the whole collection.

In practice, as illustrated in Figure 3, partitioning resulted in a set of 2,000 ranked documents. The top 200 documents from the partition were submitted to NIST as the CLARIT "A" set of results. Final querying or 'discrimination' among the documents in each partition yielded another, more accurately ranked set of 200 ranked documents, which were submitted as the CLARIT "B" results.

### 3.2 Overview of the Task

As Figure 4 shows, different portions of the total TREC database were used for the "routing" and "ad-hoc" phases of the experiment. The routing task required 'training' of the first fifty topics on the first set of data (represented as the darkened block in Figure 4). In the second step of processing, the partitioning and query vectors that were derived from step one were used to identify, first, 2000-document partitions in the second set of data (represented as a light block in Figure 4) and, second, the top-200 ranked documents in each partition. The ad-hoc query task involved the whole database, but the CLARIT team actually used the first set of data for a preliminary retrieval of documents (based on partitioning). A few (5–10) of the top 20–50 were chosen by quick manual inspection to supplement the query vector and then a second automated round of partitioning over the total database was performed. The final top-200 ranked documents ultimately derived from these second-pass, 2000-document partitions.

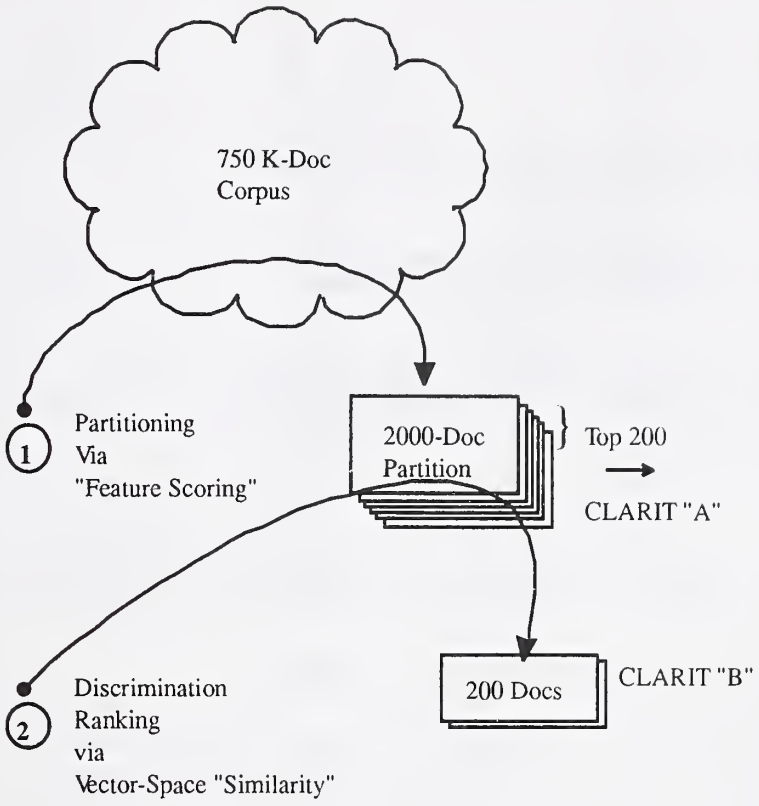



Figure 3: Overview of CLARIT-TREC Processing

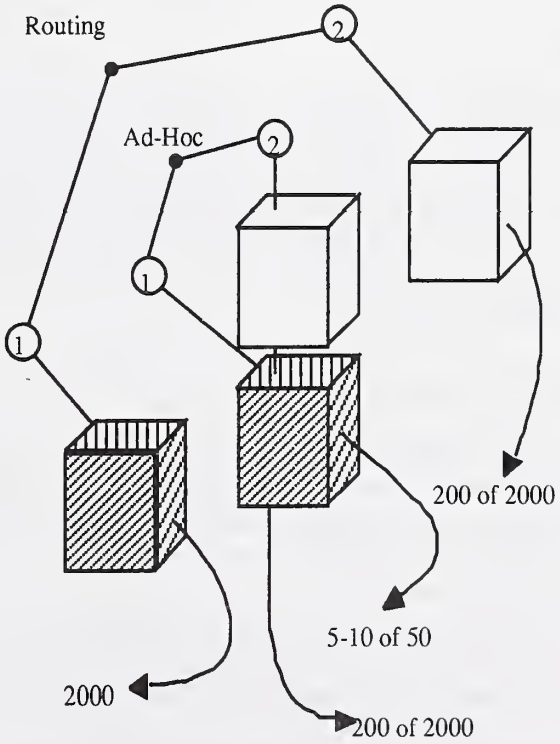



Figure 4: Overview of Processing for "Routing" vs. "Ad-Hoc" Queries

```
\*
\#
WSJ891102-0187
\#
\!
McDermott International Inc. said its Babcock & Wilcox
unit completed the sale of its Bailey Controls Operations
to Finmeccanica S.p.A. for $295 million.
\@
Finmeccanica is an Italian state-owned holding company
with interests in the mechanical engineering industry.
\@
Bailey Controls, based in Wickliffe, Ohio,
makes computerized industrial controls systems.
It employs 2,700 people and has annual revenue
of about $370 million.
\@
\!
\*
```

Figure 5: Sample of Data—Document After Text Formating

## 4 Details of the CLARIT–TREC Experiments

Both "routing" and "ad-hoc" query experiments took advantage of basic CLARIT processing. There are several features the two experiments share. The experiments are distinct in that "routing" involved a special step of creation of a partitioning thesaurus using larger sets of supplied relevant documents and "ad-hoc" queries involved partitioning the document set once using only automatically derived (but manually weighted) query terms and choosing a small set of relevant documents to expand the final query vector.

### 4.1 Preparing Data

Each TREC document had to be formated for CLARIT processing. This involved making the unique text ID accessible to CLARIT as a special field and delimiting the beginning and end of each text in a file. Figure 5 gives a sample formated document. As can be seen in the sample, the beginning and end of the record is marked by a backslash followed by "*". The unique ID is set off by a backslash followed by "#". The beginning and end of the text of the document is marked by a backslash followed by "!". Each paragraph is separated from the next by a backslash followed by "@".[13]

### 4.2 Processing TREC Corpora (NLP)

Figure 6 gives a schematic representation of the processing steps that occurred subsequent to data formating. The process labeled "NLP" in the figure includes all the steps illustrated in the "NLP" portion of Figure 2: morphological analysis of words and parsing for simplex NPs. Simplex NPs were extracted for all TREC documents; words were morphologically normalized.[14]

[13]Though CLARIT data preparation demarks paragraph units, the CLARIT–TREC process did not distinguish divisions of text at this level. For CLARIT–TREC purposes, all the text between the "!"-marks was used as the source of information about a document. Thus, longer and shorter documents were treated uniformly as 'unit' texts.

[14]The manually-supplied keywords attached to some TREC documents in a "keyword field" were discarded.

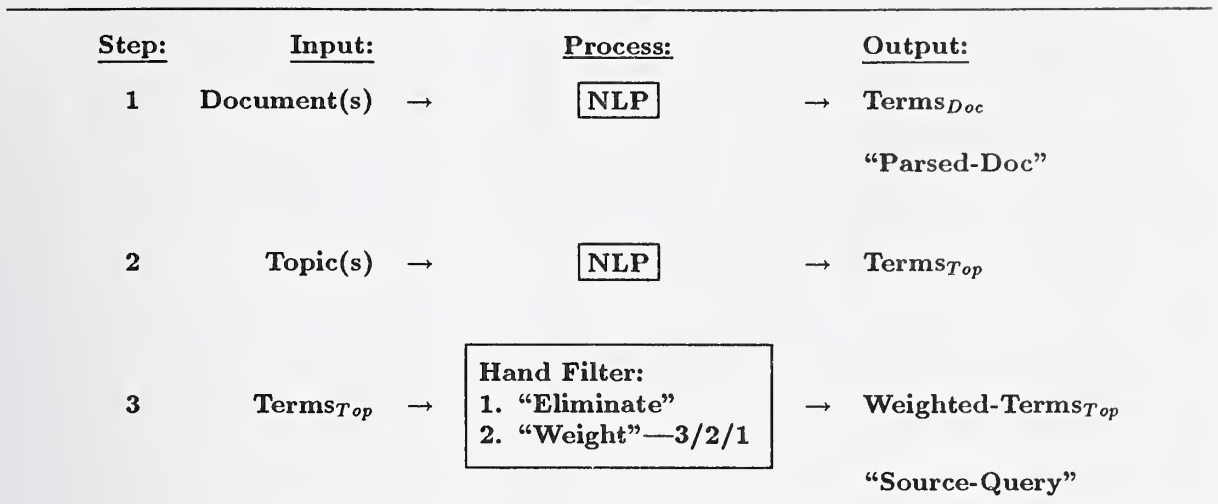



Figure 6: Schematic Representation of Data Preparation

```
\*
\#
WSJ891102-0187
\#
\!
"mcdermott" na mcdermott
"international" adj international
"inc." ukw? inc.
"said" vt-past say vt-pastprt say
"its" gen its
"babcock" na babcock
"\&" *and* and
"wilcox" na wilcox
"unit" sn unit
"completed" vt-past complete vt-pastprt complete
"the" d the
"sale" sn sale sn sell
"of" prep of
"its" gen its
"bailey" sn bailey
"controls" vt-pressg3 control pn control vt-pressg3 control
"operations" pn operation
...
"of" prep of
"about" prep about
"$370" ukw? $370
"million" quant million
"\." *period* \.
\$
\@
\!
\*
```

Figure 7: Sample of Data—Document After Morphological Analysis

```
\*
# 1
\#
WSJ891102-0187
\#
\!
-1 (mcdermott) () 0
-1 (international inc.) () 0
-1 (babcock) () 0
-1 (wilcox unit) () 0
-1 (sale) () 0
-1 (bailey control operation) () 0
-1 (finmeccanica s.p.a.) () 0
-1 ($295) () 0
\@
-1 (finmeccanica) () 0
-1 (italian state) () 0
-1 (owned holding company) () 0
-1 (interest) () 0
-1 (mechanical engineering industry) () 0
\@
-1 (bailey control) () 0
-1 (wickliffe) () 0
-1 (ohio) () 0
-1 (computerized industrial control system) () 0
-1 (employ) () 0
-1 (people) () 0
-1 (annual revenue) () 0
-1 ($370) () 0
\@
\!
\*
```

Figure 8: Sample of Data—Document After NP Extraction

A sample of a document after morphological analysis is given in Figure 7. A sample of the same document after simplex-NP extraction is given in Figure 8. Note that "owned holding company" and "$295" or "$370" are treated as NPs along with legitimate phrases like "computerized industrial control system". While CLARIT does have facilities to discover and eliminate inappropriate participles (such as "owned" in isolation) and can recognize nonce adjectives, such as "state-owned", such processing was not employed in the TREC tasks. Hence, the correct expression, "Italian state-owned holding company" was *not* found or used in this case. In addition, as noted previously, the CLARIT–TREC system did not 'tokenize' company names or dates or other 'regular-expression'-like phrases; there was no time in our schedule for such processing.

All NLP (and other) processing steps were piped through the system; intermediate files were not retained. The parsed representation of all the texts took up approximately 98% of the space occupied by the original text. Intermediate (but unretained) files generated in CLARIT processing included a file of the words in each text, in their original order, annotated with morphological categories. Other files contained the output of the parser as a list of NPs in the order in which they occurred in each text. The parsed representation of the text was retained and used at all subsequent steps of processing. Indeed, hereafter, unless otherwise specified, any reference to a document or collection of documents refers to the CLARIT representation of the text, viz., a sequence of normalized simplex NPs.[15]

### 4.3 Identifying Terms from Topics

All fields of topic statements, such as given in Figure 9, were similarly processed for NPs. Team members reviewed the NPs and assigned weights of "1", "2", or "3" to each NP according to whether the term was central or peripheral to the topic. (Some extracted NPs were discarded as irrelevant or ill-formed; the vast majority were retained.) A sample set of weighted terms for the topic in Figure 9 is given in Figure 10. The manual review and weighting of terms from the topic statement took less than 5 minutes per topic. All subsequent processing of the query was performed automatically.

### 4.4 Establishing Sets of 'Relevant' Documents

Given the need to 'evoke' candidate documents and to 'partition' the database into subsets that were easier to manage, we were naturally interested in identifying features in the topics that would be useful as discriminators. We had little confidence, however, that the specific terms in topics, which constitute the "source query", were either most respresentative of the domain of the topic (= the 'satisfaction class') or reasonably comprehensive. We thus decided to supplement the source query with additional terms.

In particular, we used the CLARIT thesaurus-discovery technique on known relevant documents to identify terminology that might be better representative of the satisfaction-class documents than the source query alone. The process produced a list of terms from the available topic-relevant documents (or from a small sample of relevent documents that we may have found) and automatically nominated the top (approximately 20%) ranked terms to supplement the original query (as derived from the topic statement) to produce a "routing/partitioning thesaurus" for the topic.

Since the routing topics already had accompanying relevant documents, we used these as a source of additional terminology. Ad-hoc queries, on the other hand, had no associated relevant documents, so we designed a preliminary, partial 'retrieval' step that would help us

[15] From the point of view of the CLARIT system, the information in a document is entirely represented by the extracted noun phrases.

<top>
<head> Tipster Topic Description
<num> Number: 057
<dom> Domain: U.S. Economics
<title> Topic: MCI
<desc> Description:
Document will discuss how MCI has been doing since the Bell System breakup.
<narr> Narrative:
A relevant document will discuss the financial health of MCI Communications Corp. since the breakup of the Bell System (AT&T and the seven regional Baby Bells) in January 1984. The status indicated may not necessarily be a direct or indirect result of the breakup of the system and ensuing regulation and deregulation of Ma Bell or of the restrictions placed upon the seven Bells; it may result from any number of factors, such as advances in telecommunications technology, MCI initiative, etc. MCI's financial health may be reported directly: a broad statement about its earnings or cash flow, or a report containing financial data such as a quarterly report; or it may be reflected by one or more of the following: credit ratings, share of customers, volume growth, cuts in capital spending, $$ figure net loss, pre-tax charge, analysts' or MCI's own forecast about how well they will be doing, or MCI's response to price cuts that AT&T makes at its own initiative or under orders from the Federal Communications Commission (FCC), such as price reductions, layoffs of employees out of a perceived need to cut costs, etc. Daily OTC trading stock market and monthly short interest reports are NOT relevant; the inventory must be longer term, at least quarterly.
<con> Concept(s):
1. MCI Communications Corp.
2. Bell System breakup
3. Federal Communications Commission, FCC
4. regulation, deregulation
5. profits, revenue, net income, net loss, write-downs
6. NOT daily OTC trading, NOT monthly short interest
<fac> Factor(s):
Time: after January 1984
</fac>
<def> Definition(s):
</top>

Figure 9: Sample of Data—Topic 57

```
\*
\#
 057
\#
\!
...
2 (bell system breakup) () 0
...
2 (capital spending) () 0
2 (cash flow) () 0
2 (credit rating) () 0
2 (customer) () 0
...
1 (ma bell) () 0
3 (mci communication corporation) () 0
3 (mci financial health) () 0
1 (mci initiative) () 0
2 (mci) () 0
2 (net income) () 0
2 (net loss) () 0
1 (order) () 0
2 (pre tax charge) () 0
2 (price cut) () 0
2 (price reduction) () 0
2 (profit) () 0
2 (quarterly report) () 0
1 (regional baby bell) () 0
...
1 (telecommunication technology) () 0
...
\!
\*
```

Figure 10: Sample of Data—Hand-Weighted Term-Set for Topic 57

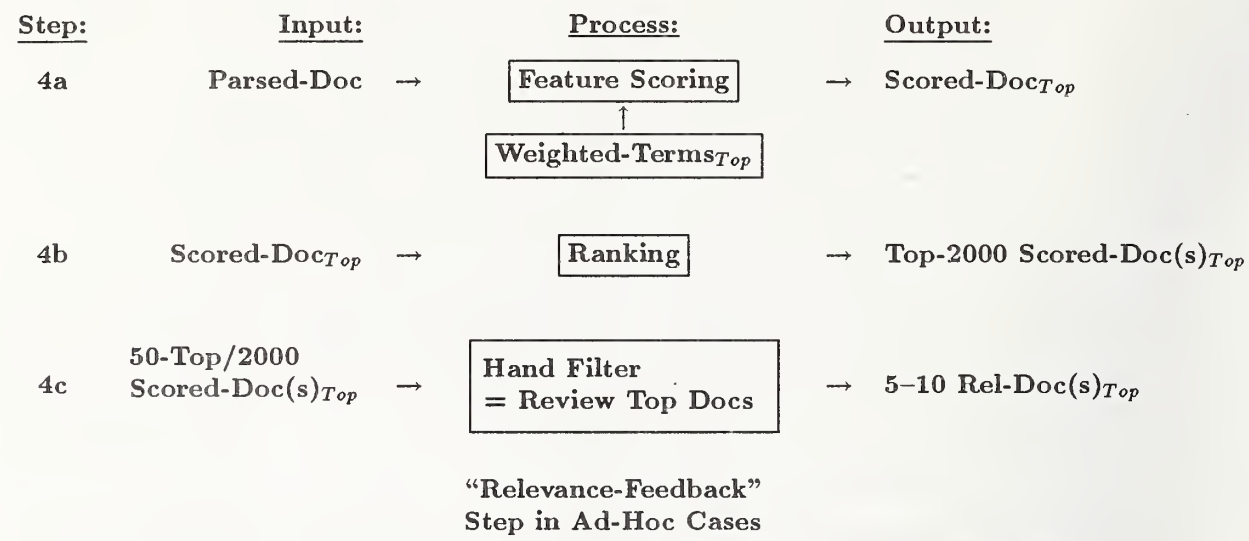



Figure 11: Schematic Representation of Processing When 'Relevant' Documents are not Given

find candidate relevant documents. In practice, this required a partitioning of a sample of data and a review of the returned top-ranked documents. This phase of processing is illustrated in Figure 11.

As shown in Figure 11, Step 4a, the weighted, relevant terms were taken as a query vector representing a subset of positive instances of concepts in the equivalence class of the topic. In the case of ad-hoc querying, the query vector was used to identify a sample of 50 candidate documents from a subset of the corpus, which were reviewed in rank order by team members until 5–10 'true' relevant documents were identified (Step 4c). This can be regarded as a 'relevance-feedback' step in the querying process. In the case of routing, the sample of 'true' relevants provided by the TREC organizers was accepted as valid and no review was performed.

### 4.5 Using Relevant Documents to Create 'Partitioning Thesauri'

As indicated in Figure 11 Step 4d, the 'authoritative' set of relevant documents was processed with CLARIT 'thesaurus-discovery' modules to produce a set of terms that (arguably) bear some relation to the topic. We refer to the output of this process as a "pseudo-thesaurus". The actual routing/partitioning thesaurus was generated by CLARIT by combining the set of weighted terms for the topic with the pseudo-thesaurus, as shown in Step 5. Note that partial noun phrases, derived from pseudo-thesaurus entries, and attested in the documents, were also added to the routing/partitioning thesaurus with a partial score.

As illustrated in Figures 13 (and Figure 14), the partitioning thesaurus itself is a list of terms, where each term has an associated vector of information specifying its importance in any number of topics. In the case illustrated for Topic 57, for example, the term "bell system breakup" has the triple "<057 1 2.0>" associated with it. The "057" indicates that the term is relevant to Topic 57; the "1" indicates that the term is a full term (not an attested sub-phrase of a term); and the "2.0" gives the term's relative weight or importance (in this case, reflecting the score that was assigned by hand).

### 4.6 'Feature Scoring' to Partition Documents

Figure 14 gives a portion of the composite or 'super thesaurus' for all 100 topics. Each

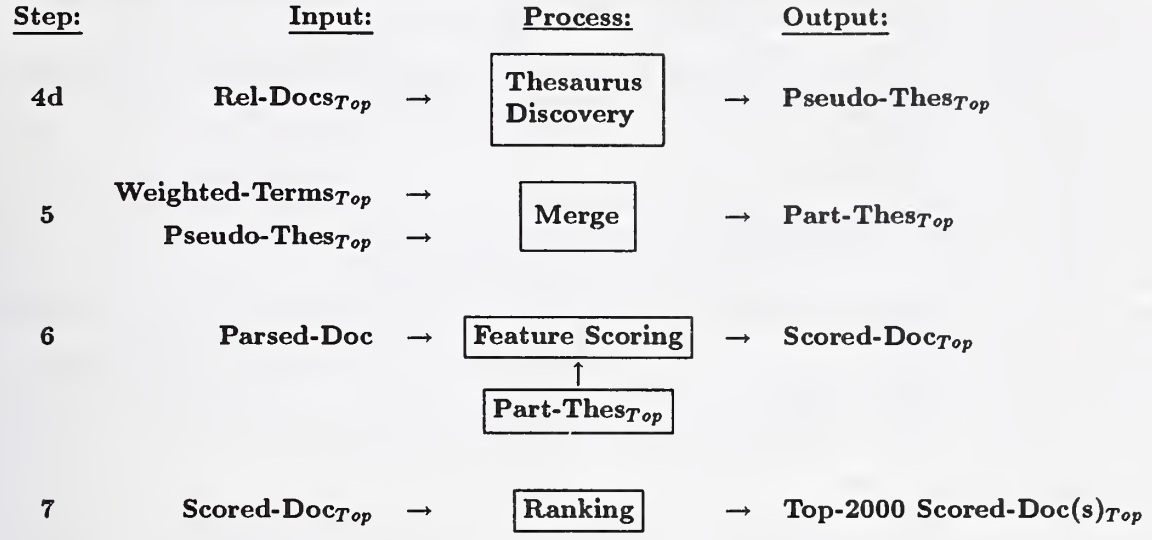



Figure 12: Schematic Representation of Processing When 'Relevant' Documents are Available

```
advance | <057 1 1.0>
at&t | <057 1 1.0>
bell system breakup | <057 1 2.0>
bell system | <057 1 1.0>
bell | <057 1 1.0>
breakup | <057 1 1.0>
broad statement | <057 1 1.0>
capital spending | <057 1 2.0>
cash flow | <057 1 2.0>
credit rating | <057 1 2.0>
customer | <057 1 2.0>
cut cost | <057 1 1.0>
cut | <057 1 1.0>
deregulation | <057 1 1.0>
direct indirect result | <057 1 1.0>
...
ma bell | <057 1 1.0>
mci communication corporation | <057 1 3.0>
mci financial health | <057 1 3.0>
mci initiative | <057 1 1.0>
mci | <057 1 2.0>
net income | <057 1 2.0>
net loss | <057 1 2.0>
order | <057 1 1.0>
...
telecommunication technology | <057 1 1.0>
united states economics | <057 1 1.0>
volume growth | <057 1 2.0>
```

Figure 13: Sample of Data—1,201-Term Partitioning Thesaurus for Topic 57

```
advance | <057 1 1.0> <065 1 0.30> <075 1 0.30> <076 1 0.30>
american telephone | <057 1 0.30>
analyst | <054 1 0.30> <055 1 0.50> <057 1 0.50> <074 1 0.50> <080 1 0.50> <082 1 0.50> <088 1 0.50>
announcement | <057 1 0.30>
at&t cut | <057 1 0.50>
at&t price | <057 1 0.50>
at&t | <057 1 1.0>
bell system breakup | <057 1 2.0>
bell system | <057 1 1.0>
bell | <057 1 1.0>
benefit | <057 1 0.30> <060 1 0.30> <073 1 0.50> <074 1 0.50> <075 1 0.50> <088 1 0.30> <099 1 0.30>
breakup | <057 1 1.0>
broad statement | <057 1 1.0>
business customer | <057 1 0.50>
capital spending | <057 1 2.0>
...
jack grubman | <057 1 0.50>
...
late price cut | <057 1 0.30>
layoff | <057 1 1.0>
least quarterly | <057 1 2.0>
local phone company | <057 1 0.50>
local telephone company | <057 1 0.30>
long distance carrier | <057 1 0.30>
long distance telephone rate | <057 1 0.30>
ma bell | <057 1 1.0>
margin | <057 1 0.30>
market share | <057 1 0.30> <089 1 0.30>
mci communication corporation | <057 1 3.0>
mci communication | <057 1 0.50>
mci earning | <057 1 0.50>
mci executive | <057 1 0.50>
mci financial health | <057 1 3.0>
mci initiative | <057 1 1.0>
mci move | <057 1 0.50>
mci official | <057 1 0.50>
mci price | <057 1 0.50>
mci spokesman | <057 1 0.50>
mci | <057 1 2.0>
...
result | <057 1 1.0> <070  1 1.0> <081 1 2.0>
revenue | <057 1 1.0> <081 1 2.0> <053 1 0.30> <054 1 0.30> <093 1 0.30>
rising cost | <057 1 0.30>
...
telecommunication technology | <057 1 1.0>
telegraph company | <057 1 0.30>
telegraph | <057 1 0.30>
united states economics | <057 1 1.0> <072 1 1.0>
united telecommunication inc. | <057 1 0.50>
united telecommunication | <057 1 0.50>
...
washington based mci | <057 1 0.50>
washington based telecommunication concern | <057 1 0.50>
william e conway jr. | <057 1 0.50>
...
```

Figure 14: Sample of Data—15,287-Term 'Super' Partitioning Thesaurus

- *Data from Partitioning Thesaurus:*

| | |
|---|---|
| **Thes_Weight(term):** | Real number weight assigned to term in partitioning thesaurus |
| **Thes_Whole(term):** | Boolean value indicating that term is a whole term in the thesaurus (1) or an attested sub-term of a whole term (0) |

- *Data from Document Text:*

| | |
|---|---|
| **Tot_Terms:** | Number of terms in a document |
| **Num_Terms:** | Number of unique terms (or sub-terms) found in document that match terms (or sub-terms) in the partitioning thesaurus |
| **Term_Freq(term):** | Frequency of term in document |
| **Term_Length(term):** | Number of words in term |
| **Text_Whole(term):** | Boolean value indicating that term is a whole term in the text (1) or an attested sub-term of a whole term (0). |

Figure 15: Feature Matching Score (Partitioning) Input Data

$$\textbf{Doc\_Score} = \frac{\sum_{i=1}^{\text{Num\_Terms}} \text{Term\_Score}(\text{term}_i)}{\ln(\text{Tot\_Terms} + 1.72)} \times \left[ \frac{\sum_{i=1}^{\text{Num\_Terms}} \text{Term\_Freq}(\text{term}_i)}{(\text{Tot\_Terms} + 1)} \right]^2$$

$$\textbf{Term\_Score(term)} = \begin{cases} \text{Raw\_Score(term)} & \text{if } (\text{Thes\_Whole(term)} = 1) \wedge (\text{Text\_Whole(term)} = 1) \\ \frac{\text{Raw\_Score(term)}}{4.0} & \text{if } (\text{Thes\_Whole(term)} = 1) \wedge (\text{Text\_Whole(term)} = 0) \\ \frac{\text{Raw\_Score(term)}}{8.0} & \text{if } (\text{Thes\_Whole(term)} = 0) \wedge (\text{Text\_Whole(term)} = 1) \\ 0.0 & \text{if } (\text{Thes\_Whole(term)} = 0) \wedge (\text{Text\_Whole(term)} = 0) \end{cases}$$

$$\textbf{Raw\_Score(term)} = 2^{[\min(\text{Term\_Length(term)},3)-1]} \times \text{Thes\_Weight(term)} \times \text{Term\_Freq(term)}$$

Figure 16: Formula for Scoring Documents in 'Partitioning'

$$\underbrace{\left[\frac{1}{\ln(total)}\right]}_{\textbf{Doc-Length Factor}} \times \underbrace{\left[\left(\frac{hits}{total}\right)^2\right]}_{\textbf{Hit-Opportunity Factor}} \times \left[ \sum term\_weight \times term\_freq \times \underbrace{[2^{\cdots}]}_{\textbf{Phrasal-Term Factor}} \times \underbrace{\begin{bmatrix} 1 \\ 1 \\ \frac{1}{4} \\ \frac{1}{8} \\ 0 \end{bmatrix}}_{\textbf{Status Factor}} \right]$$

Figure 17: Schematization of 'Partitioning' Formula

```
ZF09-435-245 7.720000
WSJ870123-0031 6.470000
FR89214-0026 6.310000
WSJ870519-0094 6.130000
WSJ900912-0046 5.830000
WSJ870305-0055 5.360000
ZF07-783-164 5.310000
ZF07-189-244 5.100000
ZF07-443-642 4.980000
AP881122-0107 4.060000
WSJ911018-0122 4.060000
ZF07-971-724 4.050000
ZF07-251-245 3.930000
WSJ870421-0065 3.780000
ZF08-084-048 3.740000
ZF07-621-948 3.670000
WSJ911030-0170 3.610000
ZF09-584-807 3.570000
ZF07-294-735 3.420000
ZF09-526-239 3.390000
ZF07-789-516 3.330000
ZF07-218-520 3.300000
ZF07-800-964 3.300000
ZF07-495-528 3.280000
WSJ900629-0110 3.210000
ZF09-559-173 3.200000
ZF07-118-812 3.170000
WSJ871030-0149 3.160000
ZF07-878-828 3.120000
WSJ870309-0110 3.070000
AP880419-0280 3.050000
...
```

Figure 18: Sample of Data—First-Pass Partitioning Results for Topic 57

TREC document was 'scored' against the super thesaurus in a single pass (Step 6 in Figure 11): effectively, each document was scored against the routing/partitioning thesaurus for each topic in parallel. In particular, every NP in each document was matched against the NPs (terms) in the routing thesaurus; partial matches were allowed. The definitions in Figure 15 and the formula in Figure 16 (given schematically in Figure 17) were used to yield a composite score for the document based on the number of exact and partial hits as a function of document length and term 'value'.

The routing/partitioning thesaurus was used to score the full database, yielding a ranking of all documents relative to all topics simultaneously. As shown in Step 7 in Figure 11, the top 2000 documents for each topic were retained as the partition for the topic for the next stage of processing.

Figure 18 gives sample results of the rankings of documents based on feature scoring for Topic 57. Figure 19 shows the set of 'true' relevants chosen by manual review of the top 10–50 ranked documents.

### 4.7 Final 'Querying'

Figure 20 gives the final steps in the process. There were two essential phases in querying at this point: building the final query vector and querying a partition of the database to retrieve the final set of relevant documents. Note that the query was weighted based on statistics

WSJ861204-0059
WSJ870123-0031
ZF08-695-706
WSJ870305-0055
WSJ870519-0094
WSJ871030-0149
ZF08-096-680
WSJ861208-0024
ZF08-318-964
ZF08-338-122

Figure 19: Sample of Data—Hand-Selected 10 Relevants for Topic 57

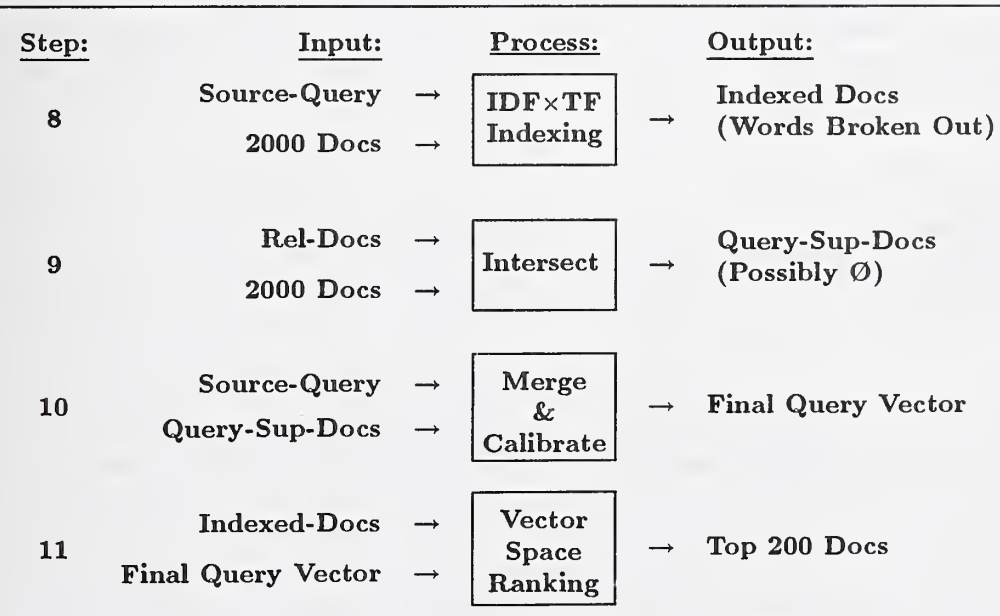



Figure 20: Schematic Representation of Final Steps in Processing

```
\*
\!
58.274554 (mci) () 0
54.311172 (mci communication corporation) () 0
27.003252 (customer) () 0
22.711744 (price cut) () 0
20.039764 (sprint) () 0
19.527712 (price reduction) () 0
19.383471 (at&t) () 0
15.882106 (pre tax charge) () 0
15.090805 (make) () 0
14.682230 (industrywide price cut) () 0
14.592388 (price) () 0
13.906499 (communication) () 0
13.878939 (distance) () 0
13.527033 (industrywide) () 0
...
12.290086 (capital spending) () 0
12.258577 (response) () 0
11.278964 (cash flow) () 0
10.960371 (telecommunication) () 0
10.681927 (conway) () 0
10.480049 (william e conway jr.) () 0
...
1.260289 (product) () 0
1.134040 (york) () 0
1.133174 (computer) () 0
1.121569 (reported) () 0
1.121431 (gain) () 0
\!
\*
\~
```

Figure 21: Sample of Data—Final Query for Topic 57

extracted from a partition of the database. For the ad-hoc queries, the partition used for the statistics was the same as the partition actually being queried. For the routing queries, however, the final query vector was fixed before processing the new text (i.e., the second set of TREC documents). In particular, in this case, the partition used to weight the routing-query vector was extracted from the training corpus (the first set of TREC documents); this vector was then queried against a partition extracted from the new, test corpus.

The NPs and their contained words among the documents in each partition were scored for distribution and frequency; each NP/term- and word-type was given an IDF–TF score. As noted above, for routing queries, the IDF–TF score was based on statistics from the original partition of 2000 documents from the training corpus; it was a static query vector. For the ad-hoc queries, on the other hand, the final partition of 2000 documents was used as the source of statistics for the IDF–TF scoring. Therefore, the scores for terms in the query vector for the ad-hoc queries could vary depending on the set of documents selected in the partitioning process. Figure 21 gives a sample of a final query.

The terms in each topic's routing/partitioning thesaurus were given IDF–TF scores based on the sample; original-query terms were added and the factors of those terms ("1", "2", or "3") were used to multiply their IDF–TF-based scores; the combined terms and their contained words thus formed an extended-query vector (the final query vector).

The 2000 documents for each topic were modeled in vector space (in which all terms and their contained words formed the dimensions) and the final query vector was used to identify and rank the 200 'best' documents, which constituted our results.

### 4.8 Summary of the Process

Figures 22 and 23 summarize the CLARIT–TREC processes described in detail in the preceeding sections. As noted previously, there were only two steps in the CLARIT–TREC process that required non-automatic processing: (1) initial review and weighting of the index terms automatically-nominated and derived for the topic and (2) in the case of ad-hoc queries, review of first-pass retrieved documents to identify 5–10 relevant ones for use in creating a pseudo-thesaurus for further processing.

## 5 Results and Evaluation

This section presents the CLARIT–TREC results in several forms, including broad overviews of the performance, the "official" results tables, and tables of data that focus on statistics that are especially relevant to the CLARIT–TREC approach. Results are presented with only abbreviated explanations.[16]

As noted previously, the CLARIT team submitted both intermediate results ("A") and final results ("B"). The intermediate results were generated by taking the highest-scoring 200 (out of 2000) documents as determined by the routing/partitioning process. Since the strategy of routing/partitioning was to nominate a moderately large candidate subset of documents in which all the true relevants would be found and since the procedure and scoring were designed to over-generate candidates, we expected to have many 'false positives' in each set of 2000. We had no reason to expect the relative ranking of these documents by their evoking routing/partitioning scores would be a good measure of fit to the source topic. By contrast, we expected the final steps (which utilize subset-specific term scoring and vector-space similarity measures) to induce a relative ranking of documents that would represent a good fit to the source topic.

[16] More detailed analysis of the results is given in [Evans et al. in preparation].

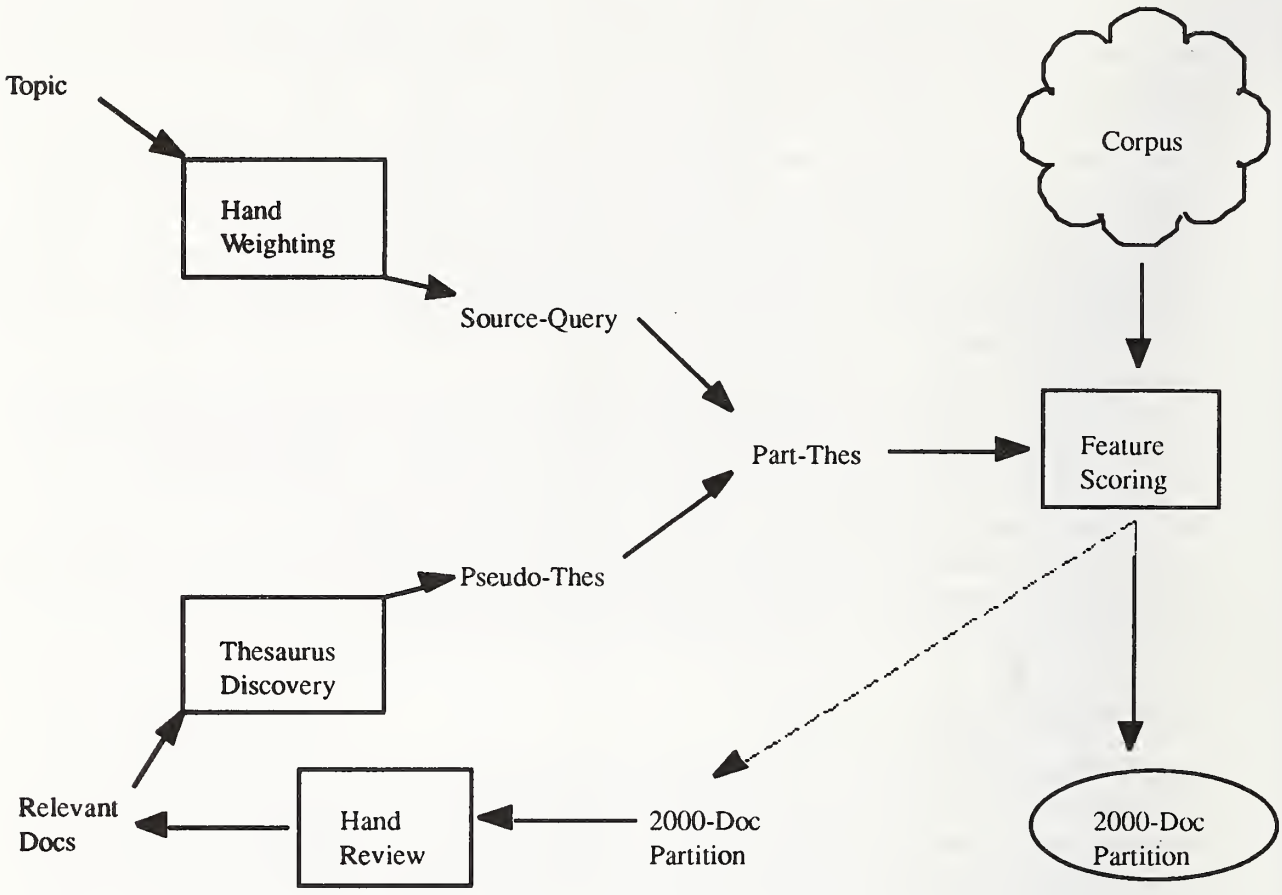



Figure 22: CLARIT "A"—*Evoking*

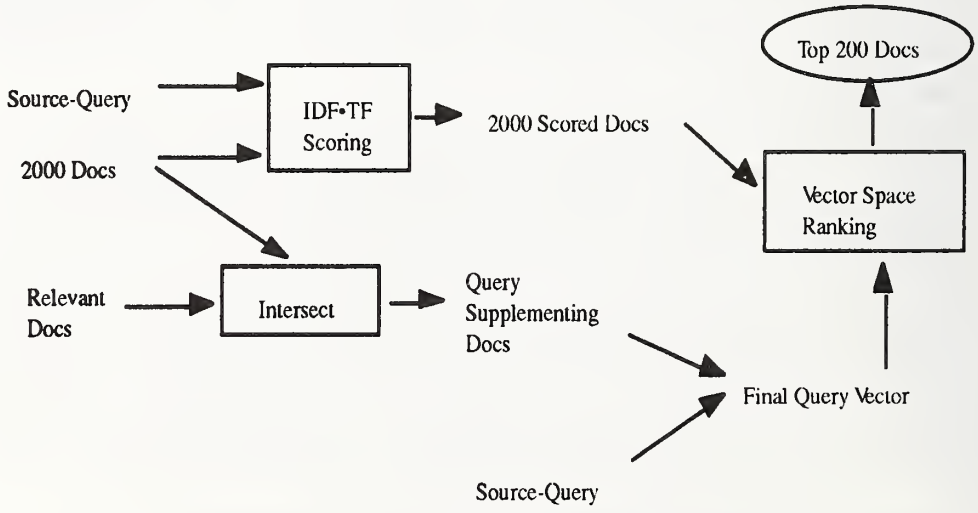



Figure 23: CLARIT "B"—*Discriminating*

| | >Median | =Median | <Median |
|---|---|---|---|
| **11pt Average** | 33 [3] | 3 | 13 [2] |
| **Rels in Top 100** | 31 [4] | 6 | 12 [1] |
| **Rels in Top 200** | 33 [3] | 4 | 12 [1] |

**Note: An Average of 0.53 Total Relevants were in the "A" 2000**

Table 1: Summary of Results for Routing (1–50)

| | >Median | =Median | <Median |
|---|---|---|---|
| **11pt Average** | 34 [7] | 2 | 14 [0] |
| **Rels in Top 100** | 32 [7] | 3 | 15 [0] |
| **Rels in Top 200** | 31 [5] | 2 | 17 [0] |

**Note: An Average of 0.39 Total Relevants were in the "A" 2000**

Table 2: Summary of Results for Ad-Hoc Queries (51–100)

Except where otherwise indicated, all reported results and all analyses in the following sections are based on the CLARIT "B" results.

### 5.1 General Summary of Results

Tables 1 and 2 give the results of applying the techniques described above to the routing topics, 1–50, and to the ad-hoc query topics, 51–100. The numbers in each cell give the number of times the CLARIT–TREC system produced results above, equal to, or below the median for all TREC-participant systems. Numbers in brackets give the instances of 'extreme' performance—best and worst—among all systems.

For the routing topics, the quick partitioning of documents (which produced our "A" set of 2000 candidate documents per topic out of the approximately 300,000 possible documents in the second data set) captured 53% of all the documents judged relevant by the TREC judges. These candidate sets were then processed by the baseline CLARIT–TREC system and ranked results were produced. The results of this ranking were better than the median results of all systems tested at TREC for more than 30 of the topics, according to the measures of average precision ("11pt Average"), the number of relevants in the top 100 returned, and the number of relevants in the top 200 returned. For three or four topics, CLARIT's results were the best of all systems tested in TREC for routing.

For the ad-hoc queries, the "A" sets of 2000 candidate documents per query contained 39% of documents considered as relevant by the human judges, yet results of the discrimination phase of CLARIT provided even better results. Seven times over the 50 queries, CLARIT processing produced the best ranking of all systems tested in terms of average precision and in terms of the number of relevant documents in the first-100 documents returned.

### 5.2 Official Results

Table 3 gives the official results as reported by NIST. The figures for precision "at 30 docs" show, for example, that on average, in the first-30 documents returned by the CLARIT–TREC system, more than half of the routing and 60% of the ad-hoc query documents were relevant.

Tables 4 and 5 present the official calculations of precision by topic, compared to the best, median, and worst performance across all evaluated TREC-participant systems. The tables also give the ranking of CLARIT precision relative to the best precision for each topic ("$B_{200}$/Best").

Tables 6 and 7 show the official results of CLARIT for the first-100 and full-200 documents retrieved for each query, along with 11-pt precision scores. Each line in the table gives a topic number ("T") followed by the total number of documents found relevant by the TREC judges ("Rel"). This is followed by the CLARIT results for the first 100 hundred documents ("$B_{100}$") and the global results (based on all TREC-participant systems) for the greatest ("Best"), the average ("Med"), and the smallest ("Worst") number of documents returned in the first 100 for each topic. This, in turn, is followed by results for the first 200 documents along with the global best, average, and worst performance and the 11-pt average precision figure for CLARIT, along with the best, average, and worst 11-pt performances.

### 5.3 CLARIT "A"/"B" Comparative Results

Tables 8 and 9 present the official results with a focus on CLARIT–TREC differential processing. Here "R#" gives the number of documents found relevant by the TREC judges. "T" is the topic number, followed by "$A_{2000}$", which gives the number of the relevant documents that were present in the partition of 2000 documents created by the evoking routing thesaurus for the topic. Since the actual identifiers of relevant documents were not reported for some topics, there are zeroes (signifying missing data) for some $A_{2000}$ amounts. (For example, we do not know how many relevant documents were in our partitions for Topic 22, 45, 49, etc.). When the $A_{2000}$ number is present, we can measure the effectiveness of our 'discrimination' processing—the final steps in the CLARIT-TREC process. As a measure of effectiveness in bringing the relevant documents to the top of the final ranked list, we give the percentage of relevant documents present in the 2000 document partition that were promoted to the first 100 returned ("% $A_{2000}$"). For the routing queries, these values range from 3% for Topic 18 in which only 3 of the 118 relevant documents available were promoted to the first 100, up to 95% for Topic 21 and 100% for Topic 6 and 23 in which all relevant documents were promoted into the first 100 documents returned. The average was about 42% promoted from among all the 2000 into the top 100. For the ad-hoc queries, the discrimination step was more successful, averaging a 52% promotion rate, and promoting *all* the relevant topics in the partition six times out of 48 (Topic 51, 52, 70, 78, 81, and 92).

These tables also present the results ranked according to performance, taking the average results of all TREC systems as a baseline. The columns marked "$B_{100}$/m" and "$B_{200}$/m" show the ratio of CLARIT results to the baseline, for the first-100 and final-200 documents returned.

## 6 Analysis

We are continuing to evaluate CLARIT–TREC processing results and to interpret CLARIT–TREC performance. This section presents initial observations.

### 6.1 General Observations

It is extremely difficult to evaluate system performance on a task such as the TREC experiments. First, as with many such experiments involving information retrieval, it is difficult to establish

| Routing Queries, 1–50 | |
|---|---|
| Num_queries: | 50 |
| Total number of documents over all queries | |
| Retrieved: | 10000 |
| Relevant: | 14216 |
| Rel_ret: | 3427 |
| | |
| Recall–Precision Averages: | |
| at 0.00 | 0.7479 |
| at 0.10 | 0.5440 |
| at 0.20 | 0.3641 |
| at 0.30 | 0.2513 |
| at 0.40 | 0.1725 |
| at 0.50 | 0.0663 |
| at 0.60 | 0.0336 |
| at 0.70 | 0.0162 |
| at 0.80 | 0.0000 |
| at 0.90 | 0.0000 |
| at 1.00 | 0.0000 |
| | |
| Average precision for all points | |
| 11-pt Avg: | 0.1996 |
| | |
| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
| 3-pt Avg: | 0.1435 |
| | |
| Recall: | |
| at 5 docs: | 0.0130 |
| at 15 docs: | 0.0431 |
| at 30 docs: | 0.0802 |
| at 100 docs: | 0.2080 |
| at 200 docs: | 0.2917 |
| | |
| Precision: | |
| At 5 docs: | 0.5120 |
| At 15 docs: | 0.5320 |
| At 30 docs: | 0.5140 |
| At 100 docs: | 0.4560 |
| At 200 docs: | 0.3427 |

| Ad-Hoc Queries, 51–100 | |
|---|---|
| Num_queries: | 50 |
| Total number of documents over all queries | |
| Retrieved: | 10000 |
| Relevant: | 16400 |
| Rel_ret: | 3409 |
| | |
| Recall–Precision Averages: | |
| at 0.00 | 0.9010 |
| at 0.10 | 0.5497 |
| at 0.20 | 0.3609 |
| at 0.30 | 0.1997 |
| at 0.40 | 0.1304 |
| at 0.50 | 0.0514 |
| at 0.60 | 0.0423 |
| at 0.70 | 0.0169 |
| at 0.80 | 0.0000 |
| at 0.90 | 0.0000 |
| at 1.00 | 0.0000 |
| | |
| Average precision for all points | |
| 11-pt Avg: | 0.2048 |
| | |
| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
| 3-pt Avg: | 0.1374 |
| | |
| Recall: | |
| at 5 docs: | 0.0179 |
| at 15 docs: | 0.0455 |
| at 30 docs: | 0.0804 |
| at 100 docs: | 0.1834 |
| at 200 docs: | 0.2567 |
| | |
| Precision: | |
| At 5 docs: | 0.7560 |
| At 15 docs: | 0.6533 |
| At 30 docs: | 0.6020 |
| At 100 docs: | 0.4570 |
| At 200 docs: | 0.3409 |

Table 3: Offical Results, CLARIT "B"—Topics 1–50 and 51–100

| Results by Topic | | | | | | Results Ranked by Best Precision | | | |
|---|---|---|---|---|---|---|---|---|---|
| Topic | $B_{200}$ | Best | Median | Worst | $B_{200}$/Best | Topic | $B_{200}$ | Best | $B_{200}$/Best |
| 1 | 0.0991 | 0.1897 | 0.0640 | 0.0000 | 0.5224 | 40 | 0.1818 | 0.1818 | 1.0000 |
| 2 | 0.0466 | 0.1470 | 0.0631 | 0.0021 | 0.3170 | 22 | 0.3327 | 0.3327 | 1.0000 |
| 3 | 0.2004 | 0.3374 | 0.1515 | 0.0182 | 0.5940 | 14 | 0.2908 | 0.2908 | 1.0000 |
| 4 | 0.3074 | 0.3088 | 0.1457 | 0.0028 | 0.9955 | 4 | 0.3074 | 0.3088 | 0.9955 |
| 5 | 0.0862 | 0.5286 | 0.1585 | 0.0687 | 0.1631 | 36 | 0.5526 | 0.5694 | 0.9705 |
| 6 | 0.3544 | 0.3879 | 0.1371 | 0.0264 | 0.9136 | 37 | 0.4349 | 0.4506 | 0.9652 |
| 7 | 0.1540 | 0.2886 | 0.1806 | 0.0010 | 0.5336 | 42 | 0.1735 | 0.1806 | 0.9607 |
| 8 | 0.1008 | 0.1225 | 0.0325 | 0.0020 | 0.8229 | 6 | 0.3544 | 0.3879 | 0.9136 |
| 9 | 0.0759 | 0.1765 | 0.1188 | 0.0040 | 0.4300 | 34 | 0.3345 | 0.3776 | 0.8859 |
| 10 | 0.4889 | 0.5764 | 0.3789 | 0.0455 | 0.8482 | 21 | 0.6065 | 0.6889 | 0.8804 |
| 11 | 0.2154 | 0.3005 | 0.0849 | 0.0063 | 0.7168 | 10 | 0.4889 | 0.5764 | 0.8482 |
| 12 | 0.1249 | 0.2494 | 0.1146 | 0.0022 | 0.5008 | 26 | 0.2459 | 0.2928 | 0.8398 |
| 13 | 0.3298 | 0.9058 | 0.2847 | 0.0061 | 0.3641 | 8 | 0.1008 | 0.1225 | 0.8229 |
| 14 | 0.2908 | 0.2908 | 0.1122 | 0.0000 | 1.0000 | 49 | 0.2005 | 0.2438 | 0.8224 |
| 15 | 0.0709 | 0.1591 | 0.1016 | 0.0202 | 0.4456 | 27 | 0.2625 | 0.3326 | 0.7892 |
| 16 | 0.1930 | 0.2763 | 0.0641 | 0.0015 | 0.6985 | 20 | 0.3436 | 0.4361 | 0.7879 |
| 17 | 0.2469 | 0.4791 | 0.2436 | 0.0000 | 0.5153 | 45 | 0.2099 | 0.2722 | 0.7711 |
| 18 | 0.0146 | 0.2158 | 0.1044 | 0.0146 | 0.0677 | 19 | 0.1378 | 0.1801 | 0.7651 |
| 19 | 0.1378 | 0.1801 | 0.1378 | 0.0531 | 0.7651 | 24 | 0.1937 | 0.2552 | 0.7590 |
| 20 | 0.3436 | 0.4361 | 0.2307 | 0.0083 | 0.7879 | 35 | 0.3626 | 0.4800 | 0.7554 |
| 21 | 0.6065 | 0.6889 | 0.5016 | 0.0000 | 0.8804 | 11 | 0.2154 | 0.3005 | 0.7168 |
| 22 | 0.3327 | 0.3327 | 0.2262 | 0.0303 | 1.0000 | 16 | 0.1930 | 0.2763 | 0.6985 |
| 23 | 0.4088 | 0.6626 | 0.2672 | 0.0101 | 0.6170 | 38 | 0.1782 | 0.2619 | 0.6804 |
| 24 | 0.1937 | 0.2552 | 0.0952 | 0.0440 | 0.7590 | 39 | 0.2271 | 0.3597 | 0.6314 |
| 25 | 0.1611 | 0.2726 | 0.1204 | 0.0000 | 0.5910 | 23 | 0.4088 | 0.6626 | 0.6170 |
| 26 | 0.2459 | 0.2928 | 0.1338 | 0.0227 | 0.8398 | 46 | 0.1725 | 0.2824 | 0.6108 |
| 27 | 0.2625 | 0.3326 | 0.2388 | 0.0182 | 0.7892 | 3 | 0.2004 | 0.3374 | 0.5940 |
| 28 | 0.1088 | 0.2095 | 0.1210 | 0.0210 | 0.5193 | 25 | 0.1611 | 0.2726 | 0.5910 |
| 29 | 0.0909 | 0.3677 | 0.0455 | 0.0000 | 0.2472 | 32 | 0.1768 | 0.3157 | 0.5600 |
| 30 | 0.0578 | 0.2613 | 0.1374 | 0.0108 | 0.2212 | 7 | 0.1540 | 0.2886 | 0.5336 |
| 31 | 0.0422 | 0.2235 | 0.0668 | 0.0000 | 0.1888 | 1 | 0.0991 | 0.1897 | 0.5224 |
| 32 | 0.1768 | 0.3157 | 0.0909 | 0.0000 | 0.5600 | 28 | 0.1088 | 0.2095 | 0.5193 |
| 33 | 0.1450 | 0.3147 | 0.1411 | 0.0152 | 0.4608 | 17 | 0.2469 | 0.4791 | 0.5153 |
| 34 | 0.3345 | 0.3776 | 0.2380 | 0.0130 | 0.8859 | 12 | 0.1249 | 0.2494 | 0.5008 |
| 35 | 0.3626 | 0.4800 | 0.3041 | 0.0000 | 0.7554 | 33 | 0.1450 | 0.3147 | 0.4608 |
| 36 | 0.5526 | 0.5694 | 0.2263 | 0.0000 | 0.9705 | 15 | 0.0709 | 0.1591 | 0.4456 |
| 37 | 0.4349 | 0.4506 | 0.3483 | 0.0088 | 0.9652 | 9 | 0.0759 | 0.1765 | 0.4300 |
| 38 | 0.1782 | 0.2619 | 0.1536 | 0.0227 | 0.6804 | 44 | 0.1540 | 0.3761 | 0.4095 |
| 39 | 0.2271 | 0.3597 | 0.2271 | 0.0182 | 0.6314 | 13 | 0.3298 | 0.9058 | 0.3641 |
| 40 | 0.1818 | 0.1818 | 0.1554 | 0.0267 | 1.0000 | 2 | 0.0466 | 0.1470 | 0.3170 |
| 41 | 0.0182 | 0.1360 | 0.0182 | 0.0000 | 0.1338 | 29 | 0.0909 | 0.3677 | 0.2472 |
| 42 | 0.1735 | 0.1806 | 0.1632 | 0.0427 | 0.9607 | 30 | 0.0578 | 0.2613 | 0.2212 |
| 43 | NA | NA | NA | NA | NA | 31 | 0.0422 | 0.2235 | 0.1888 |
| 44 | 0.1540 | 0.3761 | 0.0909 | 0.0029 | 0.4095 | 48 | 0.0303 | 0.1700 | 0.1782 |
| 45 | 0.2099 | 0.2722 | 0.1823 | 0.0000 | 0.7711 | 5 | 0.0862 | 0.5286 | 0.1631 |
| 46 | 0.1725 | 0.2824 | 0.1855 | 0.0199 | 0.6108 | 47 | 0.0373 | 0.2569 | 0.1452 |
| 47 | 0.0373 | 0.2569 | 0.0932 | 0.0031 | 0.1452 | 41 | 0.0182 | 0.1360 | 0.1338 |
| 48 | 0.0303 | 0.1700 | 0.0492 | 0.0015 | 0.1782 | 18 | 0.0146 | 0.2158 | 0.0677 |
| 49 | 0.2005 | 0.2438 | 0.1682 | 0.0130 | 0.8224 | 50 | 0.0000 | 0.2374 | 0.0000 |
| 50 | 0.0000 | 0.2374 | 0.0039 | 0.0000 | 0.0000 | 43 | NA | NA | NA |

Table 4: Official Results, CLARIT "B"—11-Point P–R, Topics 1–50

| Results by Topic | | | | | | Results Ranked by Best Precision | | | |
|---|---|---|---|---|---|---|---|---|---|
| Topic | $B_{200}$ | Best | Median | Worst | $B_{200}$/Best | Topic | $B_{200}$ | Best | $B_{200}$/Best |
| 51 | 0.1722 | 0.7088 | 0.3101 | 0.0000 | 0.2429 | 94 | 0.1950 | 0.1950 | 1.0000 |
| 52 | 0.0909 | 0.3625 | 0.2526 | 0.0015 | 0.2508 | 90 | 0.2709 | 0.2709 | 1.0000 |
| 53 | 0.1975 | 0.2282 | 0.1350 | 0.0216 | 0.8655 | 86 | 0.5920 | 0.5920 | 1.0000 |
| 54 | 0.3345 | 0.4840 | 0.3139 | 0.0455 | 0.6911 | 79 | 0.3545 | 0.3545 | 1.0000 |
| 55 | 0.2577 | 0.2615 | 0.1718 | 0.0312 | 0.9855 | 76 | 0.2158 | 0.2158 | 1.0000 |
| 56 | 0.2627 | 0.2708 | 0.1788 | 0.1332 | 0.9701 | 68 | 0.3482 | 0.3482 | 1.0000 |
| 57 | 0.1987 | 0.3151 | 0.1230 | 0.0202 | 0.6306 | 100 | 0.4166 | 0.4166 | 1.0000 |
| 58 | 0.1770 | 0.5602 | 0.2285 | 0.0211 | 0.3160 | 92 | 0.0909 | 0.0913 | 0.9956 |
| 59 | 0.0992 | 0.2331 | 0.1229 | 0.0040 | 0.4256 | 55 | 0.2577 | 0.2615 | 0.9855 |
| 60 | 0.1584 | 0.2181 | 0.0799 | 0.0041 | 0.7263 | 56 | 0.2627 | 0.2708 | 0.9701 |
| 61 | 0.3252 | 0.4957 | 0.3175 | 0.0909 | 0.6560 | 85 | 0.1625 | 0.1691 | 0.9610 |
| 62 | 0.1671 | 0.2261 | 0.1352 | 0.0138 | 0.7391 | 67 | 0.2148 | 0.2313 | 0.9287 |
| 63 | 0.1299 | 0.2737 | 0.1086 | 0.0029 | 0.4746 | 98 | 0.1408 | 0.1538 | 0.9155 |
| 64 | 0.1279 | 0.2088 | 0.1279 | 0.0055 | 0.6125 | 93 | 0.5258 | 0.5879 | 0.8944 |
| 65 | 0.1184 | 0.2023 | 0.1184 | 0.0070 | 0.5853 | 96 | 0.1500 | 0.1717 | 0.8736 |
| 66 | 0.0714 | 0.3977 | 0.1324 | 0.0011 | 0.1795 | 53 | 0.1975 | 0.2282 | 0.8655 |
| 67 | 0.2148 | 0.2313 | 0.0909 | 0.0126 | 0.9287 | 80 | 0.1225 | 0.1434 | 0.8543 |
| 68 | 0.3482 | 0.3482 | 0.1143 | 0.0035 | 1.0000 | 99 | 0.2768 | 0.3324 | 0.8327 |
| 69 | 0.2408 | 0.6548 | 0.1478 | 0.0051 | 0.3677 | 70 | 0.6451 | 0.7798 | 0.8273 |
| 70 | 0.6451 | 0.7798 | 0.4824 | 0.0023 | 0.8273 | 72 | 0.1673 | 0.2073 | 0.8070 |
| 71 | 0.0455 | 0.2239 | 0.0909 | 0.0000 | 0.2032 | 62 | 0.1671 | 0.2261 | 0.7391 |
| 72 | 0.1673 | 0.2073 | 0.1144 | 0.0012 | 0.8070 | 60 | 0.1584 | 0.2181 | 0.7263 |
| 73 | 0.1343 | 0.1870 | 0.0303 | 0.0000 | 0.7182 | 73 | 0.1343 | 0.1870 | 0.7182 |
| 74 | 0.0606 | 0.1493 | 0.0909 | 0.0076 | 0.4059 | 54 | 0.3345 | 0.4840 | 0.6911 |
| 75 | 0.0909 | 0.1573 | 0.0331 | 0.0027 | 0.5779 | 83 | 0.1586 | 0.2308 | 0.6872 |
| 76 | 0.2158 | 0.2158 | 0.1604 | 0.0107 | 1.0000 | 82 | 0.1804 | 0.2651 | 0.6805 |
| 77 | 0.3503 | 0.5503 | 0.2380 | 0.0051 | 0.6366 | 61 | 0.3252 | 0.4957 | 0.6560 |
| 78 | 0.3319 | 0.6278 | 0.3830 | 0.0006 | 0.5287 | 87 | 0.2262 | 0.3490 | 0.6481 |
| 79 | 0.3545 | 0.3545 | 0.1763 | 0.0354 | 1.0000 | 77 | 0.3503 | 0.5503 | 0.6366 |
| 80 | 0.1225 | 0.1434 | 0.1082 | 0.0136 | 0.8543 | 57 | 0.1987 | 0.3151 | 0.6306 |
| 81 | 0.0909 | 0.2873 | 0.1409 | 0.0036 | 0.3164 | 64 | 0.1279 | 0.2088 | 0.6125 |
| 82 | 0.1804 | 0.2651 | 0.2284 | 0.0883 | 0.6805 | 65 | 0.1184 | 0.2023 | 0.5853 |
| 83 | 0.1586 | 0.2308 | 0.1601 | 0.0262 | 0.6872 | 75 | 0.0909 | 0.1573 | 0.5779 |
| 84 | 0.0147 | 0.1150 | 0.0455 | 0.0000 | 0.1278 | 88 | 0.1491 | 0.2670 | 0.5584 |
| 85 | 0.1625 | 0.1691 | 0.1477 | 0.0147 | 0.9610 | 78 | 0.3319 | 0.6278 | 0.5287 |
| 86 | 0.5920 | 0.5920 | 0.1348 | 0.0038 | 1.0000 | 63 | 0.1299 | 0.2737 | 0.4746 |
| 87 | 0.2262 | 0.3490 | 0.0314 | 0.0005 | 0.6481 | 89 | 0.1997 | 0.4286 | 0.4659 |
| 88 | 0.1491 | 0.2670 | 0.1410 | 0.0061 | 0.5584 | 59 | 0.0992 | 0.2331 | 0.4256 |
| 89 | 0.1997 | 0.4286 | 0.0909 | 0.0043 | 0.4659 | 74 | 0.0606 | 0.1493 | 0.4059 |
| 90 | 0.2709 | 0.2709 | 0.1223 | 0.0182 | 1.0000 | 69 | 0.2408 | 0.6548 | 0.3677 |
| 91 | 0.0490 | 0.2394 | 0.0455 | 0.0000 | 0.2047 | 97 | 0.0455 | 0.1306 | 0.3484 |
| 92 | 0.0909 | 0.0913 | 0.0091 | 0.0000 | 0.9956 | 95 | 0.0909 | 0.2743 | 0.3314 |
| 93 | 0.5258 | 0.5879 | 0.4020 | 0.0028 | 0.8944 | 81 | 0.0909 | 0.2873 | 0.3164 |
| 94 | 0.1950 | 0.1950 | 0.0909 | 0.0101 | 1.0000 | 58 | 0.1770 | 0.5602 | 0.3160 |
| 95 | 0.0909 | 0.2743 | 0.1012 | 0.0083 | 0.3314 | 52 | 0.0909 | 0.3625 | 0.2508 |
| 96 | 0.1500 | 0.1717 | 0.1262 | 0.0045 | 0.8736 | 51 | 0.1722 | 0.7088 | 0.2429 |
| 97 | 0.0455 | 0.1306 | 0.0753 | 0.0035 | 0.3484 | 91 | 0.0490 | 0.2394 | 0.2047 |
| 98 | 0.1408 | 0.1538 | 0.1243 | 0.0341 | 0.9155 | 71 | 0.0455 | 0.2239 | 0.2032 |
| 99 | 0.2768 | 0.3324 | 0.2393 | 0.1157 | 0.8327 | 66 | 0.0714 | 0.3977 | 0.1795 |
| 100 | 0.4166 | 0.4166 | 0.2500 | 0.0455 | 1.0000 | 84 | 0.0147 | 0.1150 | 0.1278 |

Table 5: Official Results, CLARIT "B"—11-Point P–R, Topics 51–100

| T | Rel | $B_{100}$ | Best | Med | Worst | $B_{200}$ | Best | Med | Worst | 11-pt | Best | Med | Worst |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 216 | 29 | 42 | 25 | 0 | 45 | 62 | 30 | 0 | 0.0991 | 0.1897 | 0.0640 | 0.0000 |
| 2 | 384 | 24 | 50 | 24 | 1 | 51 | 72 | 43 | 1 | 0.0466 | 0.1470 | 0.0631 | 0.0021 |
| 3 | 431 | 61 | 91 | 58 | 6 | 95 | 167 | 84 | 8 | 0.2004 | 0.3374 | 0.1515 | 0.0182 |
| 4 | 48 | 28 | 29 | 14 | 2 | 31 | 33 | 18 | 2 | 0.3074 | 0.3088 | 0.1457 | 0.0028 |
| 5 | 150 | 20 | 67 | 27 | 6 | 35 | 116 | 38 | 10 | 0.0862 | 0.5286 | 0.1585 | 0.0687 |
| 6 | 137 | 51 | 57 | 32 | 13 | 78 | 78 | 45 | 15 | 0.3544 | 0.3879 | 0.1371 | 0.0264 |
| 7 | 200 | 44 | 55 | 41 | 0 | 55 | 87 | 63 | 1 | 0.1540 | 0.2886 | 0.1806 | 0.0010 |
| 8 | 159 | 11 | 28 | 11 | 2 | 18 | 43 | 18 | 3 | 0.1008 | 0.1225 | 0.0325 | 0.0020 |
| 9 | 638 | 37 | 81 | 58 | 2 | 78 | 117 | 87 | 8 | 0.0759 | 0.1765 | 0.1188 | 0.0040 |
| 10 | 233 | 87 | 88 | 76 | 10 | 121 | 153 | 110 | 15 | 0.4889 | 0.5764 | 0.3789 | 0.0455 |
| 11 | 196 | 41 | 57 | 29 | 3 | 67 | 89 | 52 | 7 | 0.2154 | 0.3005 | 0.0849 | 0.0063 |
| 12 | 262 | 42 | 61 | 36 | 1 | 54 | 103 | 54 | 4 | 0.1249 | 0.2494 | 0.1146 | 0.0022 |
| 13 | 112 | 46 | 99 | 36 | 2 | 53 | 111 | 46 | 5 | 0.3298 | 0.9058 | 0.2847 | 0.0061 |
| 14 | 203 | 55 | 55 | 28 | 0 | 85 | 85 | 48 | 0 | 0.2908 | 0.2908 | 0.1122 | 0.0000 |
| 15 | 624 | 41 | 75 | 50 | 14 | 62 | 114 | 80 | 17 | 0.0709 | 0.1591 | 0.1016 | 0.0202 |
| 16 | 88 | 25 | 34 | 19 | 0 | 27 | 44 | 24 | 1 | 0.1930 | 0.2763 | 0.0641 | 0.0015 |
| 17 | 303 | 69 | 88 | 69 | 0 | 87 | 154 | 81 | 0 | 0.2469 | 0.4791 | 0.2436 | 0.0000 |
| 18 | 147 | 3 | 36 | 19 | 1 | 16 | 61 | 31 | 2 | 0.0146 | 0.2158 | 0.1044 | 0.0146 |
| 19 | 985 | 67 | 98 | 74 | 48 | 102 | 161 | 102 | 74 | 0.1378 | 0.1801 | 0.1378 | 0.0531 |
| 20 | 403 | 92 | 96 | 74 | 5 | 151 | 178 | 124 | 5 | 0.3436 | 0.4361 | 0.2307 | 0.0083 |
| 21 | 47 | 36 | 41 | 31 | 0 | 37 | 44 | 35 | 0 | 0.6065 | 0.6889 | 0.5016 | 0.0000 |
| 22 | 466 | 96 | 96 | 78 | 10 | 140 | 162 | 120 | 14 | 0.3327 | 0.3327 | 0.2262 | 0.0303 |
| 23 | 100 | 48 | 63 | 37 | 3 | 48 | 74 | 41 | 5 | 0.4088 | 0.6626 | 0.2672 | 0.0101 |
| 24 | 345 | 57 | 69 | 38 | 11 | 86 | 113 | 59 | 11 | 0.1937 | 0.2552 | 0.0952 | 0.0440 |
| 25 | 36 | 20 | 24 | 11 | 0 | 20 | 34 | 14 | 0 | 0.1611 | 0.2726 | 0.1204 | 0.0000 |
| 26 | 313 | 65 | 75 | 42 | 1 | 111 | 122 | 49 | 1 | 0.2459 | 0.2928 | 0.1338 | 0.0227 |
| 27 | 232 | 55 | 63 | 49 | 6 | 109 | 109 | 91 | 10 | 0.2625 | 0.3326 | 0.2388 | 0.0182 |
| 28 | 332 | 18 | 56 | 36 | 13 | 39 | 89 | 47 | 14 | 0.1088 | 0.2095 | 0.1210 | 0.0210 |
| 29 | 142 | 7 | 61 | 10 | 0 | 10 | 79 | 13 | 0 | 0.0909 | 0.3677 | 0.0455 | 0.0000 |
| 30 | 269 | 28 | 64 | 37 | 9 | 44 | 92 | 48 | 17 | 0.0578 | 0.2613 | 0.1374 | 0.0108 |
| 31 | 156 | 19 | 42 | 22 | 0 | 31 | 66 | 31 | 0 | 0.0422 | 0.2235 | 0.0668 | 0.0000 |
| 32 | 119 | 27 | 41 | 13 | 0 | 36 | 52 | 15 | 0 | 0.1768 | 0.3157 | 0.0909 | 0.0000 |
| 33 | 462 | 55 | 83 | 55 | 10 | 83 | 147 | 71 | 17 | 0.1450 | 0.3147 | 0.1411 | 0.0152 |
| 34 | 303 | 66 | 80 | 60 | 6 | 125 | 129 | 104 | 6 | 0.3345 | 0.3776 | 0.2380 | 0.0130 |
| 35 | 270 | 74 | 86 | 70 | 0 | 117 | 139 | 113 | 0 | 0.3626 | 0.4800 | 0.3041 | 0.0000 |
| 36 | 158 | 77 | 82 | 41 | 0 | 103 | 110 | 50 | 0 | 0.5526 | 0.5694 | 0.2263 | 0.0000 |
| 37 | 409 | 100 | 100 | 92 | 6 | 170 | 189 | 158 | 19 | 0.4349 | 0.4506 | 0.3483 | 0.0088 |
| 38 | 810 | 96 | 100 | 70 | 25 | 133 | 169 | 120 | 37 | 0.1782 | 0.2619 | 0.1536 | 0.0227 |
| 39 | 501 | 76 | 100 | 76 | 14 | 123 | 184 | 117 | 24 | 0.2271 | 0.3597 | 0.2271 | 0.0182 |
| 40 | 800 | 99 | 99 | 85 | 16 | 147 | 150 | 121 | 16 | 0.1818 | 0.1818 | 0.1554 | 0.0267 |
| 41 | 144 | 8 | 25 | 8 | 0 | 10 | 34 | 11 | 0 | 0.0182 | 0.1360 | 0.0182 | 0.0000 |
| 42 | 696 | 90 | 96 | 79 | 10 | 125 | 131 | 92 | 10 | 0.1735 | 0.1806 | 0.1632 | 0.0427 |
| 43 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| 44 | 241 | 38 | 63 | 29 | 1 | 43 | 105 | 35 | 1 | 0.1540 | 0.3761 | 0.0909 | 0.0029 |
| 45 | 304 | 63 | 72 | 52 | 0 | 79 | 103 | 71 | 0 | 0.2099 | 0.2722 | 0.1823 | 0.0000 |
| 46 | 51 | 22 | 28 | 21 | 6 | 38 | 40 | 31 | 9 | 0.1725 | 0.2824 | 0.1855 | 0.0199 |
| 47 | 237 | 16 | 59 | 19 | 1 | 32 | 80 | 35 | 2 | 0.0373 | 0.2569 | 0.0932 | 0.0031 |
| 48 | 189 | 11 | 35 | 17 | 1 | 16 | 48 | 28 | 2 | 0.0303 | 0.1700 | 0.0492 | 0.0015 |
| 49 | 139 | 40 | 46 | 34 | 4 | 61 | 65 | 56 | 4 | 0.2005 | 0.2438 | 0.1682 | 0.0130 |
| 50 | 26 | 0 | 12 | 1 | 0 | 0 | 12 | 1 | 0 | 0.0000 | 0.2374 | 0.0039 | 0.0000 |

Table 6: Official Results, CLARIT "B", Topics 1–50

| T | Rel | $B_{100}$ | Best | Med | Worst | $B_{200}$ | Best | Med | Worst | 11-pt | Best | Med | Worst |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 51 | 138 | 24 | 93 | 50 | 0 | 24 | 105 | 77 | 0 | 0.1722 | 0.7088 | 0.3101 | 0.0000 |
| 52 | 535 | 39 | 100 | 89 | 1 | 39 | 191 | 157 | 3 | 0.0909 | 0.3625 | 0.2526 | 0.0015 |
| 53 | 571 | 57 | 84 | 47 | 12 | 115 | 116 | 98 | 42 | 0.1975 | 0.2282 | 0.1350 | 0.0216 |
| 54 | 171 | 53 | 71 | 52 | 7 | 84 | 107 | 81 | 14 | 0.3345 | 0.4840 | 0.3139 | 0.0455 |
| 55 | 810 | 100 | 100 | 87 | 30 | 166 | 178 | 156 | 62 | 0.2577 | 0.2615 | 0.1718 | 0.0312 |
| 56 | 878 | 100 | 100 | 95 | 50 | 178 | 194 | 160 | 102 | 0.2627 | 0.2708 | 0.1788 | 0.1332 |
| 57 | 461 | 63 | 83 | 42 | 12 | 100 | 151 | 67 | 12 | 0.1987 | 0.3151 | 0.1230 | 0.0202 |
| 58 | 159 | 31 | 73 | 42 | 9 | 52 | 117 | 60 | 16 | 0.1770 | 0.5602 | 0.2285 | 0.0211 |
| 59 | 579 | 27 | 81 | 61 | 2 | 58 | 132 | 99 | 4 | 0.0992 | 0.2331 | 0.1229 | 0.0040 |
| 60 | 60 | 14 | 19 | 9 | 2 | 16 | 25 | 15 | 3 | 0.1584 | 0.2181 | 0.0799 | 0.0041 |
| 61 | 206 | 56 | 81 | 57 | 10 | 92 | 113 | 85 | 10 | 0.3252 | 0.4957 | 0.3175 | 0.0909 |
| 62 | 298 | 39 | 57 | 41 | 9 | 69 | 92 | 70 | 29 | 0.1671 | 0.2261 | 0.1352 | 0.0138 |
| 63 | 208 | 24 | 51 | 22 | 3 | 27 | 71 | 27 | 3 | 0.1299 | 0.2737 | 0.1086 | 0.0029 |
| 64 | 375 | 44 | 66 | 44 | 2 | 83 | 108 | 79 | 2 | 0.1279 | 0.2088 | 0.1279 | 0.0055 |
| 65 | 386 | 32 | 65 | 39 | 4 | 47 | 111 | 55 | 7 | 0.1184 | 0.2023 | 0.1184 | 0.0070 |
| 66 | 197 | 17 | 66 | 32 | 0 | 20 | 100 | 40 | 2 | 0.0714 | 0.3977 | 0.1324 | 0.0011 |
| 67 | 534 | 71 | 83 | 47 | 7 | 122 | 127 | 76 | 27 | 0.2148 | 0.2313 | 0.0909 | 0.0126 |
| 68 | 195 | 64 | 64 | 20 | 2 | 95 | 95 | 27 | 7 | 0.3482 | 0.3482 | 0.1143 | 0.0035 |
| 69 | 52 | 22 | 48 | 14 | 2 | 29 | 50 | 17 | 5 | 0.2408 | 0.6548 | 0.1478 | 0.0051 |
| 70 | 55 | 43 | 52 | 36 | 1 | 43 | 54 | 37 | 3 | 0.6451 | 0.7798 | 0.4824 | 0.0023 |
| 71 | 380 | 17 | 66 | 38 | 0 | 29 | 109 | 59 | 0 | 0.0455 | 0.2239 | 0.0909 | 0.0000 |
| 72 | 119 | 27 | 34 | 20 | 0 | 35 | 47 | 33 | 2 | 0.1673 | 0.2073 | 0.1144 | 0.0012 |
| 73 | 183 | 27 | 32 | 13 | 0 | 39 | 56 | 20 | 0 | 0.1343 | 0.1870 | 0.0303 | 0.0000 |
| 74 | 499 | 28 | 63 | 28 | 6 | 37 | 90 | 37 | 8 | 0.0606 | 0.1493 | 0.0909 | 0.0076 |
| 75 | 365 | 20 | 34 | 17 | 3 | 30 | 73 | 28 | 3 | 0.0909 | 0.1573 | 0.0331 | 0.0027 |
| 76 | 294 | 56 | 62 | 42 | 4 | 79 | 84 | 62 | 10 | 0.2158 | 0.2158 | 0.1604 | 0.0107 |
| 77 | 139 | 57 | 68 | 39 | 4 | 62 | 99 | 52 | 6 | 0.3503 | 0.5503 | 0.2380 | 0.0051 |
| 78 | 162 | 54 | 81 | 56 | 0 | 54 | 128 | 86 | 1 | 0.3319 | 0.6278 | 0.3830 | 0.0006 |
| 79 | 232 | 68 | 68 | 39 | 12 | 104 | 104 | 55 | 12 | 0.3545 | 0.3545 | 0.1763 | 0.0354 |
| 80 | 374 | 30 | 50 | 29 | 10 | 69 | 94 | 43 | 21 | 0.1225 | 0.1434 | 0.1082 | 0.0136 |
| 81 | 62 | 4 | 27 | 16 | 1 | 4 | 40 | 25 | 1 | 0.0909 | 0.2873 | 0.1409 | 0.0036 |
| 82 | 602 | 75 | 95 | 79 | 51 | 86 | 177 | 128 | 55 | 0.1804 | 0.2651 | 0.2284 | 0.0883 |
| 83 | 633 | 70 | 84 | 68 | 23 | 106 | 136 | 121 | 28 | 0.1586 | 0.2308 | 0.1601 | 0.0262 |
| 84 | 396 | 14 | 41 | 14 | 0 | 14 | 70 | 21 | 0 | 0.0147 | 0.1150 | 0.0455 | 0.0000 |
| 85 | 896 | 84 | 85 | 68 | 7 | 115 | 155 | 116 | 27 | 0.1625 | 0.1691 | 0.1477 | 0.0147 |
| 86 | 214 | 97 | 97 | 40 | 3 | 130 | 130 | 63 | 6 | 0.5920 | 0.5920 | 0.1348 | 0.0038 |
| 87 | 188 | 51 | 61 | 7 | 0 | 75 | 80 | 16 | 1 | 0.2262 | 0.3490 | 0.0314 | 0.0005 |
| 88 | 166 | 37 | 48 | 28 | 3 | 61 | 75 | 50 | 8 | 0.1491 | 0.2670 | 0.1410 | 0.0061 |
| 89 | 175 | 34 | 69 | 20 | 1 | 46 | 105 | 25 | 4 | 0.1997 | 0.4286 | 0.0909 | 0.0043 |
| 90 | 266 | 64 | 66 | 35 | 12 | 100 | 100 | 53 | 26 | 0.2709 | 0.2709 | 0.1223 | 0.0182 |
| 91 | 40 | 3 | 25 | 5 | 0 | 5 | 30 | 9 | 0 | 0.0490 | 0.2394 | 0.0455 | 0.0000 |
| 92 | 88 | 6 | 18 | 3 | 0 | 6 | 21 | 5 | 0 | 0.0909 | 0.0913 | 0.0091 | 0.0000 |
| 93 | 171 | 77 | 77 | 62 | 2 | 111 | 130 | 90 | 4 | 0.5258 | 0.5879 | 0.4020 | 0.0028 |
| 94 | 310 | 53 | 63 | 24 | 8 | 72 | 89 | 31 | 14 | 0.1950 | 0.1950 | 0.0909 | 0.0101 |
| 95 | 263 | 10 | 64 | 19 | 3 | 15 | 84 | 32 | 5 | 0.0909 | 0.2743 | 0.1012 | 0.0083 |
| 96 | 693 | 79 | 80 | 49 | 1 | 104 | 133 | 82 | 1 | 0.1500 | 0.1717 | 0.1262 | 0.0045 |
| 97 | 352 | 11 | 39 | 18 | 2 | 12 | 61 | 31 | 4 | 0.0455 | 0.1306 | 0.0753 | 0.0035 |
| 98 | 666 | 57 | 67 | 53 | 25 | 95 | 109 | 80 | 27 | 0.1408 | 0.1538 | 0.1243 | 0.0341 |
| 99 | 288 | 68 | 68 | 54 | 30 | 106 | 129 | 98 | 46 | 0.2768 | 0.3324 | 0.2393 | 0.1157 |
| 100 | 316 | 87 | 88 | 66 | 11 | 149 | 149 | 89 | 12 | 0.4166 | 0.4166 | 0.2500 | 0.0455 |

Table 7: Official Results, CLARIT "B", Topics 51–100

| R# | T | $A_{2000}$ | % $A_{2000}$ | $B_{100}$ | b | m | w | $B_{100}/m$ | T | $A_{2000}$ | % $A_{2000}$ | $B_{200}$ | b | m | w | $B_{200}/m$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 119 | 32 | 82 | 0.33 | 27 | 41 | 13 | 0 | 2.08 | 32 | 82 | 0.44 | 36 | 52 | 15 | 0 | 2.40 |
| 48 | 4 | 32 | 0.88 | 28 | 29 | 14 | 2 | 2.00 | 26 | 161 | 0.69 | 111 | 122 | 49 | 1 | 2.27 |
| 203 | 14 | 148 | 0.37 | 55 | 55 | 28 | 0 | 1.96 | 36 | 0 | NA | 103 | 110 | 50 | 0 | 2.06 |
| 158 | 36 | 0 | NA | 77 | 82 | 41 | 0 | 1.88 | 14 | 148 | 0.57 | 85 | 85 | 48 | 0 | 1.77 |
| 36 | 25 | 20 | 1.00 | 20 | 24 | 11 | 0 | 1.82 | 6 | 91 | 0.86 | 78 | 78 | 45 | 15 | 1.73 |
| 137 | 6 | 91 | 0.56 | 51 | 57 | 32 | 13 | 1.59 | 4 | 32 | 0.97 | 31 | 33 | 18 | 2 | 1.72 |
| 313 | 26 | 161 | 0.40 | 65 | 75 | 42 | 1 | 1.55 | 1 | 92 | 0.49 | 45 | 62 | 30 | 0 | 1.50 |
| 345 | 24 | 193 | 0.30 | 57 | 69 | 38 | 11 | 1.50 | 24 | 193 | 0.45 | 86 | 113 | 59 | 11 | 1.46 |
| 196 | 11 | 112 | 0.37 | 41 | 57 | 29 | 3 | 1.41 | 25 | 20 | 1.00 | 20 | 34 | 14 | 0 | 1.43 |
| 810 | 38 | 449 | 0.21 | 96 | 100 | 70 | 25 | 1.37 | 42 | 327 | 0.38 | 125 | 131 | 92 | 10 | 1.36 |
| 88 | 16 | 30 | 0.83 | 25 | 34 | 19 | 0 | 1.32 | 11 | 112 | 0.60 | 67 | 89 | 52 | 7 | 1.29 |
| 241 | 44 | 47 | 0.81 | 38 | 63 | 29 | 1 | 1.31 | 44 | 47 | 0.91 | 43 | 105 | 35 | 1 | 1.23 |
| 100 | 23 | 48 | 1.00 | 48 | 63 | 37 | 3 | 1.30 | 46 | 0 | NA | 38 | 40 | 31 | 9 | 1.23 |
| 112 | 13 | 53 | 0.87 | 46 | 99 | 36 | 2 | 1.28 | 20 | 195 | 0.77 | 151 | 178 | 124 | 5 | 1.22 |
| 403 | 20 | 195 | 0.47 | 92 | 96 | 74 | 5 | 1.24 | 40 | 467 | 0.31 | 147 | 150 | 121 | 16 | 1.21 |
| 466 | 22 | 0 | NA | 96 | 96 | 78 | 10 | 1.23 | 34 | 0 | NA | 125 | 129 | 104 | 6 | 1.20 |
| 304 | 45 | 0 | NA | 63 | 72 | 52 | 0 | 1.21 | 27 | 148 | 0.74 | 109 | 109 | 91 | 10 | 1.20 |
| 139 | 49 | 0 | NA | 40 | 46 | 34 | 4 | 1.18 | 2 | 120 | 0.42 | 51 | 72 | 43 | 1 | 1.19 |
| 262 | 12 | 86 | 0.49 | 42 | 61 | 36 | 1 | 1.17 | 22 | 0 | NA | 140 | 162 | 120 | 14 | 1.17 |
| 800 | 40 | 467 | 0.21 | 99 | 99 | 85 | 16 | 1.16 | 33 | 170 | 0.49 | 83 | 147 | 71 | 17 | 1.17 |
| 216 | 1 | 92 | 0.32 | 29 | 42 | 25 | 0 | 1.16 | 23 | 48 | 1.00 | 48 | 74 | 41 | 5 | 1.17 |
| 47 | 21 | 38 | 0.95 | 36 | 41 | 31 | 0 | 1.16 | 13 | 53 | 1.00 | 53 | 111 | 46 | 5 | 1.15 |
| 696 | 42 | 327 | 0.28 | 90 | 96 | 79 | 10 | 1.14 | 3 | 189 | 0.50 | 95 | 167 | 84 | 8 | 1.13 |
| 233 | 10 | 167 | 0.52 | 87 | 88 | 76 | 10 | 1.14 | 16 | 30 | 0.90 | 27 | 44 | 24 | 1 | 1.12 |
| 232 | 27 | 148 | 0.37 | 55 | 63 | 49 | 6 | 1.12 | 38 | 449 | 0.30 | 133 | 169 | 120 | 37 | 1.11 |
| 303 | 34 | 0 | NA | 66 | 80 | 60 | 6 | 1.10 | 45 | 0 | NA | 79 | 103 | 71 | 0 | 1.11 |
| 409 | 37 | 0 | NA | 100 | 100 | 92 | 6 | 1.09 | 10 | 167 | 0.72 | 121 | 153 | 110 | 15 | 1.10 |
| 200 | 7 | 56 | 0.79 | 44 | 55 | 41 | 0 | 1.07 | 49 | 0 | NA | 61 | 65 | 56 | 4 | 1.09 |
| 270 | 35 | 241 | 0.31 | 74 | 86 | 70 | 0 | 1.06 | 37 | 0 | NA | 170 | 189 | 158 | 19 | 1.08 |
| 431 | 3 | 189 | 0.32 | 61 | 91 | 58 | 6 | 1.05 | 17 | 123 | 0.71 | 87 | 154 | 81 | 0 | 1.07 |
| 51 | 46 | 0 | NA | 22 | 28 | 21 | 6 | 1.05 | 21 | 38 | 0.97 | 37 | 44 | 35 | 0 | 1.06 |
| 501 | 39 | 338 | 0.22 | 76 | 100 | 76 | 14 | 1.00 | 39 | 338 | 0.36 | 123 | 184 | 117 | 24 | 1.05 |
| 462 | 33 | 170 | 0.32 | 55 | 83 | 55 | 10 | 1.00 | 35 | 241 | 0.49 | 117 | 139 | 113 | 0 | 1.04 |
| 384 | 2 | 120 | 0.20 | 24 | 50 | 24 | 1 | 1.00 | 19 | 538 | 0.19 | 102 | 161 | 102 | 74 | 1.00 |
| 303 | 17 | 123 | 0.56 | 69 | 88 | 69 | 0 | 1.00 | 12 | 86 | 0.63 | 54 | 103 | 54 | 4 | 1.00 |
| 159 | 8 | 47 | 0.23 | 11 | 28 | 11 | 2 | 1.00 | 8 | 47 | 0.38 | 18 | 43 | 18 | 3 | 1.00 |
| 144 | 41 | 16 | 0.50 | 8 | 25 | 8 | 0 | 1.00 | 31 | 118 | 0.26 | 31 | 66 | 31 | 0 | 1.00 |
| 985 | 19 | 538 | 0.12 | 67 | 98 | 74 | 48 | 0.91 | 30 | 172 | 0.26 | 44 | 92 | 48 | 17 | 0.92 |
| 156 | 31 | 118 | 0.16 | 19 | 42 | 22 | 0 | 0.86 | 5 | 96 | 0.36 | 35 | 116 | 38 | 10 | 0.92 |
| 237 | 47 | 0 | NA | 16 | 59 | 19 | 1 | 0.84 | 47 | 0 | NA | 32 | 80 | 35 | 2 | 0.91 |
| 624 | 15 | 312 | 0.13 | 41 | 75 | 50 | 14 | 0.82 | 41 | 16 | 0.62 | 10 | 34 | 11 | 0 | 0.91 |
| 269 | 30 | 172 | 0.16 | 28 | 64 | 37 | 9 | 0.76 | 9 | 313 | 0.25 | 78 | 117 | 87 | 8 | 0.90 |
| 150 | 5 | 96 | 0.21 | 20 | 67 | 27 | 6 | 0.74 | 7 | 56 | 0.98 | 55 | 87 | 63 | 1 | 0.87 |
| 142 | 29 | 0 | NA | 7 | 61 | 10 | 0 | 0.70 | 28 | 139 | 0.28 | 39 | 89 | 47 | 14 | 0.83 |
| 189 | 48 | 0 | NA | 11 | 35 | 17 | 1 | 0.65 | 15 | 312 | 0.20 | 62 | 114 | 80 | 17 | 0.78 |
| 638 | 9 | 313 | 0.12 | 37 | 81 | 58 | 2 | 0.64 | 29 | 0 | NA | 10 | 79 | 13 | 0 | 0.77 |
| 332 | 28 | 139 | 0.13 | 18 | 56 | 36 | 13 | 0.50 | 48 | 0 | NA | 16 | 48 | 28 | 2 | 0.57 |
| 147 | 18 | 118 | 0.03 | 3 | 36 | 19 | 1 | 0.16 | 18 | 118 | 0.14 | 16 | 61 | 31 | 2 | 0.52 |
| 26 | 50 | 0 | NA | 0 | 12 | 1 | 0 | 0.00 | 50 | 0 | NA | 0 | 12 | 1 | 0 | 0.00 |
| 0 | 43 | 0 | NA | 0 | 0 | 0 | 0 | NA | 43 | 0 | NA | 0 | 0 | 0 | 0 | NA |

Table 8: General Results, CLARIT "A"/"B", Topics 1–50, Ranked by Performance

| R# | T | $A_{2000}$ | % $A_{2000}$ | $B_{100}$ | b | m | w | $B_{100}/m$ | T | $A_{2000}$ | % $A_{2000}$ | $B_{200}$ | b | m | w | $B_{200}/m$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 188 | 87 | 81 | 0.63 | 51 | 61 | 7 | 0 | 7.29 | 87 | 81 | 0.93 | 75 | 80 | 16 | 1 | 4.69 |
| 195 | 68 | 0 | NA | 64 | 64 | 20 | 2 | 3.20 | 68 | 0 | NA | 95 | 95 | 27 | 7 | 3.52 |
| 214 | 86 | 0 | NA | 97 | 97 | 40 | 3 | 2.42 | 94 | 78 | 0.92 | 72 | 89 | 31 | 14 | 2.32 |
| 310 | 94 | 78 | 0.68 | 53 | 63 | 24 | 8 | 2.21 | 86 | 0 | NA | 130 | 130 | 63 | 6 | 2.06 |
| 183 | 73 | 0 | NA | 27 | 32 | 13 | 0 | 2.08 | 73 | 0 | NA | 39 | 56 | 20 | 0 | 1.95 |
| 88 | 92 | 6 | 1.00 | 6 | 18 | 3 | 0 | 2.00 | 90 | 173 | 0.58 | 100 | 100 | 53 | 26 | 1.89 |
| 266 | 90 | 173 | 0.37 | 64 | 66 | 35 | 12 | 1.83 | 79 | 138 | 0.75 | 104 | 104 | 55 | 12 | 1.89 |
| 232 | 79 | 138 | 0.49 | 68 | 68 | 39 | 12 | 1.74 | 89 | 92 | 0.50 | 46 | 105 | 25 | 4 | 1.84 |
| 175 | 89 | 92 | 0.37 | 34 | 69 | 20 | 1 | 1.70 | 69 | 35 | 0.83 | 29 | 50 | 17 | 5 | 1.71 |
| 693 | 96 | 144 | 0.55 | 79 | 80 | 49 | 1 | 1.61 | 100 | 0 | NA | 149 | 149 | 89 | 12 | 1.67 |
| 52 | 69 | 35 | 0.63 | 22 | 48 | 14 | 2 | 1.57 | 67 | 308 | 0.40 | 122 | 127 | 76 | 27 | 1.61 |
| 60 | 60 | 24 | 0.58 | 14 | 19 | 9 | 2 | 1.56 | 80 | 179 | 0.39 | 69 | 94 | 43 | 21 | 1.60 |
| 534 | 67 | 308 | 0.23 | 71 | 83 | 47 | 7 | 1.51 | 57 | 148 | 0.68 | 100 | 151 | 67 | 12 | 1.49 |
| 461 | 57 | 148 | 0.43 | 63 | 83 | 42 | 12 | 1.50 | 96 | 144 | 0.72 | 104 | 133 | 82 | 1 | 1.27 |
| 139 | 77 | 63 | 0.90 | 57 | 68 | 39 | 4 | 1.46 | 76 | 99 | 0.80 | 79 | 84 | 62 | 10 | 1.27 |
| 119 | 72 | 0 | NA | 27 | 34 | 20 | 0 | 1.35 | 93 | 134 | 0.83 | 111 | 130 | 90 | 4 | 1.23 |
| 294 | 76 | 99 | 0.57 | 56 | 62 | 42 | 4 | 1.33 | 88 | 155 | 0.39 | 61 | 75 | 50 | 8 | 1.22 |
| 316 | 100 | 0 | NA | 87 | 88 | 66 | 11 | 1.32 | 92 | 6 | 1.00 | 6 | 21 | 5 | 0 | 1.20 |
| 166 | 88 | 155 | 0.24 | 37 | 48 | 28 | 3 | 1.32 | 98 | 168 | 0.57 | 95 | 109 | 80 | 27 | 1.19 |
| 288 | 99 | 266 | 0.26 | 68 | 68 | 54 | 30 | 1.26 | 77 | 63 | 0.98 | 62 | 99 | 52 | 6 | 1.19 |
| 896 | 85 | 184 | 0.46 | 84 | 85 | 68 | 7 | 1.24 | 53 | 179 | 0.64 | 115 | 116 | 98 | 42 | 1.17 |
| 171 | 93 | 134 | 0.57 | 77 | 77 | 62 | 2 | 1.24 | 70 | 43 | 1.00 | 43 | 54 | 37 | 3 | 1.16 |
| 571 | 53 | 179 | 0.32 | 57 | 84 | 47 | 12 | 1.21 | 56 | 0 | NA | 178 | 194 | 160 | 102 | 1.11 |
| 55 | 70 | 43 | 1.00 | 43 | 52 | 36 | 1 | 1.19 | 99 | 266 | 0.40 | 106 | 129 | 98 | 46 | 1.08 |
| 365 | 75 | 43 | 0.47 | 20 | 34 | 17 | 3 | 1.18 | 61 | 168 | 0.55 | 92 | 113 | 85 | 10 | 1.08 |
| 810 | 55 | 454 | 0.22 | 100 | 100 | 87 | 30 | 1.15 | 75 | 43 | 0.70 | 30 | 73 | 28 | 3 | 1.07 |
| 208 | 63 | 37 | 0.65 | 24 | 51 | 22 | 3 | 1.09 | 60 | 24 | 0.67 | 16 | 25 | 15 | 3 | 1.07 |
| 666 | 98 | 168 | 0.34 | 57 | 67 | 53 | 25 | 1.08 | 55 | 454 | 0.37 | 166 | 178 | 156 | 62 | 1.06 |
| 878 | 56 | 0 | NA | 100 | 100 | 95 | 50 | 1.05 | 72 | 0 | NA | 35 | 47 | 33 | 2 | 1.06 |
| 633 | 83 | 153 | 0.46 | 70 | 84 | 68 | 23 | 1.03 | 64 | 254 | 0.33 | 83 | 108 | 79 | 2 | 1.05 |
| 374 | 80 | 179 | 0.17 | 30 | 50 | 29 | 10 | 1.03 | 54 | 115 | 0.73 | 84 | 107 | 81 | 14 | 1.04 |
| 171 | 54 | 115 | 0.46 | 53 | 71 | 52 | 7 | 1.02 | 74 | 121 | 0.31 | 37 | 90 | 37 | 8 | 1.00 |
| 499 | 74 | 121 | 0.23 | 28 | 63 | 28 | 6 | 1.00 | 63 | 37 | 0.73 | 27 | 71 | 27 | 3 | 1.00 |
| 396 | 84 | 0 | NA | 14 | 41 | 14 | 0 | 1.00 | 85 | 184 | 0.62 | 115 | 155 | 116 | 27 | 0.99 |
| 375 | 64 | 254 | 0.17 | 44 | 66 | 44 | 2 | 1.00 | 62 | 123 | 0.56 | 69 | 92 | 70 | 29 | 0.99 |
| 206 | 61 | 168 | 0.33 | 56 | 81 | 57 | 10 | 0.98 | 83 | 153 | 0.69 | 106 | 136 | 121 | 28 | 0.88 |
| 162 | 78 | 54 | 1.00 | 54 | 81 | 56 | 0 | 0.96 | 58 | 91 | 0.57 | 52 | 117 | 60 | 16 | 0.87 |
| 602 | 82 | 95 | 0.79 | 75 | 95 | 79 | 51 | 0.95 | 65 | 0 | NA | 47 | 111 | 55 | 7 | 0.85 |
| 298 | 62 | 123 | 0.32 | 39 | 57 | 41 | 9 | 0.95 | 82 | 95 | 0.91 | 86 | 177 | 128 | 55 | 0.67 |
| 386 | 65 | 0 | NA | 32 | 65 | 39 | 4 | 0.82 | 84 | 0 | NA | 14 | 70 | 21 | 0 | 0.67 |
| 159 | 58 | 91 | 0.34 | 31 | 73 | 42 | 9 | 0.74 | 78 | 54 | 1.00 | 54 | 128 | 86 | 1 | 0.63 |
| 352 | 97 | 19 | 0.58 | 11 | 39 | 18 | 2 | 0.61 | 59 | 131 | 0.44 | 58 | 132 | 99 | 4 | 0.59 |
| 40 | 91 | 5 | 0.60 | 3 | 25 | 5 | 0 | 0.60 | 91 | 5 | 1.00 | 5 | 30 | 9 | 0 | 0.56 |
| 263 | 95 | 23 | 0.43 | 10 | 64 | 19 | 3 | 0.53 | 66 | 23 | 0.87 | 20 | 100 | 40 | 2 | 0.50 |
| 197 | 66 | 23 | 0.74 | 17 | 66 | 32 | 0 | 0.53 | 71 | 171 | 0.17 | 29 | 109 | 59 | 0 | 0.49 |
| 138 | 51 | 24 | 1.00 | 24 | 93 | 50 | 0 | 0.48 | 95 | 23 | 0.65 | 15 | 84 | 32 | 5 | 0.47 |
| 380 | 71 | 171 | 0.10 | 17 | 66 | 38 | 0 | 0.45 | 97 | 19 | 0.63 | 12 | 61 | 31 | 4 | 0.39 |
| 579 | 59 | 131 | 0.21 | 27 | 81 | 61 | 2 | 0.44 | 51 | 24 | 1.00 | 24 | 105 | 77 | 0 | 0.31 |
| 535 | 52 | 39 | 1.00 | 39 | 100 | 89 | 1 | 0.44 | 52 | 39 | 1.00 | 39 | 191 | 157 | 3 | 0.25 |
| 62 | 81 | 4 | 1.00 | 4 | 27 | 16 | 1 | 0.25 | 81 | 4 | 1.00 | 4 | 40 | 25 | 1 | 0.16 |

Table 9: General Results, CLARIT "A"/"B", Topics 51–100, Ranked by Performance

a 'gold standard' for the task—authoritative and comprehensive knowledge of the 'correct' responses to a query. A gold standard is difficult to establish in general and is genuinely problematic in the case of the TREC experiments because of the sheer size of the corpus. Second, for the CLARIT-TREC effort in particular, many errors resulted from simple mistakes (i.e., human errors) made in the course of processing. It is difficult to isolate such incidental errors from actual flaws in the design and performance of the CLARIT–TREC system.

In the following sections, we offer thoughts about the 'official' performance of the CLARIT–TREC system, several hypotheses about sources of failure, and a list of known problems in the design and application of the CLARIT–TREC system to the TREC tasks.

### 6.2 Observations About CLARIT–TREC Performance

An evaluation of CLARIT–TREC performance must certainly begin with the comparison of CLARIT–TREC results to the NIST-identified 'correct' results. Such results are reflected in, but are not restricted to, recall–precision statistics, e.g., as given in Table 3 and the comparative results in Tables 4 through 7. We must bear in mind, however, that recall–precision statistics grossly under-simplify the analysis of a system's performance in a retrieval task.

CLARIT recall–precision curves demonstrate very high precision at low percentages of recall. The first few documents returned by the system are extremely likely to be relevant for the given query. Such a result is encouraging, and suggests straightforward methods to improve the recall rates of the system. A simple, automatic iteration of the query, augmented with the top few relevant documents, should extend the 'net' of retrieved relevant documents, as has been well demonstrated in past IR experiments.[17]

Such high precision at low recall tends to validate several hypotheses about performance characteristics of the CLARIT system. A priori, we expect that one of the benefits of accurate and appropriate NLP in information retrieval is an improved ability to discriminate among similar documents. Furthermore, increased precision is an expected result of our 'evoke-and-discriminate' system design. Because only a small subset of candidate relevant documents was considered in the discrimination phase of CLARIT–TREC processing, the distinctions among the documents could be highlighted through more 'expensive' processing of the smaller topical partitions. We were able to use a vector-space model with a large number of dimensions (all multi-word terms and individual words) relative to the number of documents under consideration.

The CLARIT-TREC results are clearly competitive with other state-of-the-art information retrieval systems. As indicated in Tables 1 and 2 and 4 through 7, CLARIT performance relative to other TREC-participant systems is quite good. CLARIT performs consistently above the median and often at or near the top of the group. There are relatively few cases where CLARIT performance is the worst; for the ad-hoc queries, CLARIT does not perform minimally on any of the topics.

### 6.3 Hypotheses About Failure

Comparison of CLARIT "A" recall rates against the full results of CLARIT "B" (Tables 8 and 9) helps to isolate some sources of failure and possible flaws in CLARIT processing. CLARIT "B" processing is confined to the restricted document set identified by the partitioning procedure; it is impossible for the final results to demonstrate recall rates better than the number of documents present in the 2000 document partition. In some cases, many of the actually relevant documents simply were not available in the partition. As noted previously, on average, at

[17] Cf. [Salton & McGill 1983] for discussion, for example.

Feature: Exclusions

| Total | Yes | No |
|---|---|---|
| 71 | 24 | 47 |
| Good | 12 | 25 |
| Bad | 12 | 22 |

Feature: Generalizations

| Total | Yes | No |
|---|---|---|
| 71 | 30 | 41 |
| Good | 15 | 22 |
| Bad | 15 | 19 |

Feature: Temporal Constraints

| Total | Yes | No |
|---|---|---|
| 71 | 11 | 60 |
| Good | 7 | 30 |
| Bad | 4 | 30 |

**Note: None of the above shows significant correlation!**

Table 10: Topic Features × Performance

least half of the relevant documents were missing in both the ad-hoc and routing experiments. Relative to the available relevant documents, the vector-space discrimination phase of CLARIT–TREC processing demonstrated fairly accurate retrieval. Refinements in the techniques used to nominate an initial partition, therefore, become a natural focus for efforts to improve overall system performance.

In some cases, CLARIT processing missed relatively large numbers of correct documents. While it is difficult to imagine a single-strategy IR system that would perform optimally on *all* types of queries, it is important to understand why certain queries may cause failures for a given system. To this end, we conducted several experiments to attempt to identify features in the queries that might have caused sub-optimal performance. We examined all of the topic statements to determine if the presence or absence of certain features is correlated with 'good' or 'bad' recall rates for that topic. ("Good" here is defined as being above the median, "bad" as being below the median recall rate.) In particular, we tested three hypotheses:

1. *The presence of exclusions in the topic causes poor retrieval.* "Exclusions" include any statements that specifically identify concepts or interpretations related to the topic that are *not* to be considered relevant for the purposes of retrieval. For example, one query asked for information on computer crimes, but specifically excluded computer viruses. In CLARIT–TREC processing, no attempt was made to accommodate such exclusions except that specifically negated phrases were deleted from the query term list during initial manual evaluation and term weighting.

2. *The presence of generalizations in the topic causes poor retrieval.* For example, one query asked for information on currently proposed acquisitions involving a U.S. company and a foreign company. Without a list of U.S. and foreign companies, one cannot accurately evaluate candidate documents about acquisitions.

3. *The presence of temporal constraints in the topic causes poor retrieval.* For example, one query asked for the location of presidential candidates during a certain time period. Documents describing events outside of that time period were not considered relevant.

Table 10 gives the instances of 'good' and 'bad' performance relative to the presence or absence of the specific features in topic statements. There is no correlation. The sources of difficulty that the CLARIT–TREC system experienced cannot be traced to such simple characteristics of queries. (Of course, it is still the case that such features of queries present special problems to all IR systems.)

### 6.4 A Collection of Known Problems

As noted previously, we made a number of obvious mistakes in processing the TREC corpus that we simply did not have time to correct. It is likely that some such mistakes contributed to poor performance. The following is a list of known problems that occurred while processing the TREC corpus:

1. *Errors in the Lexicon.* All CLARIT NLP depends on information derived from the lexicon; incorrect lexical entries will cause errors throughout the system. As with morphological processing, these errors result in false analysis for some words.

2. *Morphological Processing Errors.* A certain number of rather simple bugs have been discovered in the morpholgical analysis module. The bugs have caused incorrect analyses of some words.

3. *"Robust" Parsing of Training Corpus.* Unfortunately, the initial parsing of half of the TREC corpus was done in a mode where *all* phrases (not just NPs) in the input were retained in the output. Therefore, this output contained quite a bit of 'noise' in the form of verbs, adverbs, and adjective phrases.

4. *Limited Partition Size.* In creating the partition using the partitioning thesauri, we were limited to sets of 2000 documents. As noted above, one result of the 'low' cutoff is that many relevant documents were simply not included in the partition. Using larger partitions should improve overall performance.

In addition, there are many facets of the CLARIT–TREC process that we believe are not properly calibrated or configured. These include the following:

1. *Iteration of Retrieval.* Automatic feedback of retrieval results can be used to expand the set of relevant documents retrieved. However, we did not have time to perform such feedback during the TREC experiment.

2. *Alternative Scoring Functions.* It is not clear that IDF–TF is the best scoring function to use in biased collections of text, such as our 2000 document partitions. In fact, IDF scoring will specifically demote terms in the document that are known to be important, yet are very well distributed because of their prominence in the partitioning thesaurus.

3. *Refinements to Document Partitioning Formula.* TREC represents our initial attempt at using the partitioning techniques on a large corpus. The feature-scoring formula has not been validated. Additional experiments will likely lead to refinements.

4. *Construction of Query Vectors.* The techniques used to compose query vectors from sample documents are problematic. For example, all terms from the contributing documents were included in the query vector. It is clear that this will contribute 'noise' to the final vector.

5. *Scoring of Query Vectors.* The terms in the query vector were scored in their independent documents, and then the scores were combined using addition. This has the desirable effect of reinforcing terms that occur in several contributing documents, but it is a crude mechanism for doing so.

Improvements can be made in many elements of the CLARIT–TREC system. It is clear that we require further experimentation and analysis to evaluate the system.

## 7 Conclusion

The CLARIT system has been and is continuing to be developed as part of a university research project. The specific configuration of the system used in the TREC experiments was developed in less than one week. As a research prototype, the system has not been engineered for optimal performance.

The TREC task was challenging, in part, because of the size of the corpus. The CLARIT–TREC team had not previously worked with such a large database; we 'invented' solutions to many engineering problems on the fly. We were often inefficient.

Many text processing functions that are available in the CLARIT system or are near completion were not used on TREC documents. In future evaluations, we plan to utilize some of the more sophisticated functionality in the system. For example, we have been developing grammars for recognizing complex tokens such as proper names, dates, times, monetary values, etc., but did not use token recognition modules in CLARIT–TREC processing. We believe that such token recognition would greatly improve the results for queries involving specific persons or time intervals. In addition, we believe that it will be possible to improve results by taking advantage of sub-document scoring. By dividing a long document into smaller, multi-paragraph units we will be able to score documents more accurately with respect to a topic. Finally, we have also been experimenting with generating sub-corpus-derived equivalence classes for words and terms. Equivalence classes will make it possible to expand query terms precisely and selectively.

Clearly, there is a great deal more to be learned from the TREC experiment. In our continuing analyses, we will attempt to parameterize CLARIT performance and to experiment with extensions of CLARIT functionality that may result in superior retrieval.

## 8 Acknowledgements

CLARIT team participation in the TREC activities was made possible, in part, by grants from DARPA/NIST and from the Digital Equipment Corporation. In addition, several groups at Carnegie Mellon University—including the Laboratory for Computational Linguistics, the University Libraries, and the School of Computer Science—provided resources to the CLARIT team. All the groups that supported our effort have our sincerest thanks and appreciation.

## 9 References


[Evans 1990] Evans, David A., "Concept Management in Text via Natural-Language Processing: The CLARIT Approach." *Working Notes of the 1990 AAAI Symposium on "Text-Based Intelligent Systems"*, Stanford University, March, 27–29, 1990, 93–95.

[Evans et al. 1991a] Evans, David A., Ginther-Webster, Kimberly, Hart, Mary, Lefferts, Robert G., Monarch, Ira A., "Automatic Indexing Using Selective NLP and First-Order Thesauri." *RIAO '91*, April 2–5, 1991, Autonoma University of Barcelona, Barcelona, Spain, 624–644.

[Evans et al. 1991b] Evans, David A., Hersh, William R., Monarch, Ira A., Lefferts, Robert G., Handerson, Steven K., "Automatic Indexing of Abstracts via Natural-Language Processing using a Simple Thesaurus." *Medical Decision Making*, *11*, (Supplement), 1991, S108–S115.

[Evans et al. 1991c] Evans, David A., Handerson, Steven K., Lefferts, Robert G., and Monarch, Ira A., *A Summary of the CLARIT Project.* Technical Report No. CMU-LCL-91-2, Laboratory for Computational Linguistics, Carnegie Mellon University, Pittsburgh, PA, 1991, 12pp.

[Evans et al. 1992] Evans, David A., Monarch, Ira A., Lefferts, Robert G., Handerson, Steven K., and Hersh, William R., "Automating Reliable Judgments of 'Similarity' among Medical Texts." Laboratory for Computational Linguistics, Carnegie Mellon University, March 1, 1992.

[Evans et al. in prep.] Evans, David A., Lefferts, Robert G., Grefenstette, Gregory, Handerson, Steven K., Hersh, William R., and Archbold, Armar A., *A Report on the CLARIT TREC-1 Experiments.* Technical Report No. CMU-LCL-93-2, Laboratory for Computational Linguistics, Carnegie Mellon University, Pittsburgh, PA, 1993. (In Preparation)

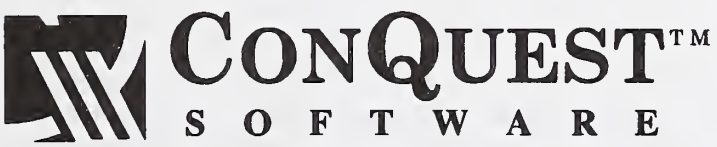
[Hersh et al. 1992] Hersh, William R., Evans, David A., Monarch, Ira A., Lefferts, Robert G., Handerson, Steven K., Gorman, P. N., "Indexing Effectiveness of Linguistic and Non-Linguistic Approaches to Automated Indexing." K.C. Lun, P. Degoulet, T.E. Piemme, and O. Rienhoff (Editors), *Medinfo 92*, Amsterdam, NL: Elsevier Science Publishers B.V. (North-Holland), 1992, 1402–1408.

[Salton & McGill 1983] Salton, G. & McGill, M., *An Introduction to Modern Information Retrieval.* New York, NY: McGraw-Hill, 1983.

# *Site Report for the Text REtrieval Conference*

# *by ConQuest Software Inc.*

Paul E. Nelson
VP of Research & Development




9700 Patuxent Woods Drive, Suite 140, Columbia, Maryland 21046
(410)-290-7150

# Introduction

ConQuest software has a commercially available text search and retrieval system* called "ConQuest" (for Concept Quest). ConQuest is primarily an advanced statistical based search system, with processing enhancements drawn from the field of Natural Language Processing (NLP).

ConQuest participated in Category A of TREC, and so produced results for 50 test queries over the entire 2.3 Gigabyte database. In this category, we constructed queries and submitted results for two methods: Method 1 where queries were automatically generated from the TREC topics, and Method 2 where queries were manually constructed by the software engineers at ConQuest.

We were extremely pleased with our performance in TREC, as the Category A system with the highest 11 point averages. This performance is not indicative of our full potential, however, for the system is still relatively young. We are continuing to evaluate, test, and tune our dictionaries, ranking algorithms, and search methods.

This paper describes our background, the system architecture used for TREC, some features of ConQuest, and a discussion of the results. This paper is written for those with a background in computers with some exposure to artificial intelligence and text retrieval.

# Background

ConQuest has been working on text retrieval since 1988. From the beginning, we meant to use Natural Language Processing (NLP) to better understand the text database and improve retrieval accuracy. This was a natural approach, since both founders of ConQuest, Edwin Addison & Paul Nelson, teach NLP at Johns Hopkins University.

During development, we concentrated on solving the two biggest problems of Natural Language Processing: 1) Most NLP requires large, hand-crafted knowledge bases, and 2) traditional techniques are not robust in the face of text errors.

To solve the first problem, we relied on machine readable dictionaries and thesauri for all knowledge data. Machine readable dictionaries provide ample information on syntax, word variations, and inflected forms. Thesauri and similar sources provide semantic information (word and meaning relationships) which were compiled into structured networks. Both sources were judged to be useful for text retrieval. Combining the resources, however, required significant engineering effort.

The second problem, that of robust processing, required work in two areas. First, NLP development was directed towards statistical approaches and away from rule-based approaches. Statistical approaches typically use heuristics or probabilities to provide confidence factors that accumulate evidence. Unlike rules, which are typically pass/fail, statistical approaches can handle unexpected or variable input without causing total system failure.

But to fully solve the problem of robust processing, we had to have good, solid software engineering. Most NLP systems are ad-hoc affairs, often thrown together at the last minute and patched up. ConQuest has concentrated on producing concrete bullet-proof software,

* For additional information on ConQuest, please contact the author

even if this means delaying some advanced or exotic modules. Our philosophy is that simple programs, well executed, will always out-perform complex tools poorly done.

# System Architecture

ConQuest uses pre-built indexes to perform text database searches at fast speeds. In such a system, all text to be searched must first be indexed. The indexes built in this process can then be used by the text search engine to produce results. Both indexing and search use a dictionary with a semantic network to perform various NLP tasks.

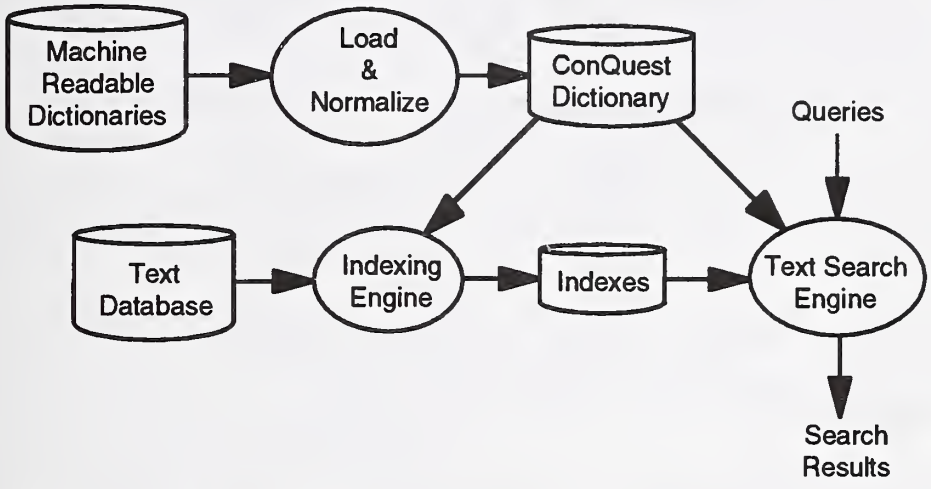



**Figure 1** ConQuest System Architecture

There are other modules in the ConQuest system not shown in Figure 1. These include the library manager, which is responsible for system parameters, database configuration, resource allocation, and physical partitioning of the indexes. Also the dictionary editor, which can be used to edit words, meanings, links, and definitions.

## The Dictionary

ConQuest uses a dictionary augmented with a semantic network to perform indexing and queries. The dictionary is a list of words where each word contains multiple meanings. Each meaning contains syntactic information (part-of-speech, feature values), and a dictionary definition.

The semantic network contains nodes which correspond to meanings of words. These nodes are linked to other related nodes. Relationships between nodes are extracted from machine readable dictionaries. Some example relationship types include synonym, antonym, child-of, parent-of, related-to, part-of, substance-of, contrasting, and similar-to.

ConQuest uses the dictionary for morphological analysis (see below) and idiom processing. The semantic networks are used to expand the query to include related terms. Since connections are made between meanings of words, both in the dictionary and the semantic networks, processing is much more accurate compared to a simple thesaurus.

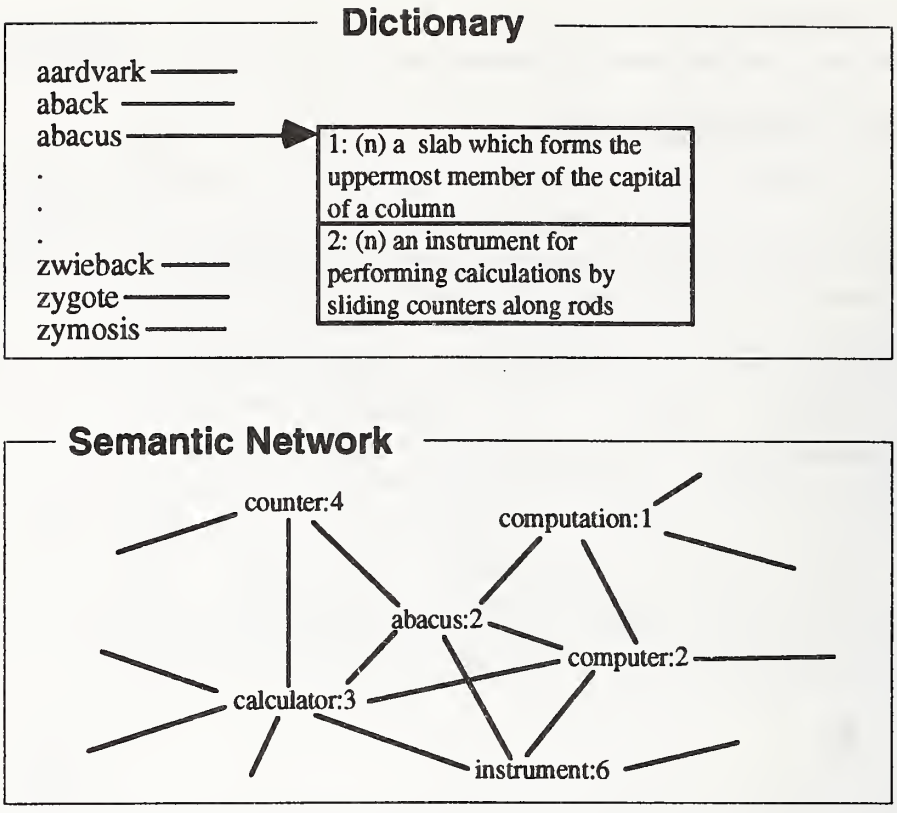



**Figure 2** Example Structures for the Dictionary & Semantic Networks

## Indexing

ConQuest performs several steps of normalization and information extraction during indexing to help queries run faster. Words are normalized and enhanced before indexing with tokenization, morphology, and idiom processing.

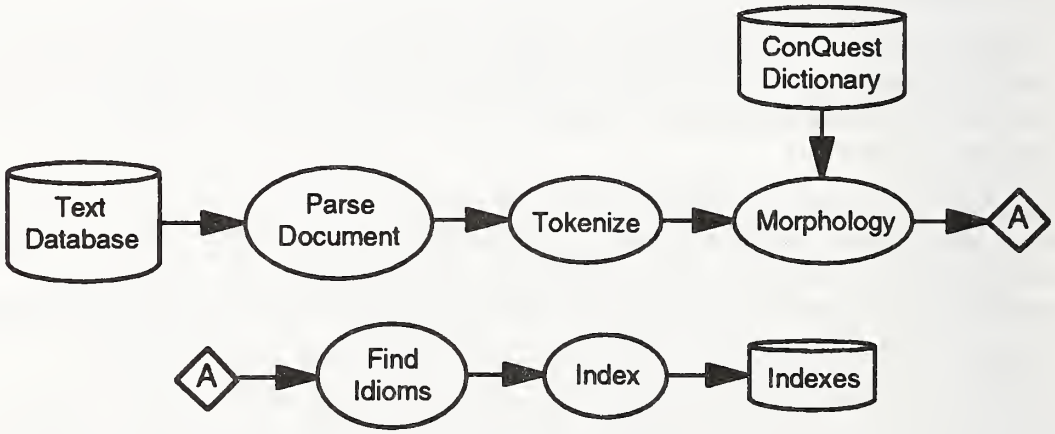



**Figure 3** The Indexing Process

The modules used for indexing are as follows:

- *Parse Document:* Looks for codes in the text database to locate fields such as title, headline, authors, etc. These fields can be indexed, ignored, or stored in a special database for fast access. Parse document takes a command file which describes the structure of the text database to be indexed and can handle a wide variety of different text file formats.
- *Tokenize:* Divides a string of characters into words. This may include special processing for dates, phone numbers, floating point numbers, hyphens, etc.
- *Morphology:* An advanced form of stemming, attempts to remove suffixes and perform spelling changes to reduce words to simpler forms which are found in the dictionary. For example, one morphology rule will take "babies," strip the "ies," add "y", and produce "baby" which is found in the dictionary. Irregular forms of words are stored directly in the dictionary and are not subject to morphological analysis.

  Morphology is a much more accurate form of word reduction than stemming, because the dictionary can be used to validate the transformations. Morphology will not reduce proper nouns, and will produce much more accurate reductions, especially for words ending in "e."
- *Find Idioms:* Idioms are phrases which have a meaning beyond that of the individual words added together. For example, "kangaroo court" has nothing whatsoever to do with kangaroos. Also proper nouns, such as "United States," have a meaning beyond the sum of their component parts.

  This module finds idioms in the text and indexes the idiom as a single unit. This prevents idioms such as "Dow Jones Industrial Average" from getting confused with queries on "industrial history." Words inside of idioms can still be located individually, if desired.
- *Index:* The final step is to store the reduced words and collected idioms into the indexes. The index is an inverted positional word index, which is conceptually similar to the index at the back of a textbook.

The speed of indexing by ConQuest has been measured to be approximately 40 megabytes per hour. This evaluation was done on a Sun IPC Sparc-Station (a 14 MIP computer), with 32 megabytes of RAM. The same speed has been achieved on a 486 IBM-PC, running at a clock rate of 33 Mhz, with 16 megabytes of RAM.

## Query

The query process is more complex than indexing, due to word expansion and ranking. Generally speaking, ConQuest attempts to refine and enhance the user's query. The result is then matched against the indexes to look for documents which contain similar patterns.

Queries are not "understood" in the traditional sense of natural language processing. ConQuest makes no attempt to deeply understand the objects in the query, their interaction, or the user's intent. Rather, ConQuest attempts to understand the meaning of each individual word and the importance of the word. It then uses the set of meanings and their related terms (retrieved from the semantic networks) as a statistical set which is matched against document information stored in the indexes.

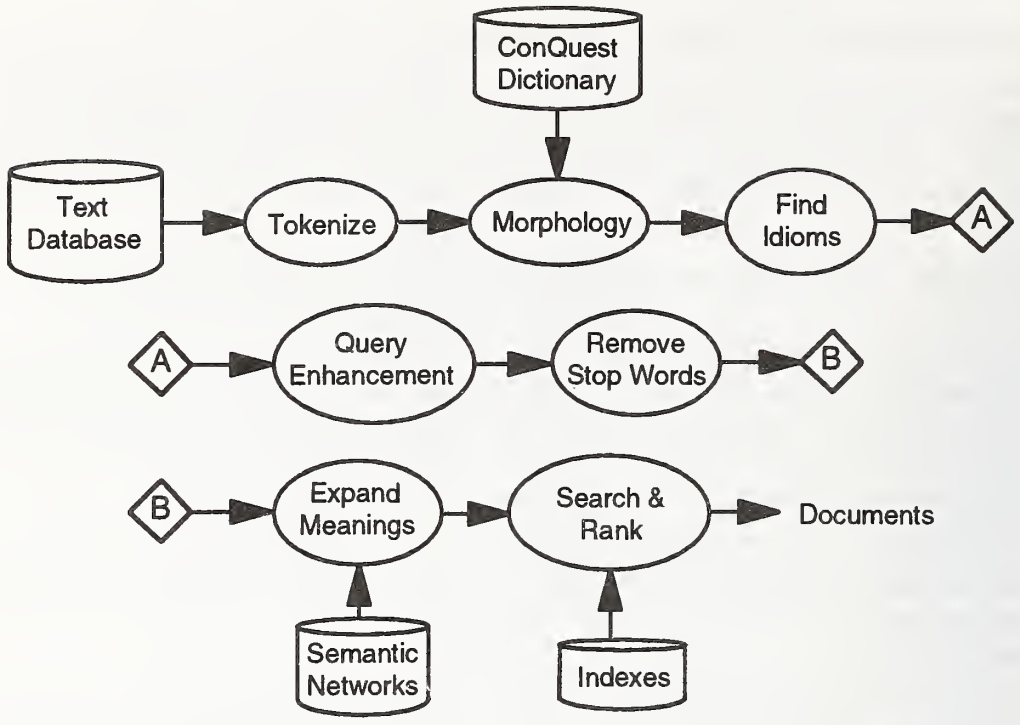



**Figure 4** The Query Process

The following is a description of the modules used for query:

- *Tokenize, Morphology, Find Idioms:* These modules are the same as for indexing.
- *Query Enhancement:* The user is given the opportunity to enhance the query for additional improvement in precision and recall. There are many options available here, but the two most important are to choose meanings and weight query terms.

  Choosing a meaning of a word will restrict the expansion of words to only related terms which are relevant to the chosen meanings. This reduces noise in the query. When running in automatic mode, ConQuest expands all meanings of all words.

  Weighting query terms identifies the importance of the various words in the query. These weights are used by the search engine when ranking documents and computing document relevance factors.
- *Remove Stop Words:* Small function words—such as determiners, conjunctions, auxiliary verbs, and small adverbs—are removed from the query, just as they were during indexing. Removing these terms makes queries faster, and also reduce ambiguous noise in the query.
- *Expand Meanings:* Words in the query are expanded to include related terms.
- *Search and Rank:* ConQuest uses an integrated search and rank algorithm which considers the relevance rankings of documents throughout the search process. Since ranking and search are integrated, the search engine automatically produces the most relevant documents right away. This is different than past approaches, which typically retrieve all matching documents and then rank and sort them as a separate step.

The speed of queries is based on many factors, including database size, amount of expansion, size of dictionaries, etc. Many of these parameters can be modified to achieve the appropriate trade-off between accuracy and speed. For TREC, a query over a 1 Gigabyte database with 15 terms took roughly 9 seconds to retrieve 500 documents on a Sparc-IPC with 32 megabytes of RAM. Equivalent speeds have been measured on 486 IBM-PC computers running at 33 Mhz with 16 megabytes of RAM.

Queries can be expanded to a very large number of terms, if desired. If the user wishes for the greatest amount of recall, a 5 word query can be expanded to 200 or 300 related terms.

## Ranking

Ranking and retrieval with ConQuest uses a variety of statistics and criterion, which are flexible and can be modified to handle varying requirements. The following are some of the factors used in ranking:

> *Completeness:* A good document should contain at least one term or related term for each word in the original query.
>
> *Contextual Evidence:* Words are supported by their related terms. If a document contains a word and its related terms, then the word is given a higher weight because it is surrounded by supporting evidence.
>
> *Semantic Distance:* The semantic network contains information on how closely two terms are related.
>
> *Proximity:* A document is considered to be more relevant if it contains matching terms which occur close together, preferably in the same paragraph or sentence.
>
> *Quantity:* The absolute quantity of hits in the document is also included, but is not as strong a discriminator of relevance as the other factors.

ConQuest is the first truly "concept-based" search system to operate over unrestricted domains. If a document contains the word and some of the related terms, the word is more likely to be used in the correct meaning, using the "contextual evidence" factor above. In this way, ConQuest can determine word meanings at query time.

## Other Features

ConQuest contains a number of other search features, used to handle specific situations and search requirements. These were not used for the Text REtrieval Conference. Some of these features are listed below:

- *Wildcards:* Are useful for locating misspellings in the queries or in the database. The user can specify a word with a wildcard, such as "compute*," and then choose which of the matching terms from the indexes should be included in the query. Query words derived from wildcards are not otherwise expanded.
- *Boolean:* The traditional boolean query mode is also available, and contains all of the basic operators. These include AND, OR, NOT, WITHIN, thesaurus expansion, and nested expressions.

- *Query By Example:* It is possible to submit an entire document as a query, which is called "query by example." Essentially, the document is used as an example, and documents similar to it are retrieved. This function works even when the example document is very large.
- *Search Within:* A query can be directed to search only within a small set of documents. The set of documents is often selected from a previous query. This function is also called "recursive" search or "recurrent" search. This function is especially useful when a statistical query searches over the results of an earlier boolean query, or visa-versa.
- *Numeric and Date Ranges:* ConQuest can find documents which contain numbers or dates within a specified range. Numbers are subject to the standard proximity tests in the boolean and natural language queries, just like other words.
- *Fielded Searches:* A search can be restricted to any particular field in a document. For example, a users often wish to search only over the "authors" field, and not over the full body of the text.
- *Document Categories:* Documents in ConQuest can be categorized as appropriate. Users can target searches to occur only over a single category, or over multiple categories.

# Evaluation Results

ConQuest had the highest overall score in TREC for Category A systems (the full 2.5 Gigabyte database) using the 11 point averages. In comparing ConQuest to other systems, we found the following two graphs to be useful.

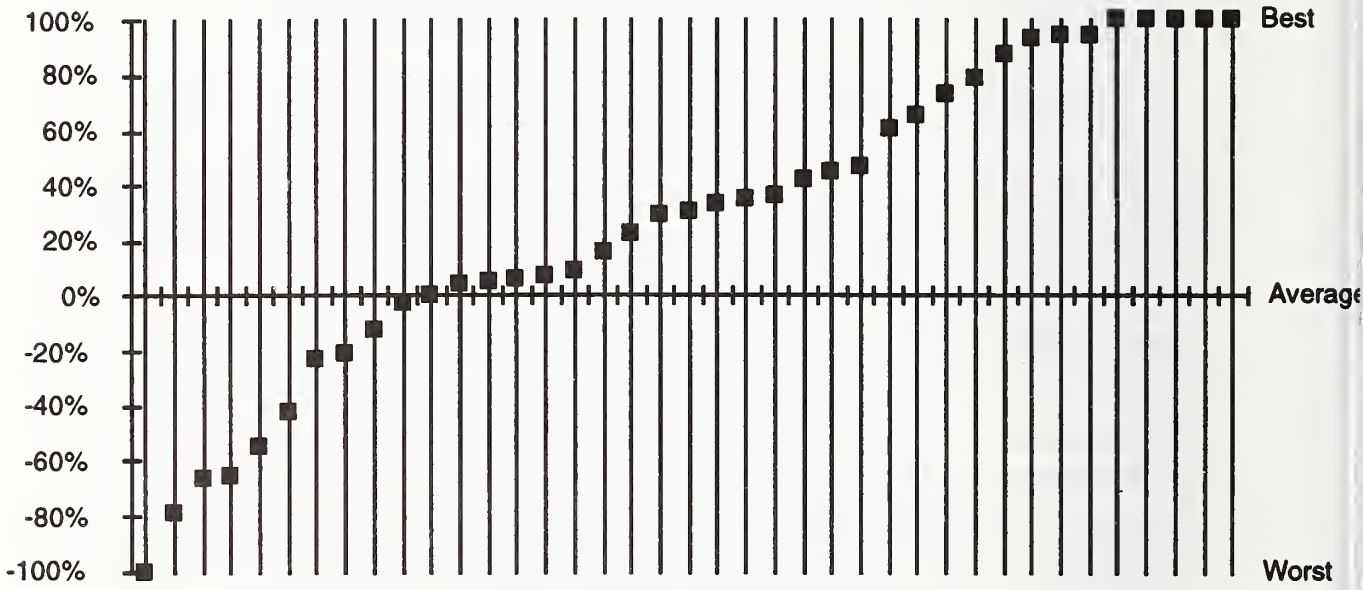



**Figure 5** ConQuest vs All Systems for the 39 Non-Zero Queries Category A, Manual Mode

This first graph (figure 5) shows the results of ConQuest for the 39 non-zero queries (the remaining 11 queries had no relevant documents in the database). A score of 100% represents the best of all Category A systems, a score of -100% represents the worst of all Category A systems. A score of 0% represents average performance.

The results show that ConQuest was best on five of the 39 queries, worst on one query, and well above average for most. Essentially, the area above average is much greater than the area below.

The next chart (figure 6) compares ConQuest to the next two highest Category A manual systems. This chart was originally produced in the TREC proceedings.

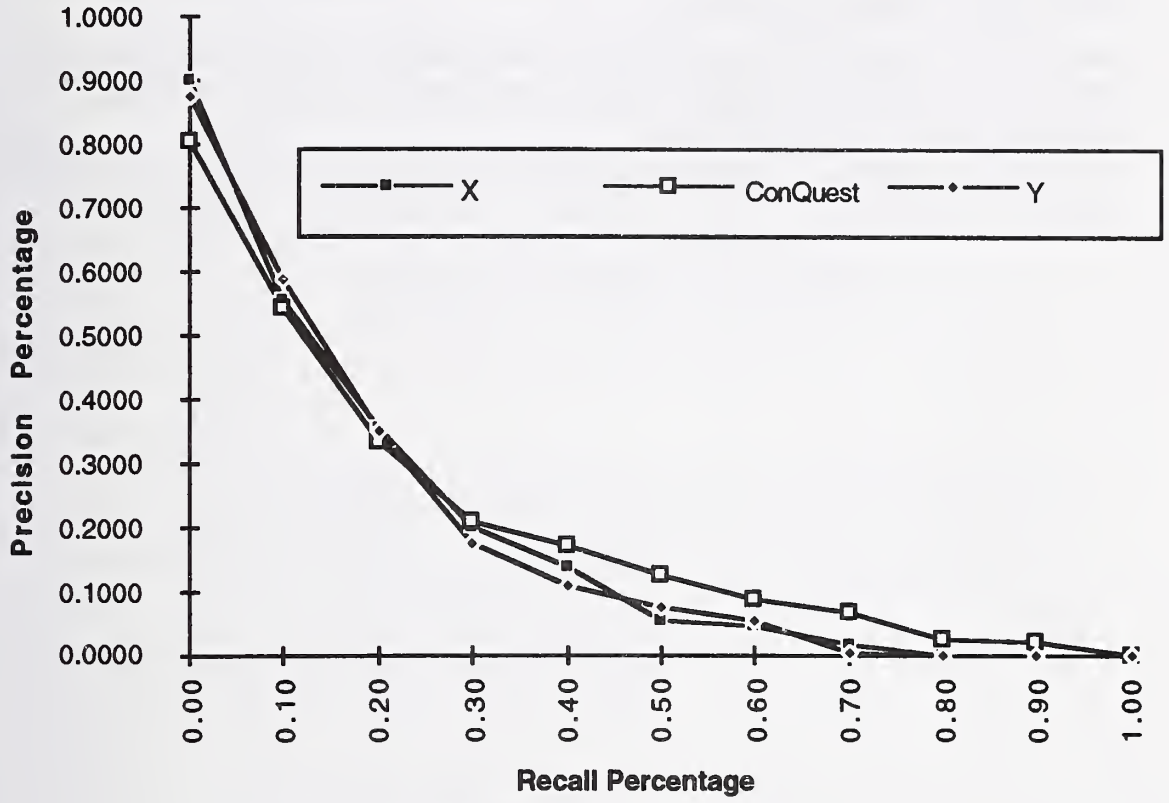



**Figure 6** Precision at Recall for the Top 3 Scorers
Category A, Manual Mode

This chart shows ConQuest having better performance in the high-recall region (where a large percentage of the relevant documents are retrieved). We suspect that the performance in this region is enhanced by our aggressive expansion of terms coupled with the flexible ranking algorithm.

Good performance in the high recall region is important to us, because studies have shown that high recall is the most difficult problem for text retrieval systems. Many systems achieve relatively high precision by discarding documents, but very few achieve high recall.

## Conclusions / Future

The performance of ConQuest exceeded all our expectations. The system is still young, and we fully expected that much more time would be required to rework the ranking algorithms and evaluate the system. These positive results from TREC indicate that we are well on our way to superior retrieval performance.

Since TREC, ConQuest has been working on detailed analysis of all factors which drive ranking and retrieval. This includes the dictionary, morphology, idiom processing, term expansion, and the ranking engine itself.

We feel that our performance on TREC is good start, but that it does not represent our full potential. We look forward to TREC-2 to see how much further we can progress.

# A Boolean Approximation Method for Query Construction and Topic Assignment in TREC*

Paul S. Jacobs, George R. Krupka, and Lisa F. Rau
GE Research and Development Center
Schenectady, NY 12301 (518) 387 - 5059

## Abstract

Word-based search is the simplest and most widely-available method for processing and retrieving volumes of information from free text. However, this common method of searching is generally cumbersome and inaccurate. The method described here automatically generates complex Boolean queries for word-based search, so that the same mechanism can be used with higher accuracy and efficiency. This practical approach worked, with good results, on the entire TREC corpus on both the "routing" and "ad hoc retrieval" tasks, with the official averages in the top group for both tasks.

## 1 Introduction

Full-text search is currently the simplest and most commonly-used method for locating information in large volumes of free text. Because users are accustomed to describing what they are looking for with specific words, and those words are often found in the texts, searching the text for selected words or word combinations is a natural and easy-to-implement method for information retrieval. However, it can be very inaccurate. In some experiments, the percentage of relevant texts retrieved has been shown to be lower than 10%, and a high percentage of material that is retrieved can be irrelevant. Also, it can be very difficult for searchers to compose "queries" that combine the words that are effective in locating relevant material without finding large quatities of irrelevant information as well.

*Special thanks to Michael Charbonneau, Mark Freeman, Geoff Gordon, John Munson and Uri Zernik, who all helped participate in the design and implementation of the GE TREC system.

This research was sponsored in part by the Defense Advanced Research Project Agency. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Project Agency or the US Government.

There are many alternatives to full-text search that can produce much higher accuracy, including statistical methods that weight matches based on relative word frequencies, automatic indexing strategies, and knowledge-based approaches that give very high accuracy for repetitive searches at the cost of a large amount of work in constructing a knowledge base. The ideal search strategy would be one with the accuracy of knowledge-based approaches, but with the simple efficiency of word searches. This is the motivation for the method described here.

The selection of this method for the TREC evaluation combined a sense of the practice of information retrieval with a particular interpretation of what the evaluation is about. For example, the strategy makes several assumptions about the task that are apparently different from those made by other sites, and which perhaps take a looser and less academic view of the experiment. These key assumptions are:

- *Relevance over ranking.* The focus of text interpretation is to assign to each text, with the highest possible accuracy, the set of topics or content indicators that apply to that text (i.e., particularly for routing, to treat the topic or relevance of each text individually rather than as a relative measure against other texts in the corpus).

- *Technology over engineering.* The goal of the experiment is to show high-accuracy, practical results, avoiding the threatening limitations of disk size, memory management, and tractability. (The system did no pre-indexing or analysis of any portion of the corpus.)

- *Text interpretation over query interpretation.* As there may not be any principled structure or methodology in the sample queries, the emphasis on matching texts to an internal representation of each query, rather than on automatic query processing, pushes the limits of the routing and retrieval engine instead of the interface to the user.

These choices were made for various reasons, and are presented here not as the *right* way to view the task but as one software system's implicit focus. A project with a different choice of focus could, for example, produce lower accuracy but have the benefit of fully automatic query processing; another project might have higher accuracy on the top few texts in each category but lower recall on a general routing task. We view our (preliminary) results as very satisfying as a test in a high-throughput, high-accuracy, routing-style interpretation.

The sections that follow cover the motivation for the design and implementation of this method, some specific details of the experiment, and an analysis of the results.

## 2 Background

There are several different methods for general-purpose information retrieval from text, as mentioned, including full-text search, statistical and probabilistic techniques, and those using automatic or manual indexing. Of these, word-based full-text search is generally believed to be the least accurate, but by far the most widely practiced. The method is used in spite of its inaccuracy because it is so easy to implement, and because it gives a direct and natural way for users to describe what they are looking for.

Knowledge-based approaches—those that rely on formal descriptions of concepts rather than combinations of words—can be the most accurate for certain limited tasks, but are the most difficult to implement and apply to general information retrieval problems. While there are a variety of knowledge-based methods, they share the common framework of looking for structured information in a text, while other methods look for simple combinations of words. For example, a simple query looking for stories about takeovers by GE in a typical word-based system might be `(GE AND ACQUISITION)`, while in a knowledge-based system it would look more like `($GE * $acquire * $company)` where the terms marked by $ indicate classes of words or phrases that the system can recognize; for example, GE might include GE and General Electric but not General Electric company of Britain, and `$company` would match any company named or described.

The knowledge-based approach is more accurate, in general, for two reasons. First, it allows the program scanning the texts to look for many ways of expressing the same concept, enhancing *recall*–the percentage of relevant documents that the program retrieves. Second, it focuses search on specific pieces of information, thus enhancing *precision*–the percentage of retrieved texts that are relevant. In this case, improved recall would come from spotting the variety of ways of expressing the concept of acquiring, and improved precision comes from eliminating texts that are about the acquisition of real estate, products, and so forth, as well as acquisitions by other companies of GE's products and services.

Most full-text search systems are driven by Boolean queries–terms joined together by *AND, OR*, and *NOT*, such as *(GE AND ACQUISITION)* above. These can be arbitrarily complex, and one can start to approximate the operation of a knowledge-based approach by combining large numbers of terms. For example, the GE query above could be expressed as:

```
(AND (OR GE (AND GENERAL ELECTRIC (NOT (AND OF BRITAIN))))
     (OR ACQUIRED ACQUISITION BOUGHT (AND (OR TAKE TOOK) OVER) ...)
     (OR COMPANY CORP INC CO LTD SA AG ...))
```

There are several problems with this approach. First, it is very difficult for users to formulate queries of this sort, so, in practice, Boolean queries are

quite simple. Second, it is easier for a system to recognize most or all company names than it is to list all the possible words that could appear in the name of a company. Third, the amount of information captured in the query is still limited; for example, it ignores the order that words appear as well as their adjacency or proximity. Many text search systems allow augmentations to queries that express these constraints, but this makes the queries still more difficult to construct and makes searches less efficient. Hence Boolean retrieval systems remain, in practice, awkward and inaccurate.

The experiment that this team performed in TREC hatched under the pressure of a very difficult task with very limited resources. While a number of individuals participated (including the three authors and five others), the program described here resulted from less than a week of programming over and above the existing software tools we had in hand to apply to the task. The constraints thus forced upon the effort included the realization that not much could be salvaged from the queries as distributed, that the sample relevance judgements were incomplete and the training samples too small for most statistical tests, and that indexing the corpus by any practical method could delay the project by weeks, overflow the disks, and prevent any corrections to the method. This led to a "bare bones" strategy that takes advantage of two strategies: (1) treating boolean queries as an approximation to more detailed, structured representations, and (2) using the *corpus*, rather than the queries, as the main source of information for formulating the topics.

# 3 Our Approach

The fundamental idea behind the method is to take a knowledge-based description of a query or topic, and convert it to a Boolean form that can be efficiently applied by a text-search engine. This Boolean form, furthermore, must be an *approximation* of the knowledge based query, in the formal sense that the Boolean expression should match all texts that the knowledge-based query would, but perhaps can admit more texts.

There are several key advantages to this approach of generating a Boolean query from a knowledge-based description. The simplest benefit is that it makes building queries easier, because much of the work in forming complex Boolean expressions is done automatically. A second major advantage is that the Boolean queries, in approximating a knowledge-based approach, are more likely to give accurate results. Finally, because retrieval using the automatically-generated Boolean query approximates the knowledge-based query, the knowledge-based system can run on the results of Boolean retrieval, thus enhancing precision without having to apply the more computationally-intensive knowledge-based processing to very much text.

One of the obvious problems to overcome in TREC was, with a limited amount of time to formulate 100 queries, with a small amount of training data

(none for the ad hoc test), how to make the queries practical and accurate. Our choice here was to keep the initial queries relatively simple, and to run the results of a "first pass" retrieval against the entire corpus through a statistical filter to pull out terms that would help to augment or refine the query. In addition, the matching engine would display the exact portion of each text that (correctly or incorrectly) matched the query, making it easy to correct glaring errors and refine ambiguous terms. This amounts to a peculiar sort of feedback mechanism that relies on detailed analysis of portions of the corpus instead of user input.

## 3.1 Detailed Method

The method brings together four key elements: (1) a language for expressing knowledge-based topics or queries, developed at GE and described in the open literature, (2) a new program to generate Boolean expressions that approximate these queries (called the *rule compiler*), (3) a program to match the automatically-generated expressions against text to be retrieved (called the *pre-filter*), and (4) a knowledge-based pattern matcher, described in the open literature [2], that takes the results of the first match and rejects texts that do not satisfy the more constrained, knowledge-based query.

Because the pattern matcher is designed as an efficient "trigger" mechanism and an aid in parsing, the knowledge-based queries are mostly simple combinations of lexical categories. The patterns largely adopt the language of regular expressions, including the following terms and operators:

- Lexical features that can be tested in a pattern:
  - token "name" (e.g. "AK-47")
  - lexical category (e.g. "adj")
  - root (e.g. "shoot")
  - conceptual category (e.g. "human")
- Logical combination of lexical feature tests
  - OR, AND , and NOT
- Wild cards
  $ - 0 or 1 tokens
  * - 0 or more tokens
  + - 1 or more tokens
- Variable assignment from pattern components
  ?X =
- Grouping operators:
  <> for grouping
  [] for disjunctive grouping

- Repetition
  * - 0 or more  + - 1 or more
- Range
  *N - 0 to N  +N - 1 to N

In practice, certain of these features are used more than others, and most queries rely most heavily on different lexical categories, grouping, and wildcards. For example, a simple description of the query looking for texts describing sanctions against South Africa is the following:

```
sanction == [(member sanction sanctions disinvestment)
     <Sullivan Principles>
     <punitive *2 measures>] ;

safrica == [(member Buthelezi Pretoria anti-apartheid apartheid)
          <De Klerk> <South (member Africa African)> ] ;

;;; rule 1
$sanction *50 $safrica => (mark-topic 52) ;

;;; rule 2
$safrica *20 $sanction => (mark-topic 52) ;
```

This description says that any matching text must have *both* an indicator of South Africa ($safrica) and one of sanctions ($sanction), and that the sanction phrase must occur within 50 words of the South Africa phrase, except if it only comes afterwards, in which case it must come within 20 words.

A sanction phrase can be any of the simple words *sanction*, *sanctions*, or *disinvestment*, or any phrase including *punitive measures* with no more than two intervening words (like *punitive economic measures*). A South Africa phrase can also be either one of a group of simple words, or a phrase, like *De Klerk*, *South Africa*, or *South African*.

These queries or topic descriptions can be quite complex, and the method has been designed to handle many queries simultaneously, so the rule compiler is designed to produce expressions that can be efficiently applied within a large set of queries. This is important because many queries can share the same simple terms or combinations of terms, and because the pre-filter must match the simplest expressions first.

For the topic description given above, the output of the rule compiler will include the following tests:

```
52        TERM    AFRICAN
 ...
2029      TERM    MEASURES
```

```
 ...
2134      TERM    SANCTION
2135      TERM    SANCTIONS
2136      TERM    DISINVESTMENT
2138      TERM    SULLIVAN
2139      TERM    PRINCIPLES
2141      TERM    PUNITIVE
2144      TERM    BUTHELEZI
2145      TERM    PRETORIA
2146      TERM    ANTI-APARTHEID
2147      TERM    APARTHEID
2149      TERM    DE
2150      TERM    KLERK
2152      TERM    SOUTH
2153      TERM    AFRICA
 ...
2137      OR      2134 2135 2136
2140      AND     2138 2139
2142      AND     2141 2029
2143      OR      2137 2140 2142
2148      OR      2144 2145 2146 2147
2151      AND     2149 2150
2154      OR      2153 52
2155      AND     2152 2154
2156      OR      2148 2151 2155
2157      AND     2143 2156
2158      AND     2156 2143
 ...
TOPIC052  OR       2157 2158
```

Each line in the above data gives a unique number (or topic designator) to the test, a test identifier (either TERM for a simple word test, OR, or AND), and a list of simple terms or previous tests. For example, test 2137 depends on tests 2134, 2135, and 2136, and is true if any of those tests is true, namely, if the text includes any of the words *sanction*, *sanctions*, or *disinvestment*. The tests are automatically ordered so that all tests that are dependent on other tests will have higher numbers than the tests they depend on; thus all TERM tests appear first. In this case, the TERM test AFRICAN appears with a much lower number simply because it is used in many different queries.

The *pre-filter*, which can work either on complete documents or paragraphs, goes through every word in its input and, using a fast table look-up, sets the TERM tests to true for every word it encounters. At the end of input, either the end of the paragraph or end of each document, it runs through the table of possible tests from low numbers to high numbers and sets tests to true if their

conditions are satisfied. A topic test produces a match if it has become true at the end of this process, meaning that the paragraph or document has passed the pre-filter for that query. A single paragraph, of course, can satisfy multiple queries.

The pattern matcher then loads in only those texts that have passed the pre-filter, and uses the original queries to apply more stringent tests to those texts, such as ordering, proximity, and more complex lexical tests. In typical cases, the amount of text the pattern matcher must scan is only 2-3% of the words in the original, and the pattern matcher discards 20-30% of the matches of the pre-filter. Hence, well over 95% of the text filtering is done by the pre-filter, making the whole process efficient as well as showing that the pre-filter closely approximates the knowledge-based pattern matcher.

# 4 Implementation and Results

The lexical analyzer and the matching method described above were programmed in the C programming language, compiled, and tested on a corpus of about 1.2 gigabytes of text (about 200 million words) with a set of 50 topics, as part of the government-sponsored Text Retrieval Conference (TREC). The program was then tested on an additional gigabyte of text (the "second" disk) with an additional 50 topics, and the results were submitted to the government for comparative evaluation purposes. The results, in this case, were a ranked list of up to 200 documents that matched each query.

The final test, run on two SUN workstations, took about 10 hours of elapsed time for the entire corpus. The pre-filter ran at a speed of several hundred thousand words per minute, thus covering the entire contents of each disk in a few hours. This would reduce the text to 3% of the original data, with each paragraph marked according to the queries that matched in the pre-filter. During query refinement, a statistical filter ran through the entire contents of this marked portion of the corpus, pulling out words and phrases with a high correlation (a variation of the metric for categorization reported in [1]) to a particular topic. For every topic, then, the statistical filter listed terms not in the (Boolean) query that occurred with a disproportionate frequency in text that matched the query. This, for example, produced the terms *Buthelezi* and *punitive measures* in the South Africa query above.

Figure 1 summarizes some of the official results from TREC, including the number of times the systems achieved the best score, and were above, below and equal to the median score. The system was very close to the top in both the ad hoc and routing tasks, and was well above median on most topics. The pre-filter, surprisingly, was better than the pattern matcher almost across the board, indicating that with this style of evaluation there seemed to be no real benefit to deeper analysis than a simple Boolean match. The logical conclusion is that a Boolean routing or retrieval engine can perform as well as much more

complex and sophisticated methods, so long as enough knowledge is used in constructing the Boolean expressions.

| | AD HOC TEST | | | ROUTING TEST | | |
|---|---|---|---|---|---|---|
| | 11−pt. avg. | Rel.@ 100 docs. | Rel./ Retrieved | 11−pt. avg. | Rel.@ 100 docs. | Rel./ Retrieved |
| Boolean pre−filter | .2029 | 47.2 | .46 | .2078 | 35.6 | .34 |
| Pattern matcher | .1961 | 46.2 | .46 | .1851 | 34.6 | .37 |
| Median for all runs | .1585 | 39.7 | | .1246 | 28.6 | |

| | top score | above median | below median | equal | top score | above median | below median | equal |
|---|---|---|---|---|---|---|---|---|
| Boolean pre−filter | 5 | 28 | 15 | 2 | 5 | 31 | 6 | 7 |
| Pattern matcher | 5 | 26 | 17 | 2 | 6 | 27 | 10 | 6 |

Figure 1: GE TREC test results

## 5 Limitations and Future Work

While the overall, relative results were generally strong, the system as it was implemented had some basic flaws that should be easily corrected. One characteristic of this method is that it seems to produce excellent results on the queries that produce large volumes of data, and tends to produce almost nothing on some of the narrowest queries. It also produces both high precision and high recall for the texts that match, but makes it difficult to "loosen up" to let in more texts. Since the match relies on at least some exact match between query terms and texts, there are some queries (for example, about the details of rewriteable disks), that produced no hits. By contrast, in one configuration, the system assigned over 4,700 texts to one query ("sightings of 1988 presidential candidates"), of which only 200 were included in the submitted results. This might be desirable behavior for a routing system, but in the TREC style of evaluation the system did badly on those queries where it failed to produce at least 200 documents. On the other hand, the system produced at least 6000 responses in even its strictest configuration, or at least 120 texts, on average, per query.

The problems with undergeneration (and the related problem of not doing a very good job of ranking the documents) were due to the fact that our system was designed for routing, while TREC used traditional retrieval evaluation methods, along with a 200-document cutoff, effectively counting recall on the harder topics much more heavily than overall recall. Our approach can correct for this by using a more flexible statistical method to expand the queries and by performing a more sophisticated ranking (the document ranking as reported was implemented post hoc in one line of Unix code).

More important than the problems to correct, there is an important result here to build on. Our experience has been that pattern matching can be a close approximation for this sort of task to natural language processing, so it might seem that advanced methods are much more critical for finding *what to put in the queries* than they are for the detailed analysis of the texts. The general framework of this approach means that, with the continued development of advanced methods for natural-language based corpus analysis, substantial performance improvements can come within the context of almost any current text retrieval systems.

# 6 Evaluation Methods

One unusual characteristic of our method is that it assumes that each relevance judgement that the system makes is made independently of all other texts, as in a routing task where the system processes each incoming message in turn and assigns topics or actions for filing or routing that message. Certainly, this style has certain advantages—it is simple, clear, and makes parallel processing easy—and it reflects some real assumptions about the nature of the task. However, although it seems to have done very well relative to other systems, it is not consistent with the instructions for submitting results in TREC, and certainly doesn't lead to the best possible showing on some of the results.

Topic 77, about poaching techniques, is one example of the different (naive, perhaps) perspective toward evaluation that our system adopts. The query specifies:

> *A relevant document will identify the type of wildlife being poached, the poaching technique or method of killing the wildlife which is used by the poacher, and the reason for the poaching (e.g. for a trophy, meat, or money).*

This is a very specific query. Our test (bootstrapping) sample produced a good number of hits, but most of them failed to include one of the required pieces of information, usually the technique or method of killing. So, we narrowed the query. The result is that, for this query, the system produced 9 total documents, 6 of which were judged relevant. This is high precision (.67), but it doesn't help the overall results, since for this topic the precision at 200 documents is treated

as .03. To us, narrowing the query seemed a good idea because the precision on this topic otherwise would have been low, but we did not realize that the documents that the system *didn't retrieve* were still treated as incorrect in this calculation.

On Topic 43 (1991 AI conferences), our system produced 3 documents, all of which were irrelevant. This "routing" topic was later discarded because no relevant documents were found in the corpus, but there is nothing inherently wrong with testing topics for which there is no data. In fact, the ideal routing system should produce 0 hits for such a topic, not 200 hits as dictated in TREC.

Certainly ranking and routing don't go together in any real task on a gigabyte sample. One way that future evaluations can test routing is to use a random (or otherwise fair) sample of the collections as a test, judge every document in that sample with respect to every query, and then measure each system's recall and precision on the basis of the sample. This would probably require less hand-work in judging relevance, but would require that each system produce topic assignments for every document in the collection (from which the assignments for the test sample would be extracted post hoc). This could be impossible for some systems. On the other hand, the strategy would give real numbers for both recall and precision, and would be much truer to the routing task.

# 7 Utility

The main purpose of this method is as a front-end for computation-intensive natural language processing of large bodies of text. Because the pre-filter closely approximates more in-depth processing with a very fast, efficient process, it permits detailed processing of large volumes of text by discarding most of the irrelevant material and by producing a rough approximation of the more detailed processing.

The method is more broadly applicable to problems in information dissemination and retrieval. Accuracy is only one appealing characteristic of the technique, since the main innovation is that it allows for improved accuracy within the context of traditional word-based full-text search.

In addition to the programs described here, the method was tested with a statistical corpus analyzer that helps to identify candidate words and phrases to include in queries. This method helps to overcome some of the limitations of word-based methods in cases where statistical approaches clearly seem to do better. As an additional experiment, this automated corpus analysis can be used to reduce further the amount of labor involved in building queries.

## 8 Summary

GE's participation in TREC involved a small implementation of a simple strategy for compiling knowledge based pattern matcher rules into the language of Boolean expressions. A statistical corpus analyzer helped to formulate and refine queries for both the ad hoc and routing tasks, and the resulting matching engine ran on the entire 2.3 gigabytes of text. The simple Boolean retrieval engine performed very well on both tasks. These results are promising, both from the perspective of accuracy and for the simplicity with which they seem to bring knowledge-based techniques to bear within the rudimentary framework of word-based retrieval.

## References

[1] Paul S. Jacobs. Joining statistics with NLP for text categorization. In *Proceedings of the 3rd Conference on Applied Natural Language Processing*, April 1992.

[2] Paul S. Jacobs, George R. Krupka, and Lisa F. Rau. Lexico-semantic pattern matching as a companion to parsing in text understanding. In *Fourth DARPA Speech and Natural Language Workshop*, pages 337–342, San Mateo, CA, February 1991. Morgan-Kaufmann.

# Text Retrieval with the TRW Fast Data Finder

Matt Mettler
TRW Systems Development Division
One Space Park R2/2194
Redondo Beach, CA 90278
(310) 814-4925
matt@wilbur.coyote.trw.com

## 1.0 Introduction

TRW has been building high performance text processing and retrieval systems for a number of years. Most of these systems have involved the application of the TRW Fast Data Finder (FDF) text search hardware and have been designed to meet the requirements of specific government customers. Our goal for the TREC conference has been to consider and experiment with the FDF as a tool for more general purpose information retrieval, and to determine the FDF's strengths and weakness compared to conventional information retrieval techniques.

Our experience with the TREC conference has left us encouraged about the ability of a text scanning approach to be competitive with the more involved information retrieval techniques. The inherent limitations of the FDF hardware do not prevent competitive precision and recall for general information retrieval applications when the user topics are properly understood and the topic queries are properly tuned to the dataset.

## 2.0 FDF Text Retrieval Approach

The Fast Data Finder is a hardware device that performs high-speed pattern matching on a stream of 8-bit data. It consists of an array of identical programmable text processing cells connected in series to form a pipeline processor. The cells are implemented using a custom VLSI chip designed and patented by TRW. In the latest implementation, each chip contains 24 processor cells and a typical system will have 3,600 of cells. Each cell can match a single character of query or perform all or part of a logical operation. The processors are interconnected with an 8-bit data path and approximately 20-bit control path. To perform a search, a microcode program is first downloaded into the pipeline to direct each processor. The database is then streamed through the pipeline. The data bytes clock through each processor in turn until the whole database has passed through all processors. As the data is clocking through, the processors alter the state of the control lines depending on their program and the data stream values.

When the pipeline's processor cells detect that a series of database characters have passed by that match the desired pattern, a hit is indicated and passed by external circuitry back to the memory of the host processor and to the user. The FDF pipeline runs at a constant speed as it performs character comparisons and logical operations, regardless of query complexity. The system we used for the TREC conference searched at 10 MB/sec.

The queries or patterns are specified in the FDF's Pattern Specification Language (PSL). The hardware directly supports all the features in the PSL query language without the need for software post-processing. The processors in the pipeline may all be used to evaluate a single large query or may assigned to evaluate numerous smaller queries. The number of pipeline cells a query needs is proportional to the size of the query. PSL provides numerous search functions, which may be nested in any combination, including:

- Boolean logic including negative conditions
- Proximity on any arbitrary pattern
- Wildcards and "don't cares"
- Character alternation
- Term counting, thresholds, and sets
- Error tolerance (fuzzy matching)
- Term weighting
- Numeric ranges

### 2.1 Advantages and Disadvantages of Hardware Scanning

There are four principle advantages in using a hardware scanning approach for information retrieval. First, the FDF can perform pattern matching functions much faster and more cost-effectively that a general purpose CPU. This benefit comes in part because of the parallelism of the FDF architecture. Second, a hardware scanner like the FDF can begin processing the data immediately upon its receipt. There is no need to wait for the data to be preprocessed or indexed before it can be searched. This is especially important for dissemination (routing) applications. Third, no extra disk space is needed to store inverted index data or other vector data beyond the text itself. Finally, the system's response time in evaluating a query is independent of the query's complexity and thus easily predictable. The FDF can perform fuzzy pattern matching on a term like "krasnoyarsk" with three missing, incorrect, or extra characters, as easily as performing an exact string match.

There are two disadvantages to the FDF scanning approach. First, it is moderately expensive to buy the hardware and adapt application programs to work with it. The approach is therefore not cost effective for low-end applications or systems that don't do significant amounts of text processing. Second, since the search is not complete until the entire database has been scanned, the time to complete a single simple query will be greater than with indexing methods.

## 3.0 Query Generation Approach

Our objective for the TREC conference was to see if we could utilize the pattern matching power of the FDF to achieve superior recall and precision. This in turn revolved around how well we could construct the queries for the FDF to execute. We were able to identify

at least three possible approaches that could be used in preparing queries for execution by the Fast Data Finder.

- Parse the topic narratives, extract key terms and phrases, expand the terms where possible, and generate queries to find documents with the same combinations of terms.

- Take a sample of relevant documents, extract common keywords and phrases, especially those the occur multiple times, and generate queries to find documents with at least some of the same phrases and keywords within a sliding window of text about the size of a paragraph.

- Construct the initial queries manually and refine them iteratively.

We elected to try both methods (ii) and (iii). To supply the relevant documents for the statistical trials, we used the sample relevance judgments supplied by NIST in late May and early June.

### 3.1 Automatic Query Generation

Our plan was to take sample documents for a particular topic, merge them together, and build a PSL query that would find similar documents. Using the single document WSJ870320-0062 as a seed, the query would be something like:

```
{30 words ->
   5+ ('cola'; 'coca'; 'coca cola'; 'bottling';
       'enterprises'; 'cola bottling'; 'cola enterprises';
       'coca cola enterprises'; 'coca cola bottling'; 'atlanta')}
```

This query finds a document which contains a 30 word sliding window with 5 or more of the specified terms or phrases. The term list is determined by removing stopwords and counting the number of occurrences for each term, 2 word phrase, and 3 word phrase in the seed document. The top 10 terms/phrases with the highest counts are selected. The "30 words" and "5 or more" values were selected arbitrarily and we'd planned to run a series of trials to determine the optimal values.

The initial experiments with this method of query construction were not encouraging. We ran into three difficulties.

- The May/June NIST sample relevance judgments seemed incomplete and inaccurate and were not giving us the statistical base we'd hoped for.

- This method assumes that the whole document in all on one subject. Longer seed documents were contributing terms that had little to do with the topic. Some method to segment the documents and indicate the interesting section is required.

- This method wasn't capturing the subtlety of the topics. The query shown above does an excellent job of finding documents about Coca Cola Enterprises or bottling units in Atlanta but completely misses the part about antitrust violations because it is only mentioned once in the article.

### 3.2 Manual Query Generation

The poor results from our initial trials with statistical query generation led us to fall back on a purely manual (with feedback) approach. We extracted key concepts from the topic description, added additional terms from outside knowledge or by observing them in database documents. In building our multiple queries to provide a coarse grain ranking, we favored documents where the subqueries matched in lead sentences or paragraphs. We mostly ignored the May/June NIST sample judgements.

Refinement of the queries was done manually by executing them, reviewing the results, and modifying the queries. The easier topics required only a few iterations, while on some of the more difficult topics we iterated several dozen times. We stopped working on a topic when it seemed that the results were converging to practical limit for our approach, i.e. when adding additional synonym keywords or altering the query structure wasn't producing more reasonable results.

## 4.0 Results and Analysis

Table I shows our results for the TREC routing queries. Since our system doesn't really rank the retrieved documents, we think the Table I presentation is more representative of our performance than the 11pt averages. The first and last columns are the topic number and description. The second column, "# Rel", is the number of relevant documents as judged by NIST in the Volume II Corpus. The next three columns give an indication of how the field did on the topic. "TRW Rel" is the number of relevant documents we submitted out of the "TRW Submit" we sent in for each topic. Our scores are summarized as follows:

| High | Above Med | Median | Below Med | Low |
|---|---|---|---|---|
| 8 | 15 | 4 | 19 | 2 |

Unlike most of the TREC participants, we did not submit the full 200 allowed documents for each topic. This turned out to be a major blunder because the TREC scoring procedure did not reward this self restraint. Many of our queries were too restrictive, achieving high precision at the expense of recall and a good score for the conference.

This problem comes about because of the binary nature of the FDF's evaluation of a query against a document. To operate properly against a routing data stream, it is necessary to execute several queries for each topic, with each successive query aiming for higher recall. When our queries were "tuned" properly, the results were quite good. Considering only those queries where we made the full submission, the distribution is well above the median.

| High | Above Med | Median | Below Med | Low |
|---|---|---|---|---|
| 6 | 8 | 3 | 5 | 0 |

Our analysis shows that we did well on topics where the ability to find phrases, acronyms, numbers, and alphanumerics were important. We had the high score on topics 28 and 29, both which involved finding references to AT&T. Since we retain and scan the full data stream, we didn't have to worry about an indexing parser splitting "AT&T" into "AT" and "T" and then throwing them both away. Our PSL subquery to find AT&T was

TABLE I - TRW TREC Results for the Routing Topics

| Qry | # Rel. | Relevant Retr. @ 200 Best | Median | Worst | TRW Rel | TRW Submit | Description |
|---|---|---|---|---|---|---|---|
| 01 | 216 | 62 | 30 | 0 | 49 | 200 | Pending Antitrust |
| 02 | 384 | 72 | 43 | 1 | 72 | 200 | Foreign Acquistions |
| 03 | 431 | 167 | 84 | 8 | 161 | 200 | US-Japan Joint Vent. |
| 04 | 48 | 33 | 18 | 2 | 10 | 49 | Debt Rescheduling |
| 05 | 150 | 116 | 38 | 10 | 80 | 141 | Japanese Dumping |
| 06 | 137 | 78 | 45 | 15 | 44 | 200 | Debt Relief |
| 07 | 169 | 87 | 63 | 1 | 63 | 200 | US Budget Deficit |
| 08 | 159 | 43 | 18 | 3 | 28 | 72 | Econimic Projection |
| 09 | 638 | 117 | 87 | 8 | 91 | 200 | Candidate Sightings |
| 10 | 233 | 153 | 110 | 15 | 153 | 188 | AIDS Treatments |
| 11 | 196 | 89 | 52 | 7 | 35 | 128 | Space Program |
| 12 | 262 | 103 | 54 | 4 | 75 | 200 | Water Pollution |
| 13 | 112 | 111 | 46 | 5 | 111 | 113 | Mitsubishi Heavy |
| 14 | 203 | 85 | 48 | 0 | 46 | 56 | Drug Approval |
| 15 | 624 | 114 | 80 | 17 | 89 | 200 | CEOs |
| 16 | 88 | 44 | 24 | 1 | 1 | 13 | Mkt Agrochemicals · |
| 17 | 303 | 154 | 81 | 0 | 33 | 58 | Agrochemical Cntls |
| 18 | 147 | 61 | 31 | 2 | 41 | 147 | Japan Stock Trends |
| 19 | 985 | 161 | 102 | 74 | 85 | 140 | Global Stock Trends |
| 20 | 403 | 178 | 124 | 5 | 161 | 178 | Patent Infringement |
| 21 | 47 | 44 | 35 | 0 | 12 | 29 | Superconductors |
| 22 | 466 | 162 | 120 | 14 | 32 | 45 | Counternarcotics |
| 23 | 100 | 74 | 41 | 5 | 37 | 54 | Legal Problems |
| 24 | 345 | 113 | 59 | 11 | 11 | 21 | Medical Technology |
| 25 | 71 | 34 | 14 | 0 | 15 | 36 | Chernobyl Effects |
| 26 | 313 | 122 | 49 | 1 | 47 | 122 | Multimedia Stds |
| 27 | 232 | 109 | 91 | 10 | 80 | 200 | AI in business |
| 28 | 332 | 89 | 47 | 14 | 89 | 200 | ATT in Comp/Comm |
| 29 | 142 | 79 | 13 | 0 | 79 | 200 | Foreign Acq ATT Tech |
| 30 | 269 | 92 | 48 | 17 | 57 | 88 | OS/2 Problems |
| 31 | 156 | 66 | 31 | 0 | 36 | 200 | OS/2 Advantages |
| 32 | 119 | 52 | 15 | 0 | 6 | 12 | Outsourcing |
| 33 | 462 | 147 | 71 | 17 | 71 | 200 | Doc Mngt Capable |
| 34 | 303 | 129 | 104 | 6 | 107 | 200 | ISDN Entities |
| 35 | 270 | 139 | 113 | 0 | 98 | 200 | Postscript Alts |
| 36 | 158 | 110 | 50 | 0 | 10 | 11 | Optical Disk Tech |
| 37 | 409 | 189 | 158 | 19 | 158 | 200 | SAA Components |
| 38 | 810 | 169 | 120 | 37 | 98 | 156 | Mini/Main Roles |
| 39 | 501 | 184 | 117 | 24 | 142 | 156 | Client-Server Plans |
| 40 | 800 | 150 | 121 | 16 | 87 | 200 | IS Impact on Orgs |
| 41 | 144 | 34 | 11 | 0 | 31 | 200 | Comp/Comm Upgrade |
| 42 | 696 | 131 | 92 | 10 | 92 | 112 | End User Computing |
| 43 | | | | | | 125 | AI Conferences |
| 44 | 241 | 105 | 35 | 1 | 105 | 200 | Layoffs at Companies |
| 45 | 304 | 103 | 71 | 0 | 103 | 200 | CASE Suceed/Fail |
| 46 | 51 | 40 | 31 | 9 | 30 | 200 | Virus Outbreaks |
| 47 | 237 | 80 | 35 | 2 | 80 | 200 | Contract > $1 mil |
| 48 | 189 | 48 | 28 | 2 | 17 | 40 | Purch Comm Equip |
| 49 | 139 | 65 | 56 | 4 | 44 | 131 | Who's in Supercomp |
| 50 | 26 | 12 | 1 | 0 | 4 | 5 | Virtual Reality |

```
define ATT '[AT\&[|amp\;]T|AT and T|\
            American Telephone [and|\&] Telegraph]' end
```

The "[|amp;]" notation means to allow an optional "amp;" as was present in the Ziff database. We had the high score on Topic 10 "AIDS Treatments". This may be due to our ability to easily find phrases like "acquired immune deficiency syndrome" or "AIDS related complex" in close proximity to drug names like "TPA", "5-fluorouracil", or "AZT".

We had the high score on topic 13 to find documents about Mitsubishi Heavy Industries. Our query that found 111 of the 112 documents the NIST judged relevant, was simply to find the two word phrase "Mitsubishi Heavy". Apparently the other TREC participants had trouble either finding phrases or determining the need to find phrases during query generation. The following sections discuss in detail topic 47 where we achieved the high score, and topic 36 where we achieved a low score.

### 4.1 Example of Good Performance - Topic 47

Topic 47 was to find documents discussing new contracts for computer systems in excess of $1 million. We found 80 good documents out of 200 submitted, the high score for this topic. We believe we did well on this topic because we were able to look for various numeric representations of $1 million in close proximity to keywords for new contracts and computer systems.

```
To be relevant, a document must identify the selection of a
source for the development or delivery of information systems
products or services valued at more than $1 million dollars.
```

The PSL query for this topic used three subqueries: one each for the "selection of a source", "information systems products or services", and "more than $1 million dollars". The PSL definitions were:

```
define award {3 words -> '[sign|award]*' and "contract"} end

define computer
"[computer|communic|network|phone|telecomm|mainframe|\
Starlan|PBX|Cyber|IBM 3090|X\-MP|Y\-MP|SCS\-40|information]" end

define million  {1 word -> "[million|billion] dollar" or
"\$[| ][|[0-9]][|[0-9]][0-9][|\.][|[0-9]][|[0-9]][|[0-9]]\
   [million|billion]" or
"\$[|][|[0-9]][|[0-9]][0-9][|\,][0-9][0-9][0-9][|\,]\
   [0-9][0-9][0-9]"} end
```

The "award" definition requires the root words "sign" or "award" to be within 3 words of "contract" in the text. This word count includes stop words, acronyms, or any other alphanumerics that were in the original text. This definition will find phrases like:

```
a contract was awarded
AT&T signed a new contract
Bellcore was awarded three new contracts.
```

The "computer" definition looks for any of the root terms shown. Note that looking for alphanumerics such "X-MP" or "IBM 3090", which may include multiple character white

space, is no problem. The "million" subquery uses proximity and an alphanumeric sequence pattern and will find items like the following:

```
a million dollar contract
a $2.3 million system
a $ 12 billion program
a $2000000 machine
a $ 2,000,000 machine
```

Note that the phrase "a 2,000,000 dollar award" would not be found by this definition. This was an oversight. The winning query was then simply

```
{50 words -> award and computer and million}
```

This finds documents which contain a 50 word sliding window in which all three subqueries match. Note how the "award" subquery that uses a 3 word sliding window can be nested inside a query using a 50 word sliding window.

### 4.2 Example of Bad Performance - Topic 36

Topic 36 was to find documents discussing how rewritable optical disks work.

```
To be relevant, a document must describe how rewritable
optical disk technology works at length and in significant and
comprehensive technical detail.
```

This topic was particularly challenging because the topic narrative describes attributes the documents must have rather than specific concepts or keywords. We started by defining a subquery to find documents mentioning rewritable optical disks.

```
define optical_disk
   {10 word -> "rewrit" and "optical[disk|drive|technolog]"} end
```

To find documents that describe the technology "at length", we wrote a subquery to find places where there were at least 5000 characters between the <TEXT> definition and the </TEXT> marker.

```
define LONG TEXT {5000 char -> no TEXTEND} end
```

To find documents that contained "significant and comprehensive technical detail" we manually extracted a list of keywords (Table II), and required that the documents to have at least 10 or more of these terms present.

The tightest query (intended for high precision) was

```
{1 document -> optical_disk and LONG and 30+ <technical terms> }
```

The loosest (intended for high recall) was

```
{1 document -> optical_disk and 10+ <technical terms> }
```

TABLE II - Subterms used for Topic 36

```
"amorphous";                    "crystalline";
"[ISO|CCITT]";                  "operation";
"Kerr effect";                  "phase[| |\-]change";
"SCSI";                         "phenomenon";
"bias";                         "polarit";
"binary";                       "polarized";
"capacity";                     "principle";
"chemical";                     "reflect";
"states";                       "refresh";
"cycles";                       "[sector|track|cylinder]";
"density";                      "[silver|gold]";
"spatial";                      "Oersteds";
"High Sierra";                  "surface reflectance";
"dye[| |\-]polymer";            "phase[| |\-]change";
"Curie temperature";            "thin film";
"gadolinium";                   "terbium";
"lanthanide";                   "magnetization";
"birefringence";                "substrate";
"emerging technolog";           "speed";
"erasable";                     "transfer";
"fatigue";                      "transluscent";
"field";                        "winchester";
"[frequency|Mhz]";              "[mega[|.]byte|[^[a-z]]MB[^[a-z]]]";
"inductance";                   "[giga[|.]byte|[^[a-z]]GB[^[a-z]]]";
"[jukebox|autochanger]";        "magneto[| |\-]optical";
"laser";                        "media";  "magnet"
```

### 4.3 Failure Analysis - Topic 36

Unfortunately, even our high recall query retrieved only 11 documents in the Volume II Corpus of which 10 were judged relevant. (The 11th was discussing WORM technology and only mentioned "rewritable optical drives" in passing.)

Upon examination of the NIST judgements, we made several observations about the relevant documents. First, we missed the keywords "erasable" as a synonym for "rewritable" and "video" as a synonym for "optical". Second, the assessor accepted articles about "optical recorders" and "optical image processing" systems. To pick up these corrections we would change the optical_disk subquery to read as follows:

```
define optical_disk {10 words ->
    "[rewrit|erasable]" and
    "[video|optical]"   and
    "[disk|drive|technolog|recorder|image processing]"} end
```

We then threw out the length restriction and reran the query requiring differing numbers of the technical terms to be present. The results from these runs are shown in Table III. This table shows two things. First, for this topic, the number of technical terms is an excellent "knob" to adjust the precision and recall. Second, the assessor was making a loose interpretation of "comprehensive technical detail". If we'd completely ignored this part of

the query, we would have had 135 good documents out of 262. Turning in only 200, we'd expect to have around 103 relevant, which would have been near the high score.

Table III - Topic 36 Results as a Function of the Number of Technical Terms

| Num Tech Terms | Rel Ret | Docs Ret | Prec | Recall |
|---|---|---|---|---|
| 10 | 26 | 31 | 0.84 | 0.17 |
| 9 | 34 | 42 | 0.81 | 0.22 |
| 8 | 41 | 59 | 0.69 | 0.26 |
| 7 | 50 | 77 | 0.65 | 0.32 |
| 6 | 58 | 100 | 0.58 | 0.37 |
| 5 | 77 | 139 | 0.55 | 0.55 |
| 4 | 96 | 176 | 0.55 | 0.62 |
| 3 | 111 | 210 | 0.53 | 0.71 |
| 2 | 120 | 222 | 0.54 | 0.77 |
| 1 | 131 | 240 | 0.55 | 0.84 |
| 0 | 135 | 262 | 0.52 | 0.87 |

## 5.0 Future Plans

During 1993 we hope to continue researching and evaluating better methods for query construction. Our objectives will be:

- Design and test a method of sequencing the execution of FDF queries to insure that 200 documents will be retrieved for each topic,
- Develop methods and algorithms to semi-automate manual query construction,
- Use the extensive relevance judgements from TREC-I to test techniques to generate FDF queries from statistical analysis of the relevant documents for each topic, and
- Examine the feasibility of using the FDF's term weighting capability to allow it to act as a back-end processor for other text retrieval techniques.

## 6.0 Acknowledgments

The FDF system is the result of extensive development by many people over the last 8 years. My role has been that of a reporter on the basic system's capabilities and the manner in which they might be applied to a TREC-like problem.

# Combining Evidence from Multiple Searches

Edward A. Fox, M. Prabhakar Koushik,
Joseph Shaw, Russell Modlin and Durgesh Rao
Department of Computer Science
Virginia Tech, Blacksburg, VA 24061-0106

**Abstract**

At Virginia Tech and PRC Inc. investigations with TREC data have focused on developing and comparing mechanisms for combining evidence related to a number of search schemes. Our work with the first CD-ROM has included various indexing, weighting, retrieval, combination, evaluation, and failure analysis efforts. Related work reported elsewhere in the proceedings by Paul Thompson discusses extensions undertaken by PRC Inc. and an evaluation of those results. Future work will develop our ideas further, try them out with additional data, and hopefully be evaluated in connection with other work on TREC and TIPSTER.

## 1 Overview

The 1992 TREC effort at Virginia Tech was carried out largely on a **DECstation 5000 Model 25** with 40 MB of RAM. The 1985 version of the **SMART** retrieval system, with numerous of our enhancements, was used for indexing, retrieval, and evaluation.

Our efforts were divided into two main phases. Prior to the TREC meeting we worked solely with the *Wall Street Journal* (WSJ) on the first CD-ROM. Thus, in Phase 1, we made eight different types of runs employing three different retrieval models – the Boolean model, the p-norm model and the vector space model — and three different weighting schemes. The queries for the Boolean and p-norm runs were manually generated by project team members. Vector queries were generated automatically from the topic descriptions. The results from these runs were then merged together to provide combined result sets. Relevance judgements were also performed on a subset of the retrieved documents to help with training studies.

In Phase 2, after the TREC meeting, we experimented with all five collections on the first CD-ROM. However, based on our work with the WSJ, we restricted our investigation to five out of the original eight cases. We also explored use of limited training information in carrying out the merging of results. Work is continuing at Virginia Tech to incorporate the results of our many runs into merged selections, to improve performance.

Subsequent sections of this paper describe all of these activities in greater detail.

## 2 Indexing and Data Structures

This section outlines the indexing done with the document collection provided by NIST. Due to limitations of available disk space, only Disc 1 was used during the experimental runs. The documents were indexed on a **DECstation 5000/25** using an enhanced version of the 1985 release

of the **SMART** Information Retrieval System [1]. The following specifications were used during the indexing process:

1. No stemming was done.
2. A stop word list of 418 words was used.
3. The Heading, Text, and Summary sections were included.
4. A controlled vocabulary was not included.

Briefly, the document text is tokenized, stop words are deleted, and non-noise words are included in the term dictionary along with their occurrence frequencies. Each term in the dictionary has a unique identification number. A document vector file is also created during indexing which contains for each document its unique ID, and a vector of term IDs and term weights. The weighting scheme itself is fairly flexible and can be changed to one of several schemes after the indexing is complete. Indexing the WSJ created a dictionary of approximately 15 MB and a document vector file of 121 MB. The other 4 collections take up space proportional to their sizes.

# 3 Retrieval Approach

## 3.1 Retrieval Runs

Several retrieval runs were then made as outlined below:

- Retrieval based on the vector model
  The topics were indexed considering the Description, Narrative, and Concepts sections to form vector queries. Retrieval was performed by varying the weighting on the document and query vectors. Three different methods were tried:

  1. Weighting with *tf*
  2. Weighting with *(tf/max(tf)) * idf*
  3. Weighting with *(0.5 + 0.5 * tf/max(tf)) * idf* — called **aug_norm**

  where
  *tf* = term frequency, and
  *idf* = inverse document frequency.

  During retrieval, the query-document similarities were computed in two different ways for each of these weighting schemes: cosine and inner product.

- Retrieval based on the Boolean model
  Boolean queries were manually generated by team members (with a background in computer science), and a retrieval run was made using these queries. The queries were composed, for the most part, using the entire text of the topic descriptions provided, and occasionally broader/narrower terms were obtained from general domain knowledge. The Boolean operators used were *AND* and *OR*.

Table 1: Summary of Retrieval Runs

| Name | Model | Sim. Function | Weighting Scheme |
|---|---|---|---|
| bool | Boolean | Boolean | binary term weights |
| pnorm1.0 | p-norm | p-norm | binary term weights, p=1.0 |
| pnorm1.5 | p-norm | p-norm | binary term weights, p=1.5 |
| pnorm2.0 | p-norm | p-norm | binary term weights, p=2.0 |
| cosine.atn | vector | cosine | aug_norm * idf |
| cosine.nnn | vector | cosine | tf |
| inner.atn | vector | inner product | aug_norm * idf |
| inner.nnn | vector | inner product | tf |

- Retrieval based on the p-norm model
  The Boolean queries described above were also used for the p-norm runs. Retrieval runs were made with three different p-values: 1.0, 1.5, and 2.0. No query term or clause weights were used during the p-norm runs.

The different runs are summarized in Table 1. Note that in Phase 1 of our efforts, we used all eight runs listed. In Phase 2, however, we focused on the *pnorm1.0* case, **with** document weighting, and the four vector runs.

### 3.2 Weighting Schemes

The weighting schemes mentioned above are detailed in Table 2.

Table 2: Weighting Scheme Options

- Term frequency normalization. This has the following choices:

| | |
|---|---|
| (n)one | $new\_tf = tf$ |
| (b)inary | $new\_tf = 1$ |
| (m)ax_norm | $new\_tf = \frac{tf}{max\_tf}$ |
| (a)ug_norm | $new\_tf = 0.5 + 0.5 * \frac{tf}{max\_tf}$ |

- Document weights. This has the following choices:

| | |
|---|---|
| (n)one | $new\_wt = new\_tf$ |
| (t)fidf | $new\_wt = new\_tf * log(\frac{num\_docs}{coll\_freq})$ |
| (p)rob | $new\_wt = new\_tf * log(\frac{num\_docs - coll\_freq}{coll\_freq})$ |

- Document vector normalization. This can be either of:

| | |
|---|---|
| (n)one | $norm\_wt = new\_wt$ |
| (s)um | $norm\_wt = \frac{new\_wt}{\Sigma new\_wt}$ |

This allows for a very flexible approach to changing the document vector weights as can be seen from Table 3.

Table 3: Matrix of weighting options

| Combinations | Term frequency | Doc. weights | Doc. vector norm. |
|---|---|---|---|
| nnn | tf | none | none |
| ntn | tf | tfidf | none |
| npn | tf | prob | none |
| bnn | binary | none | none |
| btn | binary | idf | none |
| bpn | binary | prob | none |
| mnn | max_norm | none | none |
| mtn | max_norm | idf | none |
| mpn | max_norm | prob | none |
| ann | aug_norm | none | none |
| atn | aug_norm | idf | none |
| apn | aug_norm | prob | none |

### 3.3 CPU time

The retrieval runs required approximately 4 minutes of CPU time for each topic. We did a full sequential pass through the document vector file for this since we did not have enough disk space for the inverted file.

### 3.4 Combination of Run Results

Our original plan was to compare several schemes for combining the results of a number of runs. The results of using one scheme, the CEO Model, are reported elsewhere in these proceedings by Paul Thompson. One of our goals was to use an artificial intelligence methodology called *Decision Trees*.

Decision Trees should reflect the most effective evaluation methods for each query. Our Decision Trees were produced by a commercial software package called KnowledgeSEEKER [2], by FirstMark Technologies Limited. Input to KnowledgeSEEKER is a training set of documents for which the results of the various evaluation runs are known along with the relevance values of the documents. The output of KnowledgeSEEKER is a Decision Tree that indicates in what order to apply the evaluation runs in order to determine the relevance of documents in the collection. The highest level of the tree is the test that most effectively evaluates documents for a given query. Based on the results of the most effective test, documents may be assigned a relevance score or additional tests that further evaluate the documents may be suggested. The Decision Tree partitions the collection into disjoint document groups with relevance values for the documents in each group. The results produced by KnowledgeSEEKER for the training set of documents must be parsed in order to apply the Decision Trees to other documents in the collection.

In Phase 1, training data was fed into the Decision Tree system so that the ranges of values of independent variables (e.g., similarity for a particular type of search) could be categorized into sets or intervals that predict the dependent variable values (e.g., relevance value of 0 or 1). For example, one Decision Tree developed during Phase 1 is given below.

Figure 1: Decision Tree Example for Query 52

```
RULE_1 IF
COSINE-2 = [0.032,0.07)
THEN
RELEVANCE = 0      73.3%
RELEVANCE = 1      26.7%


RULE_2 IF
COSINE-2 = [0.07,0.195]
THEN
RELEVANCE = 0       9.5%
RELEVANCE = 1      90.5%
```

This indicates that the likelihood of relevance is about .27 for very low values from the second cosine run, and about .91 for higher values from that same cosine run. When selecting this tree, the Decision Tree method suggests that ranking solely based on this cosine run would be wise. More complicated Decision Trees resulted for a number of queries, where several of the base runs' values had to be consulted. Unfortunately, no full ranking of run results using the Decision Trees could be completed in time for this report, so other, simpler methods were applied.

In Phase 1, a simple scheme was used for the results that were turned in. Essentially, the best results from each of the runs were included, until 200 distinct documents were found, for each query. This scheme is referred to as Ad Hoc Merge in discussions below.

In Phase 2, a more complex system was explored, called Recall-Precision (R-P) Merge. Details and results are given in Section 6.

## 4 Systems description

The main machine used for the indexing and retrieval runs was a **DECstation 5000 Model 25** with 40MB of RAM. This is a MIPS **R3000** CPU running at 25MHz. The total disk space used for the project was on the order of 3 GB.

## 5 Results of Phase 1

Due to limitations of disk space, only a subset of the collection comprising of Disc 1 of the *Wall Street Journal* was used during Phase 1 experimental runs. Relevance judgements were performed on a subset of this data by team members, in order to obtain a large set of training information. These were compared with the NIST judgement data and showed very high correlation. (Almost 90% of the documents we judged relevant were judged relevant by NIST.) In any case, the NIST judgments were used in the official (November 18, 1992) evaluation of our Phase 1 system, shown in Figure 2.

Figure 2: Phase 1, Disc 1, WSJ, Queries 50-100, Ad Hoc Merge

```
            Top ranked evaluation
Run number:        vpidt2  all
Num_queries:          50
Total number of documents over all queries
    Retrieved:    10000
    Relevant:      4056
    Rel_ret:       1371
Recall - Precision Averages:
    at 0.00       0.5153
    at 0.10       0.2841
    at 0.20       0.1898
    at 0.30       0.1285
    at 0.40       0.0988
    at 0.50       0.0746
    at 0.60       0.0527
    at 0.70       0.0338
    at 0.80       0.0271
    at 0.90       0.0222
    at 1.00       0.0222
Average precision for all points
   11-pt Avg:     0.1317
Average precision for 3 intermediate points (0.20, 0.50, 0.80)
    3-pt Avg:     0.0972
Recall:
    at  5 docs:   0.0476
    at 15 docs:   0.0640
    at 30 docs:   0.0843
    at 100 docs:  0.2050
    at 200 docs:  0.4481
Precision:
    At  5 docs:   0.3160
    At 15 docs:   0.2147
    At 30 docs:   0.1760
    At 100 docs:  0.1246
    At 200 docs:  0.1371
```

# 6 Results of Phase 2

## 6.1 Base Runs

In Phase 2, twenty-five base runs were made on Disc 1: 5 different retrieval methods were used for each of the 5 sub-collections. Based on our evaluation, the 11-point averages are given in Table 4. Note that in the p-norm case, document weights were utilized, in contrast to the binary weighting used in Phase 1.

Table 4: Base Runs, Phase 2, Disc 1, 11-point averages

| Run/Collection | AP | DOE | FR | WSJ | ZF |
|---|---|---|---|---|---|
| cosine.atn | 0.1138 | 0.0543 | 0.0259 | 0.2740 | 0.0813 |
| cosine.nnn | 0.1890 | 0.0330 | 0.0504 | 0.2184 | 0.0946 |
| inner.atn | 0.1241 | 0.0609 | 0.0405 | 0.3224 | 0.0888 |
| inner.nnn | 0.1478 | 0.0252 | 0.0108 | 0.1329 | 0.0101 |
| pnorm1.0 | 0.3006 | 0.0876 | 0.0727 | 0.3085 | 0.1448 |

Table 5: Base Runs + Similarity Merge

| Run/Collection | AP | DOE | FR | WSJ | ZF | Sim-Merge |
|---|---|---|---|---|---|---|
| cosine.atn | 0.1138 | 0.0543 | 0.0259 | 0.2740 | 0.813 | 0.1149 |
| cosine.nnn | 0.1890 | 0.0330 | 0.0504 | 0.2184 | 0.0946 | 0.1513 |
| inner.atn | 0.1241 | 0.0609 | 0.0405 | 0.3224 | 0.0888 | 0.1717 |
| inner.nnn | 0.1478 | 0.0252 | 0.0108 | 0.1329 | 0.0101 | 0.0075 |
| pnorm1.0 | 0.3006 | 0.0876 | 0.0727 | 0.3085 | 0.1448 | 0.1831 |

It should be noted that the p-norm run results were best in almost all situations for the given collection (except for WSJ, where inner.atn was slightly better).

## 6.2 Similarity Merge

TREC evaluations were to be done for the entire Disc 1 contents, so it is necessary to combine the results of the 5 sub-collections into an overall Disc 1 evaluation. This is a difficult matter, since each collection has a different number of relevant documents, and each was indexed separately. In 1993 we expect to consider this problem in more detail.

As a first solution to the problem we decided on the simplest possible approach — combine the results based on similarity. Thus, for a particular retrieval approach, we merged all of the documents retrieved from the 5 sub-collection runs, sorted the 1000 documents found for each query based on similarity, and returned the 200 with the highest similarity values.

For convenience, Table 5 shows the data from Table 4, but with an added column to show the Similarity Merge results.

Clearly, some improvement is needed in this collection merging process. One might use training based on the number of relevant documents in a collection, to predict a prior probability of finding a relevant document in that collection, and then use that to temper the similarity values.

## 6.3 Recall-Precision Merge

Another type of merger involves combining the several retrieval runs for a given sub-collection. To improve upon the Ad Hoc Merge used in Phase 1, we elected to train the merging process using recall-precision results. Thus, we considered the retrospective case of using the recall-precision tables from our evaluations, to help determine which runs to draw from for merging.

In particular, we use the following Recall-Precision Merge algorithm:

1. For each run to be merged, store the top 200 items on a stack, with the highest rank at the bottom of the stack.

Table 6: Base Runs + Similarity Merge + R-P Merge

| Run/Collection | AP | DOE | FR | WSJ | ZF | Sim-Merge |
|---|---|---|---|---|---|---|
| cosine.atn | 0.1138 | 0.0543 | 0.0259 | 0.2740 | 0.813 | 0.1149 |
| cosine.nnn | 0.1890 | 0.0330 | 0.0504 | 0.2184 | 0.0946 | 0.1513 |
| inner.atn | 0.1241 | 0.0609 | 0.0405 | 0.3224 | 0.0888 | 0.1717 |
| inner.nnn | 0.1478 | 0.0252 | 0.0108 | 0.1329 | 0.0101 | 0.0075 |
| pnorm1.0 | 0.3006 | 0.0876 | 0.0727 | 0.3085 | 0.1448 | 0.1831 |
| R-P Merge | 0.2268 | 0.0554 | 0.0521 | 0.2555 | 0.1003 | 0.1523 |

2. Associate with each run an estimate of the probability that the top item on the stack is relevant. Initially, this value is the precision value at recall 0.0 from our evaluation.

3. To draw the next item for the merged result, identify the stack with the highest estimated probability of relevance, pop the top of the stack, and update the probability estimate.

4. To update the probability estimate, use interpolation based on the number of popped items (i.e., the number "retrieved") and the rest of the recall-precision results.

5. Continue the process starting again with step 3, as long as less than 200 items have been drawn.

The results of applying this "R-P Merge" algorithm are given in Table 6, on the last line. Note that the first 5 columns of that line show the results of merging for a particular collection, and the very last value reflects Similarity Merge followed by R-P Merge.

From Table 6 we see that the R-P Merge method does not yield results that are as good as the best individual run. In particular, we could simply use the *pnorm1.0* results uniformly and do better than with merging.

Further improvement in the above algorithm, possibly yielding more accurate estimates in step 4, will be investigated in 1993. Other studies will consider if recall-precision data for each query could be used in a similar training situation, for subsequent testing on Disc 2.

# 7 Evaluation

## 7.1 Software engineering

We began with the 1985 version of SMART, and have enhanced it. We tried for a long period of time to use the new version of SMART on an RS/6000 but found the use of disk space to be excessive. Since we could not get reliable results, we went back to the older version.

We underwent extensive software development since May. This included writing of C programs and Unix shell scripts to partially automate indexing, retrieval, relevance judgement making, merging, evaluation and tabulation of results.

## 7.2 Problems and Failure Analysis

Problems encountered in the project were partially identified. A failure analysis was performed on a subset of documents that failed to be included in our result set in Phase 1. Observations regarding our merging methods came from further studies in Phase 2. The following observations aim to summarize our findings.

- Disk Space Limitations:
  We had problems acquiring the required disk space in time for the submission of results due to lateness of the promised award from DARPA. A great deal of time was wasted in making partial runs using a hodgepodge of computers, with NFS mounting of remote files slowing processing by almost an order of magnitude. Ultimately we started over when told that work on the WSJ would suffice.

  This forced us to submit a subset of results in Phase 1, and to only complete work on Disc 1 during Phase 2. With more disk space we could have use inverted files for indexing the data and that would have made things much faster. It would have allowed real time interactive searching which would have speeded up and improved the quality of our relevance judgement operation. Also, with more disk space, we could have used an RS/6000, to get runs done more quickly, assuming SMART could be ported and made fully operational on that platform.

- Merging of Retrieval Runs:
  Ideally, an effective method to predict relevance like the Decision Tree or CEO model should be used to take the results from the various runs and determine the relevant documents based upon the training set and the relevance judgements. However, in Phase 1 we did not have enough time to properly implement these, so we settled on a less effective approach – using the ranks of the documents from each of the runs. The top N documents from each run were included in the results set, ordered based upon the ranks. The value of N was determined by the number of document number repetitions across the various runs. The average value of N was near 40.

  This method used to combine the results from the various runs was flawed. Using the top N documents from each run assumes that each of the runs was of equal quality, which is usually not the case. Each poor run would have many non-relevant documents in the final list, while a quality run could have many relevant documents miss the cutoff of N. The similarity values should be used to determine the top-ranked documents to pull from each run, but the combinations of runs with various incompatible similarity measures made this a non-trivial task.

  In Phase 2, we used additional training data, namely the recall-precision average values from the evaluation of each run whose results were to be merged. This approach, too, is flawed, since the averages reflect general trends, while query-specific trends would be much more appropriate to use.

- Faulty Queries:
  No feedback modification could be performed to improve retrieval performance. The Boolean queries proved to be too general in many cases. The p-norm runs were made with the Boolean queries; separate p-norm queries (i.e., that made use of user-assigned p-values and query weights) would have improved retrieval effectiveness. The vector queries were generally good, but also had some faults. In particular, shorter vector queries might have led to better Similarity Merge behavior by decreasing the likelihood of spurious matches in sub-collections with few relevant documents. Also, the NOT clauses from the topics descriptions were included in the vector queries, which may have contributed to retrieval of non-relevant documents. The effect of bad queries and an inadequate combination method was lower retrieval quality than desired.

### 7.3 Possible enhancements

Because of the disk space problem we were not able to do many of the tests we planned to do, even by the end of Phase 2. Work will continue in 1993 now that new disks have been received. Among the planned tasks are:

- Phrase Identification and Matching
  In the current system, phrases are handled by using an **AND** query term. For example, **Information AND Retrieval** was used for **Information-Retrieval**. Due to the absence of a proximity operator, this leads to retrieval of non-relevant documents where these words occur widely apart. The retrieval results can be improved by providing a mechanism for dealing with phrases explicitly, and/or the use of proximity operators.

- Better Base Methods
  In addition to considering the use of phrases, further study of base runs, considering query construction, indexing, weighting schemes, and lexical information, will be undertaken. Of particular interest is the use of p-norm queries, which if tailor-made, might well out-perform the vector queries in all collections. Contrasts between stemming and morphological analysis can also be made.

- Merging Methods
  While the Ad Hoc and the R-P Merge methods are not based on elaborate theory, they do provide insight into the effects of combining results. Further refinement of the approaches, and additional testing to obtain upper-bound performance values, will be undertaken.

- The CEO Model
  The **Combination of Expert Opinion (CEO)** model [3, 4] of Thompson can be used to treat the different retrieval runs as experts, and combining their weighting probability distributions to improve performance. This could be used in a variety of ways to combine results from a variety of runs and indexing schemes.

## References


[1] C. Buckley. Implementation of the SMART information retrieval system. Technical Report 85-686, Cornell University, Department of Computer Science, May 1985.

[2] FirstMark Technologies Limited. *KnowledgeSEEKER User's Guide.* FirstMark Technologies, 14 Concourse Gate Site 680, Ottawa, Ontario, Canada, 1990.

[3] P. Thompson. A Combination of Expert Opinion approach to probabilistic information retrieval, Part 1: The conceptual model. *Information Processing & Management*, 26(3):371–382, 1990.

[4] P. Thompson. A Combination of Expert Opinion approach to probabilistic information retrieval, Part 2: Mathematical treatment of CEO Model 3. *Information Processing & Management*, 26(3):383–394, 1990.

# Multilevel Ranking in Large Text Collections Using FAIRS

**S-C. Chang, H. Dediu, H. Azzam, M-W. Du**
**GTE Laboratories**

***Abstract***

A description of a general-purpose multilevel ranking information retrieval prototype is presented. The methods used in weighting and ranking the retrieved documents are discussed. Experiments with the TREC92 collection of text and queries have been conducted without manual preprocessing. Initial results have shown the multilevel ranking scheme to be highly competitive in precision and recall relative to other ranking strategies.

## 1.0 Introduction

Information retrieval research at GTE Laboratories has led to the development of a prototype system called FAIRS (Friendly Adaptable Information Retrieval System). FAIRS has evolved into a functional and flexible system currently running on SunOS, HP/UX, Ultrix, AIX, VMS and PC platforms. It has been used in environments as diverse as literature searching, library operations, research and development, customer support, market analysis and management. FAIRS is being further tested as a retrieval engine for very large collections of text, such as those presented by the TREC92 collection and wide-area distributed collections.

FAIRS is designed to minimize user effort in the preparation of text, the learning of query syntax while providing a user-modifiable multilevel ranking scheme[1]. Experiments with the TREC92 collections of text and queries have been conducted with no human intervention in the processing of either text or queries. The results of experiments against the collection of Wall Street Journal articles are listed in Section 3.4.

1. Patent Pending

## 2.0 System Overview

FAIRS uses pure text as its information base, while allowing flexible links into non-textual information. FAIRS extracts information out of an unstructured, amorphous collection of data, in four main steps:

1. First, it partitions (logically) a raw text file or a collection of such files into retrievable record units. This simple record partitioning is necessary and sufficient for indexing to begin. The goal is to use source information as-is [1].
2. Second, FAIRS automatically constructs an index. A feature exists where deletions are permitted from an index. Statistics on the collections can also be generated at this time. Such statistics are used in normalizing the weighting of retrieved documents.
3. The third step involves the user queries. Queries are accepted and interpreted in an intelligent and sensitive manner [1,2]. A flexible approach to the understanding of the query is essential to providing good responses.
4. Finally, once the query is processed, the relevant hits are retrieved quickly and displayed in an ordered list ranked according to a relevance measure. The records corresponding to the hits can then be viewed, printed or mailed in their entirety on demand.

### 2.1 Characteristics

In FAIRS, both the responses to information requests and the way relevance is determined can be customized by the user. FAIRS can provide a tool for decision making by presenting to the user all the relevant facts in an elegant and timely fashion. There are some interesting and novel features of FAIRS and research issues associated with each of these strategies. They will be discussed in sequence.

### 2.1.1 Using Files As-Is

Raw textual information is broken into records (i.e., the units that will be subsequently retrieved). Consistent with the goal of using the information as-is (i.e., eliminating file conversion), FAIRS does not require the input files to be in any particular format [1]. Any ASCII file can be indexed and searched. However, if a file does have a logical field structure that the user wants to use in searching (e.g., restrict a search to text in the NAME field of records), then this structure must be described to FAIRS.

If the information files do not have "record" or "field" structures then an implicit method must be used to partition these files into records. To use the files as-is, FAIRS allows the user to impose a record structure. Any character string(s) can be designated as an end-of-record marker(s). Such a page/record structure has been used successfully and a number of documents have been quickly transformed into textbases that could be processed by FAIRS.

Alternatively, a fixed line count, such as 66 lines or 78 lines, can be used as an end-of-record marker. Furthermore, each file may be declared as an individual record if multiple files are present. Records can be further divided into fields by using any character string(s) as field markers.

For retrieval, FAIRS requires the information base being searched to be partitioned into indexed, retrievable records. Either the record/field structure in the original text file can be used or, in the absence of such structure, it can be imposed. In either event, a description of record/field format will control how FAIRS breaks the raw text file into records. Note that a file format description is separate from the described input file which can be used as-is (without conversion).

These methods of record definition are the result of practical experience with text collections generated in typical business environments. A flexible input text structure or pre-processor is essential to the effectiveness of a general-purpose information retrieval system.

### 2.1.2 Stop Lists

FAIRS constructs an index to support the full text retrieval of records. A user-defined stop list can be used to limit the words indexed. For the TREC92 experiments the stop word list consisted of 280 words. It was customized with the addition of several common embedded SGML strings and spurious field definitions such as: *docno, doc, text, journal, title, summary, descriptor, author, fileid, file_id, &bullet, &para, &degrees*, etc.

### 2.1.3 Stemming

Words are stemmed at index time and at query parsing time. The stemming rules used are published by Paice [4]. The rules are incomplete but are claimed to be satisfactory. Words shorter than 4 characters are not stemmed.

Conflation is used as an alternative to the inadequacies of truncation. Paice's scheme does not return "correct" roots, but does ensure that members of a family reduce to the same root, and members of different families reduce to different roots.

### 2.1.4 Query Formation

FAIRS handles free association queries for information [1,2]. Free association is the essence of being "user friendly" in information retrieval, rather than graphical user interface features such as windows, icons, mice, and pulldown menus. Free association means that there is no set of keywords that must be known and used exactly. Queries do not have to be phrased as Boolean expressions with ANDs and ORs. Words, phrases, and even word stems thought to be relevant are listed in a free association fashion. A FAIRS query has the simple form:

$$term_1\,[w_1]\,[,]\,term_2\,[w_2]\,[,]\,\ldots term_n\,[w_n]$$

Where $term_i$ can be a single word or a phrase and $w_i$ is a user-specified weight for that word. There will be a hit for each record in the information base that contains at least one word from the query.

Commas are the only delimiters used to delineate phrases, so that relative adjacency relation among search words (i.e., the distance between two words that appear as a query term such as "scenic view" or "software engineering") could be considered automatically. The weight $w_i$ is used at ranking time but if omitted, it is set by default to 1.

### 2.1.5 Previewing Records

The ranking results are shown to the user via several interactive and selectable preview features. The preview features show some aspects of the top ranked records. The aspects include the first (or all) lines containing query words, coverage, and frequency data or simply the first few lines of highly ranked records. This is designed to give users a better feel for how relevant the retrieved records really are, and thus give him a feel for how well do the ranking rules work, and how they

should be customized. After examining a preview or the full text of the highly ranked records, the user can then revise the query or even change the ranking strategy.

#### 2.1.6 Synonyms

A synonym definition capability (glossary) is also available within FAIRS. Given a word, FAIRS will retrieve instances of that word, as well as instances of its synonyms. The option can be invoked to broaden a query. Users can build their own vocabulary or invoke and modify the system-wide synonym dictionary. This can be used to ease the problem of different word usages from different people.

A very fast elastic string matching algorithm[2] [5] is being evaluated for inclusion among the query expansion features of FAIRS.

#### 2.1.7 Displaying Records

When a request is made, users will always be presented with the retrieved records in their full-text form. Furthermore, FAIRS provides several ways to associate non-text information with each record. There are basically two types of links: implicit and explicit. Implicit links use source information as-is while explicit links involve special fields embedded in the source text. Implicit links are useful in situations where an implied one-to-one mapping may be established between records and image files. Explicit links may be used to express one-to-many relations between records and other media.

#### 2.1.8 Ranking

One of the most interesting aspects of FAIRS is its unconventional ranking scheme to determine the relative relevance of retrieved records. The ranking scheme is designed to mimic the human relevancy judgement process.

When a person is asked to determine the relative relevance between two records he is likely to first weigh them using a set of criteria. If the two records have the same weight with the set of criteria, a secondary set of criteria may be used to differentiate them, and so on. The criteria used can be highly heuristic. The adaptation of this "multilevel ranking scheme" has been filed with the Patent office in the United States.

To enable FAIRS to use free association in place of Boolean semantics, a multilevel ranking model [1,2] for full-text information retrieval has been developed and implemented. FAIRS ranks records with respect to a particular query according to a set of rules. The default rules consist of six attributes in six levels. The six attributes are the *importance*, *popularity*, *frequency*, *location* of a search word, and *record size*, and *record ID* of the record it occurs in. Each attribute may have either *positive*, *negative* or no impact (*neutral*) on the relevance judgement of a record. Such arrangement also guarantees the automatic consideration of coverage (i.e., percentage of different query words covered by record), which is next to impossible to implement in a Boolean environment [1,2]. The ranking rules of FAIRS are always accessible and modifiable by the user.

Descriptions of the attributes chosen for FAIRS follow:

*FREQUENCY*: The number of occurrences of the keyword in the record. This attribute may be used to reflect interest in finding records with more repetitions of a given term. That is, when set to have a *positive* impact, the more instances of a term in a record, the more relevant the record. Therefore, the record with the higher *frequency* of a term is more likely to be retrieved.

*IMPORTANCE*: FAIRS provides the searcher the ability to assign an arbitrary weight (*importance*) to each query keyword; thus the user has additional control over how records are retrieved. For example, specifying *breakfast:4* assigns a weight of 4 to the term *breakfast* in a query. In general, keyword weighting allows the searcher to change how FAIRS sorts records in order to identify the most relevant ones. If a keyword is not weighted, FAIRS supplies a default weight which is in reverse proportion to its input position in the query (words that came to mind first deserve more weight). Therefore, if the searcher chooses to weight some or all keywords in a query with a range of weights, records strong in the heavily-weighted keywords are ranked before others. This attribute is defaulted to have a *positive* impact on the relevance judgement.

*POPULARITY*: This is the number of times a term occurs in the entire collection, as opposed to the number of times it appears in the retrieved record. For example, if the word *software* appeared, at least once, in 15 records in a collection, its popularity is considered to be 15. This attribute is usually used in the *negative* sense and, by default, FAIRS assumes that the more popular a term is, the less effective it is in retrieval.

*REC_ID*: The record ID is the location of a record in the collection. It may indicate the age of the record.This is also useful when the records are arranged according to their degree of significance (in either increasing or decreasing order.) This attribute is a good example for

2. Patent Pending

showing there is no perfect ranking rules that work in all situations. This attribute may have either a *positive* or *negative* impact, depending on user intention. Although there is no obviously better default setting for this attribute, we chose *positive* to be the default, so as to give priority to the later records as they are likely to be more timely.

*REC_SIZE*: The total word count of the record. This attribute may be used to counter (normalize) the size advantage that a larger record may have over smaller ones during rank judgement. A larger record may have more keywords simply because it contains more words. It is therefore we set its default to have a negative impact on the relevance judgement. Of course, when records are of similar size, this attribute will have minimal effect, and should probably be disabled to improve response time.

*WORD_LOC*: The location of the first occurrence of the keyword in the record. A *negative* setting (the default) of this attribute assumes that important words appear in the beginning of a collection, or record. For example, headings or titles which contain keywords describing the contents of a document, usually appear at the beginning. Of course, this depends solely on how the contents of the information are organized. This serves as another example of the context-sensitivity of the ranking process.

The following table shows the default ranking rules:

**Table 1: Ranking Attributes Settings for TREC92**

| level | Imp | Pop | Freq | Size | ID | Loc |
|---|---|---|---|---|---|---|
| 1 | --- | --- | --- | --- | --- | --- |
| 2 | pos | --- | --- | --- | --- | --- |
| 3 | --- | neg | --- | --- | --- | --- |
| 4 | pos | neg | pos | neg | --- | --- |
| 5 | --- | --- | --- | --- | --- | neg |
| 6 | --- | --- | --- | --- | pos | --- |

The first level, having no attribute values, automatically accounts for coverage if the weight computation is done using Method 1, as described below in section 2.1.9.

To perform TREC92 experiments, we changed the ranking rules as follows: *Size* was introduced on level one to balance the effect of the very large records found in the Federal Register. Consequently, we also had to specify impacts for the *Importance*, *Popularity* and *Frequency* attributes on the first level since they are the most important criteria for ranking.

The following table shows the TREC92 ranking rules:

**Table 2: Ranking Attributes Settings for TREC92**

| level | Imp | Pop | Freq | Size | ID | Loc |
|---|---|---|---|---|---|---|
| 1 | pos | neg | pos | neg | --- | --- |
| 2 | pos | neg | --- | neg | --- | --- |
| 3 | pos | neg | pos | neg | --- | --- |
| 4 | --- | --- | --- | --- | --- | neg |
| 5 | --- | --- | --- | --- | pos | --- |

#### 2.1.9 Weight computation

Two methods were available to compute the weight of a document:

- Method 1

The weight of a retrieved record $r$ at level $l$ is determined by the following formula:

$$W_l[r] = \sum_{i=1}^{K} \left[ e[i] \prod_{j=1}^{A} a[j]^{s[j]} \right]$$

Where:

$e[i]$ = 1 if the $i$th keyword exists in the record and 0 otherwise,

$a[j]$ = The value of the attribute $j$,

$A$= Number of attributes (currently 6,)

$K$ = Total number of query keywords,

$s[j]$ = 1 if the value of attribute $j$ is positive,

0 if the value of attribute $j$ is neutral,

-1 if the value of attribute $j$ is negative.

In other words, the weight at a certain level is the sum of the product of the attributes at that level. This weight computation method automatically calculates coverage since for each keyword $e[i]$, the product of attributes is never 0.

The value of the attribute is configured by the user either before running FAIRS, or before delivering the query. To set the value before running FAIRS, a file must be created containing the initial values. A default value is set during system start-up.

- Method 2

One disadvantage of method 1 is that it lacks a common reference scale that evenly distributes the influence of the attributes on the weight of the document. For example, consider a textbase with one very large record, record

number l, and assume the following settings for the variable attributes:

FREQUENCY = *neutral* ($s[1] = 0$)

POPULARITY = *neutral* ($s[2] = 0$)

WORD_LOC = *neutral* ($s[3] = 0$)

REC_ID = *neutral* ($s[4] = 0$)

REC_SIZE = *positive* ($s[5] = 1$)

The contributing factors to the weight of a retrieved document are IMPORTANCE and REC_SIZE. Because of its large size ($a[5] >> 0$,) record *l* will be irrelevantly retrieved in response to a large percentage of pending queries.

To avoid the above scenario, method 2 uses the following formula to compute the normalized weight of a retrieved record *r* at level *l*:

$$W_l[r] = \sum_{i=1}^{K} \left[ e[i] \sum_{j=1}^{A} \frac{a[j]\, s[j]}{a_m[j]\, C} \right]$$

Where:

$e[i]$ = 1 if keyword $i$ exists in the record and 0 otherwise,

$a[j]$ = The value of the attribute $j$,

$A$ = Number of attributes (currently 6,)

$K$ = Total number of query keywords,

$s[j]$ = 1 if the value of attribute $j$ is positive,

0 if the value of attribute $j$ is neutral,

-1 if the value of attribute $j$ is negative.

$C$ = Number of attributes whose value is not 0.

$a_m[j]$ = The maximum value of the attribute $j$.

To avoid the effect of disproportionately large attribute values, $a[j]/a_m[j]$ is set to 1 if $a[j]$ is one of the largest 2% attribute values. Method 2 has the advantage of accommodating large attribute values by normalizing with respect to their maximum.

However, this method does not automatically calculate the coverage since $s[j]$ contributes as a multiplicand rather than as an exponent as in Method 1. Therefore, if the attributes are all neutral (level 1 in Table 1--default attributes,) the weight would be 0, evan though query terms would occur in the record *r*. We are therefore still looking for ways to improve the method.

## 3.0 Experiments

The TREC92 collection of text and topics was used to quantify and analyze the performance of FAIRS in the domain of very large collections. The entire collection includes 2.3Gb of text from various sources and of various formats.

### 3.1 System Configuration

The hardware platform used for indexing and query processing was an IBM RS/6000 320 workstation with AIX 3.0 operating system. The core memory (RAM) installed was 32 Mb. The net disk space available was 4,169,728Kb. The CPU clock rate was 25 MHz, with a MIPS rating of 27. The disk access time was 9.8 ms on average.

### 3.2 Indexing Performance

Our experiments showed that performance in indexing is strictly constrained by I/O wait. We used several techniques to reduce this constraint and optimize throughput. We scheduled index runs to run simultaneously, thus keeping the CPU busy during I/O wait. We used a kernel which employs automatic disk caching (providing an order of magnitude improvement in indexing time.) To further exploit caching, large amounts of high-speed core memory should be used (our limit of 32 Mb is by no means ideal.)

The I/O bottleneck also implies that the very fastest disks be used. We chose 9.8 ms disks as the fastest available for the platform for a reasonable price. Using the above configuration, the sustained indexing throughput was on the order of 10 days (240 hrs.)/Gb. The storage overhead for index output was 110%. That is, for 1 Gb of text, 2.1 Gb of space should be available before indexing is initiated. By this measure, the entire TREC92 collection of 2.3 Gb would require 4.83 Gb of storage. Since only 4.17 Gb was available, we participated in category B (Wall Street Journal only.)

### 3.3 Query processing

Before submitting the raw topics to FAIRS, they were converted automatically into FAIRS-compatible queries of the form described in section 2.1.4. The TREC92 topics were pre-processed by a simple syntactic term token generator. The pre-processor removed stop words, stemmed and removed cases from topic terms. It used an ad-hoc term weighting scheme to assign IMPORTANCE weights to terms according to their positions in the topic (e.g. terms occurring in the title section were assigned an arbitrarily higher importance than those in other sections.)

The importance weights were assigned with a minimum of attention. As described below, failure analysis had later shown that great improvements in precision could be made if pre-processing were better tuned with term expansion, duplicates removal and a heuristic importance measure.

After conversion, queries were submitted to the retrieval engine through a non-interactive interface. The time for processing (ranking) was approximately 30 minutes per query. Again, I/O wait was the dominant factor in running time. The same system solutions which apply to indexing can be applied with equal effectiveness to query processing. Modification of some of the internal data structures for index storage (concordance list optimization) could also improve ranking running time.

### 3.4 Retrieval Performance

FAIRS participated in the following categories: Ad-hoc Wall Street Journal disk 1 (AW1), Ad-hoc Wall Street Journal Disk 2 (AW2), Routing, Category B (RB). Relevance judgements are as follows: AW1: 50 topics, AW2: 50, RB: 25; total 125.

Of these judgements, FAIRS' recall rates were ranked as follows: 77 on or above median, 45 below median, and 3 undetermined (1 tied, 2 with 0 relevant.) 61.6% on or above median, 36% below, 2.4% undetermined.

**Table 3: Recall Performance Summary**

| Category | %>= med | %< med | % rel ret |
|---|---|---|---|
| AW1 | 66.0 | 32.0 | 38.5 |
| AW2 | 46.0 | 50.0 | 54.7 |
| RB | 84.0 | 16.0 | 30.0 |
| **Total** | **61.6** | **36.0** | **38.8** |

Beside recall, relevancy judgements made available recall/precision figures across 11 points of recall rates. The average of the 11 recall/precision figures is called the *11-pt. average*. FAIRS' 11-pt. average rates were ranked as follows: 79 on or above median, 43 below median, and 3 undetermined. The percentages for 11-pt. averages are: 63.2% on or above, 34.4% below, 2.4% undetermined.

**Table 4: 11-pt. Average Performance Summary**

| Category | %>= med | %< med |
|---|---|---|
| AW1 | 85.7 | 31.0 |
| AW2 | 45.8 | 54.2 |
| RB | 84.0 | 16.0 |
| **Totals** | **63.2** | **34.4** |

#### 3.4.1 AW1

Ad-hoc WSJ disk 1, results were submitted by three systems. In this category, of all systems' submissions, 4,056 documents were judged relevant, 10,000 were submitted by FAIRS in response to 50 queries, of which 1,561 were among the relevant. The average 11 point average (over 50 queries over 11 recall rates for each query) was 0.2083.

The distribution of relevant retrieved (recall) over the 50 topics was: 20 ranked best, 12 on the median, 16 worst, 2 tied. The distribution of 11-pt. averages over the 50 topics was: 22 ranked best, 14 on the median and 13 worst, 1 tied.

**Table 5: AW1 Recall/Precision Performance**

| | Best | Med | Worst |
|---|---|---|---|
| Recall | 20 (40%) | 13 (26%) | 16 (32%) |
| 11-pt. | 22 (44%) | 14 (28%) | 12 (24%) |

Total recall placed FAIRS second among 3 participants, and first in 11-pt. averages recall/precision.

#### 3.4.2 AW2

Ad-hoc WSJ disk 2, results were submitted by two systems. In this category 2,172 documents were judged relevant, 10,000 were submitted in response to 50 queries, of which 1,188 were among the relevant. The 11 point average precision was 0.2216. The distribution of relevant retrieved (recall) over the 50 topics was: 17 ranked best, 22 worst, 9 tied, 2 unevaluated. Of the tied, the 11-pt. averages favored FAIRS 6 times, the other system 3 times.

**Table 6: AW2 Recall/Precision Performance**

| | Best | Worst | Tie |
|---|---|---|---|
| Recall | 17 (34%) | 22 (44%) | 11 |
| 11-pt. | 22 (44%) | 26 (52%) | 2 |

For those queries for which there was a tie in recall values, there are two queries which had 0 records judged relevant. Of the remaining 9, we considered the 11-pt. average as a tie-breaker. The result was 6 best, 3 worst. Combining the recall and 11-pt. averages, for AW2, FAIRS had 23 submissions on or above the median (46%), 25 below (50%) (4% undetermined).

### 3.4.3 RB

Routing, Category B results were submitted by 7 systems. For those topics judged, 3,766 documents were considered relevant, 5,000 were submitted by FAIRS in response to 25 queries. Of those submissions, 1,124 were among the relevant.

The distribution of relevant retrieved (recall) over the 25 topics was: 2 ranked best, 13 ranked above the median, 6 on the median, 4 below, for a total of 21 on or above the median, 4 below.

**Table 7: RB Recall/Precision Performance**

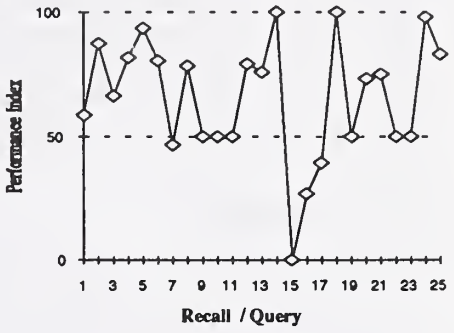
| Relation to Median | > | = | < |
|---|---|---|---|
| Recall | 15 (60%) | 6 (24%) | 4 (16%) |
| 11-pt. | 15 (60%) | 6 (24%) | 4 (16%) |

This is the only group that had enough participants to make a comparative-performance analysis meaningful. We compared our 11-point average and recall rates for each query to the best, the median, and the worst scores of that query. The performance index (*PI*) is calculated as follows:

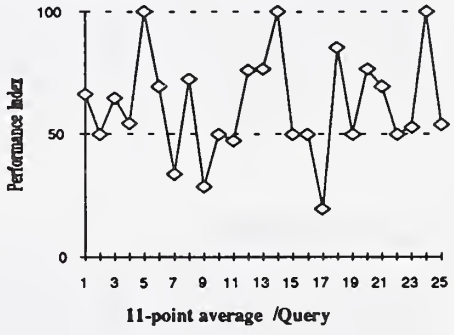
$$PI = \begin{cases} 50 + 50\left(\dfrac{score - median}{best - median}\right), & score \geq median \\ 50\left(\dfrac{score - worst}{median - worst}\right), & score < median \end{cases}$$

*PI* has the property that a value of 100 means the best is achieved, and a 50 means the performance is on the median, and a 0 means it is the worst.

The following graph illustrates the performance index of the recall rates of FAIRS compared to the group. It shows FAIRS is above average most of the time. The average recall *PI* of FAIRS is 65.8.




The next graph shows the performance index of the 11-point average of FAIRS compared to the group. It again shows FAIRS to be above average most of the time. The average 11-point-average *PI* of FAIRS is 61.8.




## 3.5 Failure Analysis

Based on the feedback from relevance judgements, we are considering several improvements in the query handling and ranking methods. These changes include:

1. Expanding of terms in the topics which are abbreviated via an abbreviation dictionary. Initial investigation of topics which have abbreviations reveals that those abbreviations had an appreciably negative impact on recall. Topic 17 is a good example, where the term "United States" is abbreviated as "U.S.", In a later trial, this simple expansion alone significantly improved the recall rate for this topic.
2. Using better term weighting based on heuristics. Up to 50% improvement was observed when term weighting was modified more intuitively (by hand.)

3. Considering more terms in the topics based on their popularity.
4. Using synonyms to expand terms.
5. Optimizing the ranking rules. This will be important in Category A participation. The diversity of the documents in Category A will necessitate the use of optimum ranking rules which minimize the negative effects of extreme attribute values. These ranking rules will require more experimentation and analysis.

## 4.0 Conclusion

Various techniques used by FAIRS to retrieve, weight and rank documents were discussed. The system was shown to successfully process the TREC92 collection and achieve a high degree of precision and recall in the categories it participated, relative to other systems in those categories.

The system was described, the ranking rules and weighting methods were also presented. The application of FAIRS to the TREC92 collection was analyzed in terms of system and retrieval performance. The retrieval performance was compared to competing systems in the same categories. It is hoped that further analysis of the results will provide insight into which features of FAIRS contribute the most to its effectiveness.

## 5.0 References


1. Chang, S. C. and A. Chow, "Towards a Friendly Adaptable Information Retrieval System," *Proceedings of RIAO 88*, March 1988, pp. 172-182.
2. Chang, S. C., "Adaptive Ranking: A Spreadsheet Approach." Technical Report TR00171288500, GTE Laboratories, Waltham, MA, Dec. 1988.
3. Chow, A. and S. C. Chang, "Text Structure Specification: A Study to Eliminate File Conversion," Technical Note TN87500.03, GTE Laboratories, Waltham, MA, Dec. 1987.
4. Paice, C. D., *Information Retrieval and the Computer*, MacDonald and Jane's Computer Monographs, London, 1977.
5. Du, M. W. and S. C. Chang, "A Model and Fast Algorithm for Multiple Errors Spelling Correction," Technical Report TR0650989500, GTE Laboratories, Waltham, MA, Sept. 1989.

# Description of the PRC CEO Algorithm for TREC

Paul Thompson
PRC Inc., Mail Stop 5S3
1500 Planning Research Drive
McLean, VA 22102
Phone: 703/556-1923
Email: thompson_paul@po.gis.prc.com

This paper describes work done on the TREC project at PRC Inc. in collaboration with Professor Edward Fox and his colleagues at Virginia Polytechnic Institute and State University (VPI&SU). The reader should refer to the description of their system included in this proceedings for further details on the common processing of the TREC data shared by PRC and VPI&SU (Fox et al. 1993). PRC developed an algorithm, the Combination of Expert Opinion (CEO), which combined the results of VPI&SU's runs. VPI&SU used a different combination technique for their final results. Originally the intent was that the CEO algorithm would be integrated with the SMART system used by VPI&SU. Both upper and lower level combination of results would take place, i.e., at the lower level of individual document features within a particular retrieval method and the upper level of combination of the output of the individual methods themselves, i.e., the various cosine and p-norm methods used by VPI&SU. Furthermore we had originally hoped to train the CEO algorithm, so that the weighting of the various methods would be optimized based on relevance judgments. For the official TREC results we were only able to use the upper level of CEO without any training. Since then we have done additional retrospective experiments in which the different methods are weighted in the CEO algorithm by one of several measures of their performance for TREC.

## Combination of Expert Opinion

The statistical technique of CEO provides a solution to the problem of combining different probabilistic models of document retrieval. This technique is expected to result in improved precision and recall over that provided by any one model, or method, since research has shown that various retrieval models retrieve different sets of relevant documents (Katzer et al. 1982, Fox et al. 1988). In the Bayesian formulation of the CEO problem (Lindley 1983) a decision maker is interested in some parameter or event; and he/she has a prior, or initial, distribution or probability for that parameter or event. The decision maker revises the distribution upon consulting several experts, each with his/her own distribution or probability for the parameter or event. To effect this revision, the decision maker must assess the relative expertise of the experts and their interdependence, both with each other and the decision maker. The experts' distributions are considered as data by the decision maker, which is used to update the prior distribution.

For automatic document retrieval, the retrieval system is the decision maker and different retrieval algorithms, or models, are the experts (Thompson 1990a,b, 1991). This is referred to as the upper level CEO. At the lower level the probabilities of individual features, e.g., terms, within a particular retrieval model can be combined using CEO. In lower level CEO the retrieval model is the decision maker and the term probabilities are viewed as lower level experts. The probability distributions supplied by these lower level experts can be updated, according to Bayes theorem, by user relevance judgments for retrieved documents. These same relevance judgments also give the system a way to evaluate the performance of each model, both in the context of a single search of several iterations and over all searches to date. These results can be used in a statistically sound

way to weight the contributions of the models in the combined probability distribution used to rank the retrieved documents. Since various algorithms, such as p-norm, are expressed in terms of correlations rather than probability distributions, it was necessary to extend the CEO algorithm to handle correlations. So far this extension has been handled in a heuristic fashion. If a retrieval method, e.g., one of the cosine methods, returned a value between 0 and 1 as a retrieval status value; the logistic transformation of this weight was interpreted as an estimate of the mean of a logistically transformed beta distribution which was provided as evidence to the decision maker. Since there was no basis with which to assign a standard deviation to this distribution, as called for by the CEO methodology, an assumption was made that all standard deviations were .4045, a value corresponding to a standard deviation of .1 in terms of probabilities.

All of the retrieval methods used by VPI&SU were combined with the CEO algorithm except for the Boolean. That is, we used weighted and unweighted cosine and inner product measures as well as p-norm measures of 1.0, 1.5, and 2.0. For measures, such as the inner product and some of the p-norm results that did not give a retrieval status value in the 0 to 1 range, the result was mapped to this interval by scaling the highest score of the method in question for a given topic to the highest score given by one of the cosine measures. Default scores half way between 0 and the lowest score achieved by a particular method were used for documents not retrieved in the top 200 in response to a given topic, since the actual score of these documents was unknown. The Boolean model was not included, because it was not a ranked retrieval method. In the future we plan to extend our normalization techniques to use the Boolean results as well. Figure 1 shows our summary official TREC results for topics 51-100 on the Wall Street Journal collection from the first CD-ROM.

Since TREC, we have experimented with weighting the different methods combined based on their performance with the TREC data. In other words we have attempted to determine an upper bound for performance based on knowledge of each method's performance on the actual test data. We used four different weighting schemes: the 11-point average, precision at 0.00 recall, precision at 0.10 recall, and unweighted (i.e., our official TREC results). We also tried using the five best methods, rather than the seven used for our official results, i.e., we excluded Pnorm1.5 and Pnorm2.0. None of the weights produced better results than the unweighted scheme. This was surprising. Figure 2 shows our summary results for CEO based on all seven methods using the 11-point average and additional relevance judgments made by VPI&SU. Figure 3 shows the same weighting scheme using only NIST relevance judgments.

Two immediate explanations suggest themselves. First, using overall averages may not be too useful. Second, our simple implementation of CEO assumes independence among the methods. To examine the first problem we intend to try weighting the methods on a topic by topic basis rather than by overall averages. Again this would be a retrospective upper bound experiment. In terms of the CEO approach (Thompson 1990a,b) using only overall averages would be analogous to using only feedback from past searches, while using topic-specific weights would correspond to receiving feedback over several iterations of the same search. We propose to investigate the second problem by analyzing the overlap of pairs of runs of the various methods to determine dependence and thus perform CEO without the independence assumption.

The PRC portion of our experiments were all run on a Sun SPARCstation 2 with 16 megabytes of RAM. The CEO code was written in g++.
Top ranked evaluation

Run number: prceo1 all
Num_queries: 50
Total number of documents over all queries
Retrieved: 10000
Relevant: 4056
Rel_ret: 1666
Recall - Precision Averages:
at 0.00 0.6598
at 0.10 0.4382
at 0.20 0.3451
at 0.30 0.2536
at 0.40 0.1753
at 0.50 0.1400
at 0.60 0.1016
at 0.70 0.0489
at 0.80 0.0185
at 0.90 0.0015
at 1.00 0.0015
Average precision for all points
11-pt Avg: 0.1985
Average precision for 3 intermediate points (0.20, 0.50, 0.80)
3-pt Avg: 0.1679

Recall:

at 5 docs: 0.0379
at 15 docs: 0.1193
at 30 docs: 0.1714
at 100 docs: 0.3438
at 200 docs: 0.4557

Precision:

At 5 docs: 0.4040
At 15 docs: 0.3653
At 30 docs: 0.3033
At 100 docs: 0.2322
At 200 docs: 0.1666

**Figure 1. TREC summary results without weights**

Top ranked evaluation

Run number: 1
Num_queries: 50
Total number of documents over all queries
  Retrieved: 10000
  Relevant: 3901
  Rel_ret: 1867
Recall - Precision Averages:
  at 0.00 0.5419
  at 0.10 0.3684
  at 0.20 0.3239
  at 0.30 0.2848
  at 0.40 0.1975
  at 0.50 0.1490
  at 0.60 0.0933
  at 0.70 0.0575
  at 0.80 0.0147
  at 0.90 0.0029
  at 1.00 0.0006
Average precision for all points
  11-pt Avg: 0.1850
Average precision for 3 intermediate points (0.20, 0.50, 0.80)
  3-pt Avg: 0.1625

Recall:
  Exact: 0.5400
  at 5 docs: 0.0270
  at 15 docs: 0.0890
  at 30 docs: 0.1592
  at 100 docs: 0.3603
  at 200 docs: 0.5400
Precision:
  Exact: 0.1867
  At 5 docs: 0.2360
  At 15 docs: 0.2920
  At 30 docs: 0.2853
  At 100 docs: 0.2234
  At 200 docs: 0.1867

**Figure 2. Post-TREC summary results with 11-point average weights and additional relevance judgments**

Top ranked evaluation

Run number: 1
Num_queries: 47
Total number of documents over all queries
  Retrieved: 9400
  Relevant: 3697
  Rel_ret: 1366
Recall - Precision Averages:
  at 0.00 0.3801
  at 0.10 0.2588
  at 0.20 0.2379
  at 0.30 0.1638
  at 0.40 0.1144
  at 0.50 0.0831
  at 0.60 0.0551
  at 0.70 0.0363
  at 0.80 0.0101
  at 0.90 0.0002
  at 1.00 0.0002
Average precision for all points
  11-pt Avg: 0.1218
Average precision for 3 intermediate points (0.20, 0.50, 0.80)
  3-pt Avg: 0.1104

Recall:
  Exact: 0.4114
  at 5 docs: 0.0137
  at 15 docs: 0.0497
  at 30 docs: 0.0800
  at 100 docs: 0.2249
  at 200 docs: 0.4114
Precision:
  Exact: 0.1453
  At 5 docs: 0.1915
  At 15 docs: 0.2156
  At 30 docs: 0.1879
  At 100 docs: 0.1683
  At 200 docs: 0.1453

**Figure 3. Post-TREC summary results with 11-point average weights and only NIST relevance judgments**

## References

Fox, E. A.; Koushik, P.; Shaw, J.; Modlin, R.; and Rao, D. 1993. "Combining Evidence from Multiple Searches" this proceedings.

Fox, E. A.; Nunn, G. L.; Lee, W. C. 1988. "Coefficients for Combining Concept Classes in a Collection" *Proceedings of the 11th. International Conference on Research and Development in Information Retrieval* June 13-15, Grenoble, France p. 291-307.

Katzer, J.; McGill, M. J.; Tessier, J. A.; Frakes, W.; DasGupta, P. 1982. "A study of the overlap among document representations" *Information Technology: Research and Development* vol. 2 P. 261-274.

Lindley, D.V. 1983. "Reconciliation of probability distributions." *Operations Research.* v. 31, no. 5, p. 866-880.

Thompson, P. 1990a. "A Combination of Expert Opinion Approach to Probabilistic Information Retrieval, Part 1: The Conceptual Model." *Information Processing and Management,* Vol. 26, No. 3, p. 371-382.

Thompson, P. 1990b. "A Combination of Expert Opinion Approach to Probabilistic Information Retrieval, Part 2: Mathematical Treatment of CEO Model 3." *Information Processing and Management,* Vol. 26, No. 3, p. 383-394.

Thompson, P. 1991. "Machine Learning in the Combination of Expert Opinion Approach to IR" In Birnbaum, L. and Collins, G. (eds.) *Machine Learning: Proceedings of the Eighth International Workshop (ML91)*, San Mateo, Morgan Kaufmann, p.270-274.

# Vector Expansion in a Large Collection

Ellen M. Voorhees and Yuan-Wang Hou
Siemens Corporate Research, Inc.
755 College Road East
Princeton, New Jersey 08540

**Abstract**

This paper investigates whether a completely automatic, statistical expansion technique that uses a general-purpose thesaurus as a source of related concepts is viable for large collections. The retrieval results indicate that the particular expansion technique used here improves the performance of some queries, but degrades the performance of other queries. The overall effectiveness of the method is competitive with other systems. The variability of the method is attributable to two main factors: the choice of concepts that are expanded and the confounding effects expansion has on concept weights. Addressing these problems will require both a better method for determining the *important* concepts of a text and a better method for determining the correct sense of an ambiguous word.

## 1 Introduction

In many retrieval systems the similarity between two texts is a function of the number of word stems that appear in both texts. While these systems are often efficient and robust, their effectiveness is depressed by the presence of homographs (words that are spelled the same but mean different things) and synonyms (different words that mean the same thing) in the texts. Homographs depress precision by causing false matches. Synonyms depress recall by causing conceptual matches to be missed. That is, if a query and a document are about the same topic, but use different words to express the idea, the document will not be retrieved in response to the query. We are investigating how *concept spaces*, data structures that define semantic relationships among ideas, can be used to mitigate the effects of synonymy and homography in retrieval systems designed to satisfy large-scale information needs.

We impose two constraints on our research with the goal of making the resulting methods more applicable to retrieving documents from large corpora. First, we want to keep human intervention in the indexing and retrieval processes at a minimum; therefore, we use strictly automatic procedures. Second, even automatic procedures need to be relatively efficient. We believe this efficiency requirement precludes the use of deep analyses of document content for the foreseeable future, and we restrict ourselves to statistical processing of the text and concept space.

There are effectiveness concerns when dealing with large corpora as well as efficiency concerns. Large corpora usually imply a diverse vocabulary, and thus the synonym and homograph problems are exacerbated. In this paper we investigate vector expansion as a solution to the synonymy problem for the large TREC collection. As the name "vector expansion" implies, we are working within the vector space model of information retrieval [5]: both documents and topics are represented as weighted vectors and the similarity between two texts is computed as the inner product of their respective vectors. The vectors are expanded by terms related to original text words in our concept space. In particular, since we are using the WordNet lexical database as our concept space, vectors are expanded by adding selected synonyms of original text words.

Although using thesauri to expand vectors has been done before (see, for example, [7], [2], [4]), it has always been done on small collections. We are interested in investigating whether comparatively simple

statistical expansion techniques are viable for large collections. The results so far indicate that our expansion technique can improve the performance of some queries, but is equally likely to degrade the performance of others. The sources of this variability are described in detail below. A description of WordNet and the expansion algorithm is given first to provide the appropriate context.

## 2 WordNet

WordNet is a manually-constructed lexical system developed by George Miller and his colleagues at the Cognitive Science Laboratory at Princeton University [3]. Originating from a project whose goal was to produce a dictionary that could be searched conceptually instead of only alphabetically, WordNet evolved into a system that reflects current psycholinguistic theories about how humans organize their lexical memories. The basic object in WordNet is a set of strict synonyms, called a *synset*. By definition, each synset a word appears in is a different sense of that word.

There are three main divisions in WordNet, one each for nouns, verbs, and adjectives. Within a division, synsets are organized by the lexical relationships defined on them. For nouns, the only division used in this study, the lexical relationships include antonymy, hypernymy/hyponymy (IS-A relation) and three different meronym/holonym (PART-OF) relations. The IS-A relation is the dominant relationship, and organizes the synsets into a set of approximately ten hierarchies[1]. Examples of synsets that are the heads of hierarchies are {*entity, thing*}, {*psychological_feature*}, {*abstraction*}, and {*possession*}.

The developers of WordNet specifically avoided including specialized vocabularies within WordNet; the coverage of "standard" English is quite good. The April, 1992 version of WordNet (the version used in this study) contains 35,155 synonym sets and 67,293 senses in the noun division. The majority of synonym sets are quite small (one or two members), but the more common nouns (i.e., those nouns that actually get used in documents and topics) tend to belong to the larger synsets. Example synsets from the noun division are shown in Figure 1. The lexical relationships that the synsets participate in, especially their parents in the IS-A hierarchy, differentiate among the senses.

We developed our own routine to access the WordNet information that differs somewhat from the access code distributed with WordNet. In our version, the access routine takes a word (a string of characters), converts it to lower case, and checks if the converted string occurs in the noun portion of WordNet. If the string is found, the routine returns either the number of synsets in which the string appears, the fact that the string is a known irregular morphological variant of a member of a synset (e.g., 'women' is an inflection of 'woman'), or both (e.g., 'media' is both a member of {*media, mass_media*} and an inflection of 'medium'). If the string is not found, several simple (regular) morphological variants of the word are tried. If none are found, the routine reports the string as not found. Otherwise, the routine returns the base form. A consequence of this simple strategy is that regular plural forms that are members of their own synsets do not return the synsets of the base word. For example, 'arms' returns the synsets {*coat_of_arms, arms, blazon, blazonry*} and {*weaponry, arms, implements_of_war*}, but not the four synsets for 'arm'.

## 3 Vector Expansion Procedure

For the retrieval results reported in this paper, both document and query vectors were expanded using synonyms of original text words. The particular expansion method we used is one of the most effective vector expansion methods among a wide variety of expansion schemes we tried on smaller collections. However, the TREC collection is much more diverse than those collections and some other scheme may be more effective on it. We intend to test some of those methods on the TREC collection in the near future.

We use the SMART retrieval system developed at Cornell as the basis for our retrieval system [1]. The SMART system is designed to facilitate information retrieval research by making it easy to substitute

[1]The actual structure is not quite a hierarchy since a few synsets have more than one parent.

surrogate (3 senses)
{*deputy, surrogate*}
{*surrogate*}
{*alternate, proxy, stand-in, substitute, surrogate, replacement*}

opinion (5 senses)
{*opinion, ruling*}
{*opinion*}
{*opinion, sentiment, persuasion, view*}
{*judgment, judgement, opinion*}
{*opinion, view*}

motherhood (1 sense)
{*motherhood, maternity*}

decision (4 senses)
{*decision*}
{*judgment, judgement, decision, judicial_decision*}
{*decision, deciding, decision_making*}
{*decision, firmness*}

court (4 senses)
{*court, courtyard*}
{*court, tennis_court*}
{*court, courtroom*}
{*court, tribunal*}

Figure 1: Example Synonym Sets

customized pieces of the program without needing to modify other components. For this work, we changed only part of the indexing module of the standard SMART system.

Indexing a piece of text proceeds as follows:

1. The text is broken into tokens by the standard SMART tokenizer.
2. Each token is passed in turn to a parser. The parser eliminates tokens designated as numbers, white space, or punctuation; the remaining tokens are assumed to be "words".
3. Each word is looked up in the standard SMART stop word list and is eliminated if it is found there. If the word is not a stop word, it is stemmed (using the SMART **triestem** stemming algorithm), assigned a concept number, and added to the list of concepts that will form the vector.
4. A word that is not a stop word is also looked up in the noun portion of WordNet before it is stemmed. If the word is in WordNet, the set of synonyms from all the synsets the word is a member of is produced. The elements of this set are also stemmed and assigned concept numbers. Instead of the concepts being inserted into the vector list, however, they are inserted into a different *relative list*. Relatives that come from original text words that have only one sense in WordNet (appear in exactly one synset) are flagged as such when they are entered into the list.

5. After all the words from a text have been processed, relatives that are flagged as being from a single-sense text word and relatives that have been added to the relative list at least twice are added to the vector list. The requirement to appear in the list at least twice if the relative is from a text word that has multiple senses is a poor-man's attempt at sense disambiguation. The idea is that is that if two original text terms agree on a relative, the relative is probably related to the correct sense of those text terms.

6. To produce the final weighted vector, the term frequency of each of the concepts in the vector list produced in step five is computed. Concepts that were added as relatives have their term frequency weight multiplied by .8 to emphasize the original terms. The term frequency weight of each concept is then multiplied by an inverse document frequency factor, and those weights are further normalized by the square root of the sum of the squares of the weights (cosine normalization). This weighting scheme is the "tfc" weights described by Salton and Buckley in [6].

As an example, take the text *court opinions and decisions on surrogate motherhood* (a paraphrase of topic 70). The vector produced for this text using the synsets shown in Figure 1 would contain the stems of *court, opinion, decision, surrogate, motherhood* (original text words), *maternity* (synonym of 'motherhood', which has only one sense), *judgment*, and *judgement* (synonyms of both 'decision' and 'opinion').

Both documents and topic statments were indexed by the procedure described above; no special manual processing of the topic statements was performed. The manually assigned keywords associated with some documents were not used in the indexing. For the topic statements only the Concepts, Description, Factors, Narrative, Nationality, and Title sections were indexed. Among those sections, no distinctions were made regarding what section a term appeared in.

The decision to expand both documents and query vectors, as opposed to only query vectors, is based on several factors. First, the WordNet synsets contain collocations such as 'judicial decision', but the tokenizer used recognizes only single words. For the collocations to participate in matches, both documents and vectors need to be expanded. Second, documents are frequently longer than topic statements. Since we require agreement on a relative before it is added to the vector, the longer documents provide more opportunities for a concept to be added to the vector. Third, in the experiments on smaller collections, expanding both documents and queries was consistently more effective than expanding only queries (although usually less effective than expanding neither). Document expansion has its costs, however, even excluding the obvious additional expense at indexing time. Longer vectors also increase storage costs and processing time at retrieval as well. Table 1 gives a histogram of the percentage increase in vector length as compared to an unexpanded collection for the TREC documents.

# 4 Experimental Results

We performed one retrieval run on the entire TREC database, retrieving documents for the 50 ad hoc queries. The official evaluation table for this run is given in Table 2. Using a Sun IPX with 64 megabytes of RAM, it took approximately 42 hours of processing to produce the inverted index of the document collection. The resulting inverted index takes 947 megabytes of disk storage. It took approximately one CPU second on average to index a topic statement and produce a query vector. The average retrieval time per query was 15 CPU seconds.

An analysis of the retrieval results show that the expanded collection is more effective than a corresponding unexpanded collection for some queries[2]. However, the effectiveness of the the expansion procedure is very variable, and the performance of other queries was degraded by the expansion. This variability is attributable to two main factors: the process of selecting which new concepts to add, and the confounding effects expansion has on concept weights.

[2]Evaluation results for the unexpanded collection were made available through the courtesy of the SMART group at Cornell University.

| % Increase | # Docs |
|---|---|
| 0–10 | 15656 |
| 10–20 | 52568 |
| 20–30 | 116766 |
| 30–40 | 162862 |
| 40–50 | 163235 |
| 50–60 | 123943 |
| 60–70 | 64029 |
| 70–80 | 24564 |
| 80–90 | 8872 |
| 90–100 | 3333 |
| > 100 | 6926 |
| > 200 | 207 |

Mean increase: 42%

Table 1: Histogram of Percentage Increase in Document Vector Length

Expansion affects both the inverse document frequency (IDF) and the term frequency (TF) components of the concept weights. A concept that is frequently added to documents is downweighted by its IDF factor relative to its weight in an unexpanded collection. Such a concept is often a very general concept and the downweighting is likely to be beneficial. Similarly, a concept that is occasionally added to documents, and occurs infrequently in the collection otherwise, is emphasized by its IDF component. This may or may not be beneficial, depending on the quality of the term. The aggregate TF component of a concept can be relatively larger in an expanded collection if the concept has many synonyms. This effect is common because the same word will frequently cause the same synonyms to be added in both the document and query vectors. Unfortunately, this effect is usually detrimental because the words occurring in large synsets are common words that contain little content. For example, if either 'couple' and 'pair' or the Roman numeral 'II' appears in a text, then the entire synset {*two, 2, ii, twain, couple, pair, twosome, duo, duet, brace, span, yoke, couplet, distich, dyad, duad, deuce, doubleton, craps, snake_eyes*} is added.

The effects of the changes in weights is illustrated by the performance of topics 95 and 70, the texts of which are given in Figures 2 and 3. Portions of the corresponding query vectors for both expanded and unexpanded collections are given in Figure 4. Topic 95 retrieved 28 relevant documents in the expanded collection; in the corresponding unexpanded collection, 17 relevant documents were retrieved. The improvement is due to increasing the weights of central themes of the topic, both by adding additional concepts (*outlaw, constabl*) and emphasizing existing concepts (*law, sleuth*). On the other hand, topic 70 retrieved only 32 relevant documents in the top 200 in the expanded collection while in the unexpanded collection 41 relevant documents were retrieved. The degradation is due to the downweighting of 'surrogate', which was added to many documents and thus has a smaller IDF weight in the expanded collection, and the increased weight for 'mother' (compounded by the addition of 'matern'), resulting in a marked preference for documents that contain mother, whether or not they also contain surrogate.

The major difficulty of the expansion process is controlling which original terms get expanded and which terms they are expanded by. In our algorithm, any word can be expanded if it occurs only once in WordNet or if there is another word that has a common synonym. Although the agreement criterion is imposed to prevent synonyms of the wrong senses of words from being added, it is not sufficient for the task. Furthermore, to save processing time we do not tag a word with its part of speech prior to looking it up in WordNet, so many words that are used as verbs and adjectives in the text are nonetheless found in the noun division of WordNet (frequently in only one sense!) and add spurious relatives. The consequence of these factors is that in addition to the concepts that are added for marginally useful words, concepts that have no bearing on the content of the text may also be added to its vector.

```
                Top ranked evaluation

Run number:        siems1  all
Num_queries:          50
Total number of documents over all queries
    Retrieved:     9992
    Relevant:     16400
    Rel_ret:       3393
Recall - Precision Averages:
    at 0.00        0.7524
    at 0.10        0.4670
    at 0.20        0.3242
    at 0.30        0.1973
    at 0.40        0.1179
    at 0.50        0.0672
    at 0.60        0.0241
    at 0.70        0.0000
    at 0.80        0.0000
    at 0.90        0.0000
    at 1.00        0.0000
Average precision for all points
   11-pt Avg:      0.1773

Recall:

    at  5 docs:    0.0131
    at 15 docs:    0.0344
    at 30 docs:    0.0650
    at 100 docs:   0.1687
    at 200 docs:   0.2564

Precision:

    At  5 docs:    0.5200
    At 15 docs:    0.4827
    At 30 docs:    0.4687
    At 100 docs:   0.4100
    At 200 docs:   0.3393
```

Table 2: Official Evaluation Results

**Computer-aided Crime Detection**
Document must describe a computer application to crime solving.
To be relevant, a document must describe either an actual or a theoretical computer application to detective work, by the police or by another law enforcement organization. A relevant document could include techniques such as profiling criminals and their methods of operation, identifying finger prints, spotting anomalies, etc.
1. police, detective, sleuth, enforcement agency
2. clue, records, fingerprints, methods

Figure 2: Text of Topic 95

**Surrogate Motherhood**
Document will report judicial proceedings and opinions on contracts for surrogate motherhood.
A relevant document will report legal opinions, judgments, and decisions regarding surrogate motherhood and the custody of any children which result from surrogate motherhood. To be relevant, a document must identify the case, state the issues which are being decided and report at least one ethical or legal question which arises from the case.
1. surrogate, mothers, motherhood
2. judge, lawyer, court, lawsuit, custody, hearing, opinion, finding

Figure 3: Text of Topic 70

Query Vectors for Topic 95

| Expanded Weight | Concept | Unexpanded Weight |
|---|---|---|
| .20626 | enforc | .25682 |
| .18793 | fingerprint | .23399 |
| .16088 | print | .11148 |
| .40100 | sleuth | .27738 |
| .10375 | law | .08331 |
| .08165 | felon | — |
| .08233 | crook | — |
| .13792 | constabl | — |
| .25840 | espial | — |
| .16667 | catch | — |
| .08366 | outlaw | — |

Query Vectors for Topic 70

| Expanded Weight | Concept | Unexpanded Weight |
|---|---|---|
| .15901 | decid | .08546 |
| .11605 | proceed | .06181 |
| .50365 | mother | .48892 |
| .14090 | case | .08597 |
| .56104 | surrog | .69887 |
| .24603 | judg | .12226 |
| .14946 | custod | .20529 |
| .10880 | matern | — |
| .18264 | judicial_decid | — |
| .10257 | legal_act | — |

Figure 4: Query Vectors for Sample Topics

As an example of these effects, consider document FR89512-0147, President Bush's 1989 Mother's Day Proclamation. This document was retrieved in response to topic 70 because it mentioned 'mother' or 'motherhood' 16 times. Figure 5 contains an excerpt of the document and a sampling of the concepts that were added. (Words that are in boldface in the excerpt are words that caused additional concepts to be added.) Approximately 60 of the 250 concepts in the vector were added by the expansion process. About half of the added concepts are the result of a wrong sense or a wrong part of speech being used in support of its addition, and another 20 of the added concepts are correct, but unimportant.

Unfortunately, the ratio of unimportant and mistaken additions to reasonable additions exhibited by document FR89512-0147 is not unusual. WordNet — and English — are rich enough such that it is likely for two words in a text to be synonyms of (different senses of) a third word. Using additional lexical relations compounds this problem: the experiments we conducted on smaller collections show a marked degradation in effectiveness if any any of the other relations represented in WordNet are used in addition to synonymy to expand a concept.

# 5 Conclusion

We have demonstrated a fully automatic, statistical expansion technique that is capable of improving the effectiveness of some queries relative to a corresponding unexpanded collection for a large, full-text collection. The overall effectiveness of the technique is competitive with other retrieval methods. However, the technique is hampered by its unpredictability, which has at least three sources:

- errors in selecting the correct sense, and therefore the correct relatives, of a text word,
- no determination of the relative importance of a word to the text before deciding to expand it, and
- the complex interaction between expansion and term weighting.

Since we believe the disambiguation of word senses to be the most fundamental of these three problems, and also useful in its own right, our current research lies in this direction.

# References


[1] Chris Buckley. Implementation of the SMART information retrieval system. Technical Report 85-686, Computer Science Department, Cornell University, Ithaca, New York, May 1985.

[2] Edward A. Fox. Lexical relations: Enhancing effectiveness of information retrieval systems. *SIGIR Newsletter*, 15(3), 1981.

[3] George Miller. Special Issue, WordNet: An on-line lexical database. *International Journal of Lexicography*, 3(4), 1990.

[4] G. Salton and M. E. Lesk. Computer evaluation of indexing and text processing. In Gerard Salton, editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*, pages 143–180. Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1971.

[5] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, November 1975.

[6] Gerard Salton and Chris Buckley. Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24:513–523, 1988.

[7] Yih-Chen Wang, James Vandendorpe, and Martha Evens. Relational thesauri in information retrieval. *Journal of the American Society for Information Science*, 36(1):15–27, January 1985.

Excerpt of document:

> ...
>
> A mother's love, while demonstrated **daily** in **acts** of tenderness and generosity, is always a **source** of wonder. Who can fathom the quiet thoughts of one who keeps in her **heart** a constant vigil over the child she has carried in her **womb**, rocked in her arms, and **watched** grow, with **eyes** full of worry, joy, and pride? Her devotion never fails to fill us with gratitude and awe.
>
> ...
>
> Today, we honor all those **women** who, by virtue of giving **birth**, or through adoption or marriage, are mothers. Each of us should let our mother know that she is ever close in our **hearts**, and that her many gifts to us are cherished and remembered_not only on Mother's Day, but throughout the year.
>
> In recognition of the contributions of all mothers to their **families** and to the Nation, the Congress, by a joint resolution approved May 8, 1914 (38 Stat. 770), has designated the second **Sunday** in May each year as Mother's Day and requested the President to call for its appropriate observance.
>
> ...
>
> IN **WITNESS** WHEREOF, I have hereunto **set** my hand this tenth day of May, in the year of our Lord nineteen **hundred** and eighty-nine, and of the Independence of the United States of America the two **hundred** and thirteenth.

Sample added concepts:

Reasonable additions

*womb* ⟶ *uterus*
*women* ⟶ *woman, adult_female*
*birth & families* ⟶ *parentage*

Unimportant additions

*eyes* ⟶ *eye, eyeball, oculus, optic, peeper, organ_of_sight*
*hundred* ⟶ *hundred, 100, c, century, one_c, centred*
*Sunday* ⟶ *sunday, sabbath, lord's_day, day_of_rest*

Mistaken additions

*acts* ⟶ *acts_of_the_apostles*
*daily* ⟶ *gazette*
*set & families* ⟶ *category, class, type*
*watched & heart* ⟶ *ticker*
*source & witness* ⟶ *informant*
*source & birth* ⟶ *beginning*

Figure 5: Concepts Added by the Expansion Procedure

# PROXIMITY-CORRELATION FOR DOCUMENT RANKING: The PARA Group's TREC Experiment

by Mark Zimmermann (zimm@alumni.caltech.edu)
P.O.Box 598
Kensington, Maryland 20895-0598
USA

## Abstract

The PARA Group's simple document routing method achieved surprisingly good results in the first TREC experiment. The system works by awarding points to documents with many query terms in near proximity to each other. The current implementation of this system is described in general terms; this note is followed by a listing of the complete source code used to rank documents for the 50 TREC test questions, written in Awk. Possible improvements, and directions for further research, are suggested.

## Acknowledgements

The PARA Group is a loose affiliation of people with common interests in free-text information retrieval, hypermedia, and free software. (For further information, or to join, send a message to "para-request@cs.cmu.edu" via the Internet.) For the TREC relevance-ranking document routing test, I consulted with other PARA Group members and implemented concepts that we discussed communally. I would like to thank Dr. Donna Harman, NIST, for allowing me to participate in TREC and for encouraging me to write up my results. I also thank the members of the PARA Group for their helpful advice. I made extensive use of, and am grateful for, software from the Free Software Foundation — in particular, the GNU Emacs text editing system, and the Gawk version of the Awk programming language. (Disclaimer: My Employer Is In No Way Responsible For This Work!)

## Approach

I began with the subjective observation that, in my personal experience, the documents which I like most tend to have local clusters of "interesting" words. I also began with the constraint that I had only a few hours of programming time to invest in my TREC experiment; contrariwise, I had a NeXT workstation with an optical disk and plenty of unused background CPU cycles available. This led me to try a quick-and-dirty approach using the regular expression pattern-matching and other programming facilities of Gawk, a free version of the Awk language. I decided to work on the document routing task using the full TREC data set.

I took the 50 TREC questions and manually constructed simple regular expressions ("regexps") for each of the key terms in them. Thus, for Topic 001, on pending antitrust cases, I had /ANTITRUST/, /CASE/, and /PEND/; for topic 002, acquisitions or mergers involving US and foreign companies, I came up with /ACQUISITION|BUYOUT|MERGER|TAKEOVER/, etc. For equivalent terms which were implicitly boolean-OR'd together, I wrote a single regexp with "|" joining the words. I spent approximately two minutes per TREC query writing these patterns, a total of about two hours, and used words contained in

the TREC topic statements plus a few obvious synonyms which occurred to me as I typed in the queries. I converted all characters in the TREC document set to upper-case before processing, so my regexps ignored case issues.

To handle the proximity (boolean-AND-like requirement) among separate terms, and to generate estimated relevances, I invented a very simple scoring system. Every time a line in a document matched one of my regexps, I added an arbitrary 5 points to that regexp's score. A line in the document got a score equal to the product of the regexp scores for that query. When moving on to the next line of a document, I multiplied all regexp scores by 0.9, to make them fade away with a characteristic length scale of 10 lines (= 1/(1-0.9)). The estimated relevance for a document as a whole was the maximum relevance of any line in that document.

For terms which were to be negatively-weighted (boolean-NOT-like), as in TREC Topic 026, instead of multiplying in the regexp's score to get a line's relevance, I multiplied in 5 minus that score. I experimented briefly with different weights and relevance-length-scales for different terms, but decided that there was neither sufficient time nor much benefit to be gained that way, and so I settled on the 5-points-per-instance, 0.9 degradation-rate standard.

## Implementation

I performed my TREC experiments over a period of a few weeks in background on a NeXT workstation (an old Cube), using the (rather slow) built-in writeable optical disk to hold data copied from the TREC CD-ROMs. On the average, each TREC query took about one-third of a second per document to execute. My implementation ran 10 queries at a time and generated a line of output for each document, listing the document ID followed by (the integer part of) its estimated relevance, in 10 columns. I then used additional UNIX shell scripts containing *colrm* and *sort* and *head* to tabulate the 200 highest-ranked documents for each topic. A final Gawk program converted my results into the standard TREC format. The total wall-clock computation time which I used was about 11 days for each CD-ROM of data. Late in the process of scoring documents, I discovered that a bug in my programs caused the last document in a data set not to be ranked — but I had no time to fix the error, and it probably did not affect my overall results significantly.

## Further Work

I believe that the proximity-correlation approach used in my TREC experiment has promise for other relevance-ranking tasks. Almost certainly, the use of a compiled language (and perhaps a simpler, faster pattern-matching facility) rather than Gawk would result in a speed increase of an order of magnitude or more. Much higher speeds should be achievable by inverted-index methods. User feedback might be valuable in modifying the default settings for term weights and relevance-lengths. A thesaurus option to automatically generate synonyms could help automate the query-creation process.

## References

[1] Mark Zimmermann. "The FreeText Project: Large-Scale Personal Information Retrieval", in Delany & Landrow, **TEXT-BASED COMPUTING IN THE HUMANITIES** (to be published by MIT Press, early 1993), pps. 51-66.

# TRECscore.Q.01-10.gawk



```
# score documents delimited by <DOCNO> lines from TREC dsts
# for first 10 questions on TREC list
# ^z 920527-29
#
#  usage:  gawk -f TRECscore.Q.1-10.gswk
#
# typicslly will want to do something like:
#  zcst wsj/19*/*.Z | tr s-z A-Z | gawk -f TRECscore.Q.1-10.gawk >Q.1-10.TRECscores.out
#
# reads from stdin and outputs scores for each document to stdout
# in formst:
# [<DOCNO>] [score1] [score2] ... [score10]

BEGIN {     m1 = 0;
            m2 = 0;
            m3 = 0;
            m4 = 0;
            m5 = 0;
            m6 = 0;
            m7 = 0;
            m8 = 0;
            m9 = 0;
            m10= 0;
            s1a = 0;
            s1b = 0;
            s1c = 0;
            s2a = 0;
            s2b = 0;
            s2c = 0;
            s3a = 0;
            s3b = 0;
            s3c = 0;
            s4a = 0;
            s4b = 0;
            s4c = 0;
            s4d = 0;
            s5a = 0;
            s5b = 0;
            s5c = 0;
            s6a = 0;
            s6b = 0;
            s6c = 0;
            s6d = 0;
            s7a = 0;
            s7b = 0;
            s7c = 0;
            s7d = 0;
            s8s = 0;
            s8b = 0;
            s8c = 0;
            s8d = 0;
            s9a = 0;
            s9b = 0;
            s9c = 0;
            s10a = 0;
            s10b = 0;
            docno = "<null>"; }

/<DOCNO>/ { printf ("%-20s %5d %5d %5d %5d %5d %5d %5d %5d %5d %5d\n",
                    docno, m1, m2, m3, m4, m5, m6, m7, m8, m9, m10);
            docno = $2;
            m1 = 0;
            m2 = 0;
            m3 = 0;
            m4 = 0;
            m5 = 0;
            m6 = 0;
            m7 = 0;
            m8 = 0;
            m9 = 0;
            m10= 0;
            s1a = 0;
            s1b = 0;
            s1c = 0;
            s2a = 0;
            s2b = 0;
            s2c = 0;
            s3a = 0;
            s3b = 0;
            s3c = 0;
            s4a = 0;
            s4b = 0;
            s4c = 0;
            s4d = 0;
            s5a = 0;
            s5b = 0;
            s5c = 0;
            s6a = 0;
            s6b = 0;
            s6c = 0;
            s6d = 0;
            s7a = 0;
            s7b = 0;
            s7c = 0;
            s7d = 0;
            s8a = 0;
            s8b = 0;
            s8c = 0;
            s8d = 0;
            s9a = 0;
            s9b = 0;
            s9c = 0;
            s10a = 0;
            s10b = 0; }

# topic 001 --- antitrust cases pending

/ANTITRUST/ { s1a += 5; }
/CASE/      { s1b += 5; }
/PEND/      { s1c += 5; }

 { s1 = s1s * s1b * s1c;
   if (s1 > m1) m1 = s1;
   s1a *= .9;
   s1b *= .9;
   s1c *= .9; }

# topic 002 --- acquisition/merger/etc. involving US & foreign (companies)

/ACQUISITION|BUYOUT|MERGER|TAKEOVER/ { s2s += 5; }
/US|U\.S\.|AMERICAN/                 { s2b += 5; }
/FOREIGN/                            { s2c += 5; }


 { s2 = s2s * s2b * s2c;
   if (s2 > m2) m2 = s2;
   s2s *= .9;
   s2b *= .9;
```

```
   s2c *= .9; }

# topic 003 --- Japanese joint venture

/JAPAN/    { s3a += 5; }
/JOINT/    { s3b += 5; }
/VENTURE/  { s3c += 5; }

 { s3 = s3a * s3b * s3c;
   if (s3 > m3) m3 = s3;
   s3a *= .9;
   s3b *= .9;
   s3c *= .9; }

# topic 004 --- debt rescheduling developing/third-world country/nation

/DEBT/                  { s4a += 5; }
/RESCHEDUL/             { s4b += 5; }
/COUNTR|NATION/         { s4c += 5; }
/DEVELOP|THIRD.WORLD/   { s4d += 5; }

 { s4 = s4a * s4b * s4c * s4d;
   if (s4 > m4) m4 = s4;
   s4a *= .9;
   s4b *= .9;
   s4c *= .9;
   s4d *= .9; }

# topic 005 --- dumping charges US/EC vs. Japan

/DUMPING/                                   { s5a += 5; }
/US|U\.S\.|EC|E\.C\.|EUROPEAN COMMUNITY/    { s5b += 5; }
/JAPAN/                                     { s5c += 5; }

 { s5 = s5a * s5b * s5c;
   if (s5 > m5) m5 = s5;
   s5a *= .9;
   s5b *= .9;
   s5c *= .9; }

# topic 006 --- third world (or developing nation) debt relief

/THIRD.WORLD|DEVELOP/           { s6a += 5; }
/COUNTR|NATION/                 { s6b += 5; }
/DEBT/                          { s6c += 5; }
/RELIEF|FORGIVE|RESCHEDUL/      { s6d += 5; }


 { s6 = s6a * s6b * s6c * s6d;
   if (s6 > m6) m6 = s6;
   s6a *= .9;
   s6b *= .9;
   s6c *= .9;
   s6d *= .9; }

# topic 007 --- US budget deficit decrease/reduction

/US|U\.S\.|FEDERAL|AMERICA/     { s7a += 5; }
/BUDGET/                        { s7b += 5; }
/DEFICIT|SHORTFALL|SPEND/       { s7c += 5; }
/REDUC|DECREAS|CUT|ELIMINAT/    { s7d += 5; }

 { s7 = s7a * s7b * s7c * s7d;
   if (s7 > m7) m7 = s7;
   s7a *= .9;
   s7b *= .9;
   s7c *= .9;
   s7d *= .9; }

# topic 008 --- non-US economic indicator/index projections/forecasts

/US|U\.S\.|AMERICA/     { s8a += 5; }
/ECONOM/                { s8b += 5; }
/INDICATOR|INDEX/       { s8c += 5; }
/PROJECT|FORECAST/      { s8d += 5; }


 { s8 = (5 - s8a) * s8b * s8c * s8d;
   if (s8 > m8) m8 = s8;
   s8a *= .9;
   s8b *= .9;
   s8c *= .9;
   s8d *= .9; }

# topic 009 --- 1988 presidential candidate sightings/locations (see name list)

/GEORGE.*BUSH|ROBERT.*DOLE|JACK.*KEMP|AL.*HAIG|PIERRE.*DUPONT|PAT.*ROBERTSON|MICHAEL.*DUK
AKIS|JESSE.*JACKSON|GARY.*HART|JO.*BIDEN|AL.*GORE|PAUL.*SIMON|RICHARD.*GEPHARDT|BRUCE.*BA
BBIT/              { s9a += 5; }
/ IN | AT | TO | NEAR /                                  { s9b += 5; }
/WASHINGTON|NEW YORK|LOS ANGELES|MOSCOW|LONDON|PARIS|TOKYO|WHITE HOUSE|CAMP DAVID|TEXAS|N
EW HAMPSHIRE|IOWA|EUROPE|CAPITOL HILL|CITY|STATE|TOWN|STREET|BUILDING/     { s9c += 5; }

 { s9 = s9a * s9b * s9c;
   if (s9 > m9) m9 = s9;
   s9a *= .9;
   s9b *= .9;
   s9c *= .9; }

# topic 010 --- AIDS treatment

/ AIDS/              { s10a += 5; }
/TREATMENT|DRUG/     { s10b += 5; }


 { s10 = s10a * s10b;
   if (s10 > m10) m10 = s10;
   s10a *= .9;
   s10b *= .9; }
```

# TRECscore.Q.11-20.gawk



```
# score documents delimited by <DOCNO> lines from TREC data
# for aecond 10 questions on TREC liat
# ^z 920527-29, 0601
#
#  usage:  gawk -f TRECscore.Q.11-20.gawk
#
# typically will want to do something like:
#   zcat wsj/19*/*.Z | tr a-z A-Z | gawk -f TRECscore.Q.11-20.gawk >Q.11-20.TRECscores.o
ut
#
# reads from stdin and outputs scores for each document to stdout
# in format:
# [<DOCNO>] [score1] [score2] ... [score10]

BEGIN {     m1 = 0;
            m2 = 0;
            m3 = 0;
            m4 = 0;
            m5 = 0;
            m6 = 0;
            m7 = 0;
            m8 = 0;
            m9 = 0;
            m10= 0;
            s1a = 0;
            s1b = 0;
            s1c = 0;
            s2a = 0;
            s2b = 0;
            s3a = 0;
            s3b = 0;
            s3c = 0;
            s4a = 0;
            s4b = 0;
            s4c = 0;
            a4d = 0;
            s5a = 0;
            s5b = 0;
            s5c = 0;
            s6a = 0;
            s6b = 0;
            s6c = 0;
            s7a = 0;
            s7b = 0;
            s7c = 0;
            s8a = 0;
            s8b = 0;
            s8c = 0;
            s9a = 0;
            s9b = 0;
            s9c = 0;
            s10a = 0;
            s10b = 0;
            s10c = 0;
            docno = "<null>"; }

/<DOCNO>/ { printf ("%-20s %5d %5d %5d %5d %5d %5d %5d %5d %5d %5d %5d\n",
                    docno, m1, m2, m3, m4, m5, m6, m7, m8, m9, m10);
            docno = $2;
            m1 = 0;
            m2 = 0;
            m3 = 0;
            m4 = 0;
            m5 = 0;
            m6 = 0;
            m7 = 0;
            m8 = 0;
            m9 = 0;
            m10= 0;
            s1a = 0;
            s1b = 0;
            s1c = 0;
            s2a = 0;
            s2b = 0;
            s3a = 0;
            s3b = 0;
            s3c = 0;
            s4a = 0;
            s4b = 0;
            s4c = 0;
            s4d = 0;
            s5a = 0;
            s5b = 0;
            s5c = 0;
            s6a = 0;
            s6b = 0;
            s6c = 0;
            s7a = 0;
            s7b = 0;
            s7c = 0;
            s8a = 0;
            s8b = 0;
            s8c = 0;
            s9a = 0;
            s9b = 0;
            s9c = 0;
            s10a = 0;
            s10b = 0;
            s10c = 0; }

# topic 011 --- space program

/SPACE/                 { s1a += 5; }
/PROGRAM|PROJECT/       { s1b += 5; }
/GOAL|PLAN/             { s1c += 5; }

 { s1 = s1a * s1b * s1c;
   if (s1 > m1) m1 = s1;
   s1a *= .9;
   s1b *= .9;
   s1c *= .9; }

# topic 012 --- water pollution

/WATER/ { a2a += 5; }
/POLLUTION/                     { s2b += 5; }


 { s2 = s2a * s2b;
   if (s2 > m2) m2 = s2;
   s2a *= .9;
   a2b *= .9; }

# topic 013 --- Mitaubishi Heavy Industries Ltd.

/MITSUBISHI/    { s3a += 5; }
/MEAVY/     { s3b += 5; }
/INDUSTR/ { s3c += 5; }
```

```
{ s3 = s3a * s3b * s3c;
  if (s3 > m3) m3 = s3;
  s3a *= .9;
  s3b *= .9;
  s3c *= .9; }

# topic 014 --- drug approval

/DRUG|MEDICINE/                { s4a += 5; }
/FINAL/                        { s4b += 5; }
/APPROVAL|MARKET CLEAR/        { s4c += 5; }
/NAME|GENERIC|BRAND/           { s4d += 5; }

{ s4 = s4a * s4b * s4c * s4d;
  if (s4 > m4) m4 = s4;
  s4a *= .9;
  s4b *= .9;
  s4c *= .9;
  s4d *= .9; }

# topic 015 --- CEO (new appointment or resignation)

/CEO|CHIEF EXECUTIVE OFFICER/              { s5a += 5; }
/COMPANY|CORPORATION|INCORPORATED|LTD|INC/  { s5b += 5; }
/APPOINT|RESIGN|CHOSE/                     { s5c += 5; }

{ s5 = s5a * s5b * s5c;
  if (s5 > m5) m5 = s5;
  s5a *= .9;
  s5b *= .9;
  s5c *= .9; }

# topic 016 --- marketing agrochemicals

/SALE|MARKET/                  { s6a += 5; }
/AGRICULTUR|CROP|AGRO/         { s6b += 5; }
/CHEMICAL|PESTICIDE|HERBICIDE|FUNGICIDE|INSECTICIDE|FERTILIZER/   { s6c += 5; }

{ s6 = s6a * s6b * s6c;
  if (s6 > m6) m6 = s6;
  s6a *= .9;
  s6b *= .9;
  s6c *= .9; }

# topic 017 --- measures to control agrochemicals

/BAN|REGULAT|CONTROL|RESTRICT|CURB/     { s7a += 5; }
/AGRICULTUR|CROP|AGRO/                  { s7b += 5; }
/CHEMICAL|PESTICIDE|HERBICIDE|FUNGICIDE|INSECTICIDE|FERTILIZER/       { s7c += 5; }

{ s7 = s7a * s7b * s7c;
  if (s7 > m7) m7 = s7;
  s7a *= .9;
  s7b *= .9;
  s7c *= .9; }

# topic 018 --- Japsnese stock msrket trends

/JAPAN|NIKKEI|TOKYO/           { s8s += 5; }
/STOCK|MARKET|AVERAGE/         { s8b += 5; }
/TREND|CHANGE|RISE|FALL/       { s8c += 5; }

{ s8 = s8a * s8b * s8c;
  if (s8 > m8) m8 = s8;
  s8a *= .9;
  s8b *= .9;
  s8c *= .9; }

# topic 019 --- global stock market trends

/STOCK/                        { s9a += 5; }
/MARKET/                       { s9b += 5; }
/TREND|CHANGE|RISE|FALL/       { s9c += 5; }

{ s9 = s9a * s9b * s9c;
  if (s9 > m9) m9 = s9;
  s9a *= .9;
  s9b *= .9;
  s9c *= .9; }

# topic 020 --- patent infringement lawsuits

/PATENT/                { s10a += 5; }
/INFRING/               { s10b += 5; }
/LAWSUIT|TRIAL|COURT/   { s10c += 5; }

{ s10 = s10a * s10b * s10c;
  if (s10 > m10) m10 = s10;
  s10a *= .9;
  s10b *= .9;
  s10c *= .9; }
```

# TRECscore.Q.21-30.gawk



```
# score documents delimited by <DOCNO> lines from TREC data
# for third 10 questions on TREC list
# ^z 920527-29, 0601, 0602
#
#  usage:  gawk -f TRECscore.Q.21-30.gawk
#
# typically will want to do something like:
#   zcat wsj/19*/*.Z | tr a-z A-Z | gawk -f TRECscore.Q.21-30.gawk >Q.21-30.TRECscores.o
ut
#       ((maybe prefixed by nohup))
# then typically will want to do something like:
# sort -n +1 TRECscores.out | tail -1000 >Q21.best
#
# this program reads from stdin and outputs scores for each document to stdout
# in format:
# [<DOCNO>] [score1] [score2] ... [score10]

BEGIN {     m1 = 0;
            m2 = 0;
            m3 = 0;
            m4 = 0;
            m5 = 0;
            m6 = 0;
            m7 = 0;
            m8 = 0;
            m9 = 0;
            m10= 0;
            s1a = 0;
            s1b = 0;
            s1c = 0;
            s2a = 0;
            s2o = 0;
            s3a = 0;
            s3b = 0;
            s3c = 0;
            s4a = 0;
            s4b = 0;
            s4c = 0;
            s5a = 0;
            s5b = 0;
            s5c = 0;
            s6a = 0;
            s6b = 0;
            s6c = 0;
            s7a = 0;
            s7b = 0;
            s8a = 0;
            s8b = 0;
            s8c = 0;
            s9a = 0;
            s9b = 0;
            s9c = 0;
            s9d = 0;
            s10a = 0;
            s10b = 0;
            docno = "<null>"; }

/<DOCNO>/ { printf ("%-20s %5d %5d %5d %5d %5d %5d %5d %5d %5d %5d %5d\n",\
                    docno, m1, m2, m3, m4, m5, m6, m7, m8, m9, m10);
            docno = $2;
            m1 = 0;
            m2 = 0;
            m3 = 0;
            m4 = 0;
            m5 = 0;
            m6 = 0;
            m7 = 0;
            m8 = 0;
            m9 = 0;
            m10= 0;
            s1a = 0;
            s1b = 0;
            s1c = 0;
            s2a = 0;
            s2b = 0;
            s3a = 0;
            s3b = 0;
            s3c = 0;
            s4a = 0;
            s4b = 0;
            s4c = 0;
            s5a = 0;
            s5b = 0;
            s5c = 0;
            s6a = 0;
            s6b = 0;
            s6c = 0;
            s7a = 0;
            s7b = 0;
            s8a = 0;
            s8b = 0;
            s8c = 0;
            s9a = 0;
            s9b = 0;
            s9c = 0;
            s9d = 0;
            s10a = 0;
            s10b = 0; }

# topic 021 --- superconductivity breakthrough with commercial application

/SUPERCONDUCT/                  { s1a += 5; }
/DISCOVER|BREAKTHR|FIRST|ADVANC/      { s1b += 5; }
/COMMERC|APPLI|PRACTIC/             { s1c += 5; }

 { s1 = s1a * s1b * s1c;
   if (s1 > m1) m1 = s1;
   s1a *= .9;
   s1b *= .9;
   s1c *= .9; }

# topic 022 --- counternarcotics

/DRUG|NARCOTIC|COCAINE|HEROIN|OPIUM|MARIJUANA/ { s2a += 5; }
/ILLEGAL|SMUGGL|CARTEL|TRAFFIC/                { s2b += 5; }


 { s2 = s2a * s2b;
   if (s2 > m2) m2 = s2;
   s2a *= .9;
   s2b *= .9; }

# topic 023 --- .legal repercussions of agrochemical use

/JUDGMENT|LAW|LEGAL|CASE|SUIT/   { s3a += 5; }
/PESTICIDE|HERBICIDE|FUNGICIDE|INSECTICIDE|FERTILIZER/   { s3b += 5; }
/MISUSE|VICTIM|ACCIDENT|TRAGEDY|TRAGIC|DISASTER/ { s3c += 5; }
```

```
 { s3 = s3a * s3b * s3c;
   if (s3 > m3) m3 = s3;
   s3a *= .9;
   s3b *= .9;
   s3c *= .9; }

# topic 024 --- new medical technology

/DRUG|MEDIC/                              { s4a += 5; }
/COMPANY|HOSPITAL|RESEARCH|INSTITUT/      { s4b += 5; }
/DRUG|EQUIPMENT|TREATMENT|PROCEDURE/      { s4c += 5; }

 { s4 = s4a * s4b * s4c ;
   if (s4 > m4) m4 = s4;
   s4a *= .9;
   s4b *= .9;
   s4c *= .9; }

# topic 025 --- aftermath of Chernobyl, European gov't actions

/CHERNOBYL/                                { s5a += 5; }
/EUROPE|BULGARIA|RUMANIA|ROMANIA|BELGI|NETHERLANDS|HOLLAND|FRANCE|FRENCH|GERMAN|SCANDINA
VIA|U\.K\.| UK |BRITAIN|ENGLAND|UNITED KINGDOM|NORWAY|NORWEGIAN|SWEDE|FINLAND|FINNISH|PO
LAND|POLISH|HUNGAR|CZECHOSLOVAK|AUSTRIA|SWITZERLAND|SWISS|GREEK|GREECE|YUGOSLAV|ITALIAN|
ITALY|SPAIN|SPANISH|PORTUG/ { s5b += 5; }
/FOOD|TESTING|FALLOUT|CONSEQUENCE|EVACUAT|HEALTH|CANCER/      { s5c += 5; }

 { s5 = s5a * s5b * s5c;
   if (s5 > m5) m5 = s5;
   s5a *= .9;
   s5b *= .9;
   s5c *= .9; }

# topic 026 --- tracking influential players in multimedia

/CD-ROM|MULTIMEDIA|MULTI-MEDIA/                  { s6a += 5; }
/APPLICATION|DEVELOPER/                          { s6b += 5; }
/APPLE|IBM|MICROSOFT/                            { s6c += 5; }

 { s6 = s6a * s6b * (5 - s6c);
   if (s6 > m6) m6 = s6;
   s6a *= .9;
   s6b *= .9;
   s6c *= .9; }

# topic 027 --- expert systems or neural networks in business or manufacturing

/EXPERT.SYSTEM|RULE.BASED|KNOWLEDGE.BASE|ARTIFICIAL INTELLIGEN|NEURAL NET/ { s7a += 5; }
/BUSINESS|MANUFACTUR/                                                      { s7b += 5; }

 { s7 = s7a * s7b;
   if (s7 > m7) m7 = s7;
   s7a *= .9;
   s7b *= .9; }

# topic 028 --- AT&T's technical efforts
/AT\&T|AMERICAN TELEPHONE|BELL SYSTEM|BELL LAB/      { s8a += 5; }
/PRODUCT|TECHNOLOG|COMPUTER|COMMUNICATION/           { s8b += 5; }
/BABY BELL|BELLCORE|UNIX/                            { s8c += 5; }

 { s8 = s8a * s8b * (5 - s8c);
   if (s8 > m8) m8 = s8;
   s8a *= .9;
   s8b *= .9;
   s8c *= .9; }

# topic 029 --- foreign installation of AT&T communications products

/FOREIGN|NATION|COUNTRY/                          { s9a += 5; }
/AT\&T|AMERICAN TELEPHONE| BELL/                  { s9b += 5; }
/TECHNOLOG|SWITCHING|FIBER|NETWORK/               { s9c += 5; }
/U\.K\.|UNITED KINGDOM|ENGLAND|BRITAIN|CANADA/    { s9d += 5; }

 { s9 = s9a * s9b * s9c * (5 - s9d) ;
   if (s9 > m9) m9 = s9;
   s9a *= .9;
   s9b *= .9;
   s9c *= .9;
   s9d *= .9; }

# topic 030 --- OS/2 Problems

/OS\/2/                                  { s10a += 5; }
/PROBLEM|TROUBLE|DELAY|IMMATUR/          { s10b += 5; }

 { s10 = s10a * s10b ;
   if (s10 > m10) m10 = s10;
   s10a *= .9;
   s10b *= .9; }
```

# TRECscore.Q.31-40.gawk



```
# score documents delimited by <DOCNO> lines from TREC data
# for fourth 10 questions on TREC list
# ^z 920527-29, 0601, 0602, 0603
#
#  usage:  awk -f TRECscore.Q.31-40.awk
#
# typically will want to do something like:
#   zcat wsj/19*/*.Z | tr a-z A-Z | awk -f TRECscore.Q.31-40.awk >Q.31-40.TRECscores.out
#       (maybe prefixed by nohup))
# then typically will want to do something like:
# sort -n +1 TRECscores.out | tail -1000 >Q31.best
# etc....
#
# this program reads from stdin and outputs scores for each document to stdout
# in format:
# [<DOCNO>] [score1] [score2] ... [score10]

BEGIN {     m1 = 0;
            m2 = 0;
            m3 = 0;
            m4 = 0;
            m5 = 0;
            m6 = 0;
            m7 = 0;
            m8 = 0;
            m9 = 0;
            m10= 0;
            s1a = 0;
            s1b = 0;
            s1c = 0;
            s2a = 0;
            s2b = 0;
            s3a = 0;
            s3b = 0;
            s3c = 0;
            s4a = 0;
            s4b = 0;
            s5a = 0;
            s5b = 0;
            s5c = 0;
            s5d = 0;
            s6a = 0;
            s6b = 0;
            s6c = 0;
            s7a = 0;
            s7b = 0;
            s8a = 0;
            s8b = 0;
            s8c = 0;
            s9a = 0;
            s9b = 0;
            s10a = 0;
            s10b = 0;
            docno = "<null>"; }

/<DOCNO>/ { printf ("%-20s %5d %5d %5d %5d %5d %5d %5d %5d %5d %5d %5d\n",\
                    docno, m1, m2, m3, m4, m5, m6, m7, m8, m9, m10);
            docno = $2;
            m1 = 0;
            m2 = 0;
            m3 = 0;
            m4 = 0;
            m5 = 0;
            m6 = 0;
            m7 = 0;
            m8 = 0;
            m9 = 0;
            m10= 0;
            s1a = 0;
            s1b = 0;
            s1c = 0;
            s2a = 0;
            s2b = 0;
            s3a = 0;
            s3b = 0;
            s3c = 0;
            s4a = 0;
            s4b = 0;
            s5a = 0;
            s5b = 0;
            s5c = 0;
            s5d = 0;
            s6a = 0;
            s6b = 0;
            s6c = 0;
            s7a = 0;
            s7b = 0;
            s8a = 0;
            s8b = 0;
            s8c = 0;
            s9a = 0;
            s9b = 0;
            s10a = 0;
            s10b = 0; }

# topic 031 --- advantages of OS/2

/OS\/2/                 { s1a += 5; }
/ADVANTAG|STRENGTH/       { s1b += 5; }
/WINDOWS|X.WINDOWS|DOS/             { s1c += 5; }

 { s1 = s1a * s1b * s1c;
   if (s1 > m1) m1 = s1;
   s1a *= .9;
   s1b *= .9;
   s1c *= .9; }

# topic 032 --- outsourcing computer work

/CONTRACT.*OUT|OUTSOURCING/             { s2a += 5; }
/COMPUTER|DATA|NETWORK/                 { s2b += 5; }


 { s2 = s2a * s2b;
   if (s2 > m2) m2 = s2;
   s2a *= .9;
   s2b *= .9; }

# topic 033 --- companies capable of producing document management systems

/DOCUMENT/    { s3a += 5; }
/MANAGEMENT|PROCESSING|AUTOMATION|OCR|OPTICAL CHARACTER RECOGNI/    { s3b += 5; }
/COMPANY|CORP|CO\.|INC\.|LTD\.|INCORPORATED/ { s3c += 5; }

 { s3 = s3a * s3b * s3c;
   if (s3 > m3) m3 = s3;
   s3a *= .9;
   s3b *= .9;
```

```
   s3c *= .9; }

# topic 034 --- ISDN applications/exploitation

/ISDN|INTEGRATED SERVICES DIGITAL NETWORK/     { s4a += 5; }
/STRATEGY|APPLICATION|PRODUCT/  { s4b += 5; }

 { s4 = s4a * s4b;
   if (s4 > m4) m4 = s4;
   s4a *= .9;
   s4b *= .9; }

# topic 035 --- alternatives to Postscript
# ((note "or" TRUETYPE implicit in ranking there --- is this cheating??))

/POSTSCRIPT/   { s5a += 5; }
/ADOBE|APPLE|MICROSOFT/ { s5b += 5; }
/ALTERNATIVE|SUBSTITUTE|COMPETIT/      { s5c += 5; }
/TRUETYPE/  { s5d += 5}

 { s5 = s5a * s5b * s5c + s5d;
   if (s5 > m5) m5 = s5;
   s5a *= .9;
   s5b *= .9;
   s5c *= .9;
   s5d *= .9;}

# topic 036 --- how rewriteable optical disks work

/OPTICAL/                   { s6a += 5; }
/DISK/                      { s6b += 5; }
/REWRIT/                    { s6c += 5; }

 { s6 = s6a * s6b * s6c;
   if (s6 > m6) m6 = s6;
   s6a *= .9;
   s6b *= .9;
   s6c *= .9; }

# topic 037 --- SAA components

/SAA|SYSTEMS APPLICATION ARCHITECTURE/                    { s7a += 5; }
/OFFICEVISION|COMPONENT|COMPLIANT|ADHERE|CONFORM|COMPLY/   { s7b += 5; }

 { s7 = s7a * s7b;
   if (s7 > m7) m7 = s7;
   s7a *= .9;
   s7b *= .9; }

# topic 038 --- role of minicomputers/mainframes in LAN/PC/workstation environments

/MINICOMPUTER|VAX|MAINFRAME/            { s8a += 5; }
/ LAN| PC/                              { s8b += 5; }
/ ROLE|TRANSITION|CHANGE|ENVIRONMENT/   { s8c += 5; }

 { s8 = s8a * s8b * s8c;
   if (s8 > m8) m8 = s8;
   s8a *= .9;
   s8b *= .9;
   s8c *= .9; }

# topic 039 --- client-server plans/expectations

/CLIENT.*SERVER/                                  { s9a += 5; }
/IMPLEMENT|BUILD|PLAN|EXPECTATION/                { s9b += 5; }

 { s9 = s9a * s9b ;
   if (s9 > m9) m9 = s9;
   s9a *= .9;
   s9b *= .9; }

# topic 040 --- impact of info systems tech on orgs

/ORGANIZATION|CORPORATION|COMPANY|EFFICIEN|PRODUCTIVITY|INNOVATION|COLLABORAT/ { s10a +=
5; }
/GRAPHICAL USER INTERFACE| GUI |LOCAL AREA NETWORK| LAN |FACSIMILE| FAX| E.MAIL| EMAIL|SP
|READSHEET|DATABASE|DESKTOP PUBLISH|WORKGROUP/ { s10b += 5; }


 { s10 = s10a * s10b ;
   if (s10 > m10) m10 = s10;
   s10a *= .9;
   s10b *= .9; }
```

# TRECscore.Q.41-50.gawk

```
score documents delimited by <DOCNO> lines from TREC data
for fifth 10 questions on TREC list
^z 920527-29, 0601, 0602, 0603, 0604

 usage:  awk -f TRECscore.Q.41-50.awk

typically will want to do something like:
  zcat wsj/19*/*.Z | tr a-z A-Z | awk -f TRECscore.Q.n1-n2.awk >Q.n1-4n2.TRECscores.ou

     ((maybe prefixed by nohup))
then typically will want to do something like:
  sort -n +1 TRECscores.out | tail -1000 >Qn1.best
etc....

this program reads from stdin and outputs scores for each document to stdout
in format:
[<DOCNO>] [score1] [score2] ... [score10]

BEGIN {     m1 = 0;
            m2 = 0;
            m3 = 0;
            m4 = 0;
            m5 = 0;
            m6 = 0;
            m7 = 0;
            m8 = 0;
            m9 = 0;
            m10= 0;
            s1a = 0;
            s1b = 0;
            s1c = 0;
            s2a = 0;
            s2b = 0;
            s3a = 0;
            s3b = 0;
            s3c = 0;
            s3d = 0;
            s4a = 0;
            s4b = 0;
            s4c = 0;
            s5a = 0;
            s5b = 0;
            s6a = 0;
            s6b = 0;
            s6c = 0;
            s7a = 0;
            s7b = 0;
            s7c = 0;
            s8a = 0;
            s8b = 0;
            s9a = 0;
            s9b = 0;
            s10a = 0;
            s10b = 0;
            docno = "<null>"; }

<DOCNO>/ { printf ("%-20s %5d %5d %5d %5d %5d %5d %5d %5d %5d %5d %5d\n",\
                    docno, m1, m2, m3, m4, m5, m6, m7, m8, m9, m10);
           docno = $2;
           m1 = 0;
           m2 = 0;
           m3 = 0;
           m4 = 0;
           m5 = 0;
           m6 = 0;
           m7 = 0;
           m8 = 0;
           m9 = 0;
           m10= 0;
           s1a = 0;
           s1b = 0;
           s1c = 0;
           s2a = 0;
           s2b = 0;
           s3a = 0;
           s3b = 0;
           s3c = 0;
           s3d = 0;
           s4a = 0;
           s4b = 0;
           s4c = 0;
           s5a = 0;
           s5b = 0;
           s6a = 0;
           s6b = 0;
           s6c = 0;
           s7a = 0;
           s7b = 0;
           s7c = 0;
           s8a = 0;
           s8b = 0;
           s9a = 0;
           s9b = 0;
           s10a = 0;
           s10b = 0; }

# topic 041 --- computer or communicstions system upgrade

/COMPUTER|COMMUNICATION.*SYSTEM/      { s1a += 5; }
/UPGRADE|TRANSITION|PHAS.*OUT/        { s1b += 5; }
/COMPANY|CORP|CO\.|INC\.|LTD\.|INCORPORATED/ { s1c += 5; }

 { s1 = s1a * s1b * s1c;
   if (s1 > m1) m1 = s1;
   s1a *= .9;
   s1b *= .9;
   s1c *= .9; }

# topic 042 --- end user computing

/ENDUSER|END.USER|DECENTRALIZ| EUC /              { s2a += 5; }
/COMPUT/                                          { s2b += 5; }


 { s2 = s2a * s2b;
   if (s2 > m2) m2 = s2;
   s2a *= .9;
   s2b *= .9; }

# topic 043 --- AI conferences between 1 Feb and 20 Mar 91

/ AI |ARTIFICIAL INTELLIGENCE/    { s3a += 5; }
/CONFERENCE|SYMPOSIUM/            { s3b += 5; }
/1991/                            { s3c += 5; }
/FEBRUARY| FEB| MARCH| MAR/       { s3d += 5; }

 { s3 = s3a * s3b * s3c * s3d;
   if (s3 > m3) m3 = s3;
```

# TRECscore.Q.41-50.gawk

```
  s3a *= .9;
  s3b *= .9;
  s3c *= .9;
  s3d *= .9; }

# topic 044 --- staff reductions at computer/communications companies

/STAFF|WORKERS/     { s4a += 5; }
/REDUCTION|LAYOFF|CUTBACK|ATTRITION|FIRING|FIRED/  { s4b += 5; }
/COMPUTER|COMMUNICATION|TELEPHONE/  { s4c += 5; }

{ s4 = s4a * s4b * s4c;
  if (s4 > m4) m4 = s4;
  s4a *= .9;
  s4b *= .9;
  s4c *= .9; }

# topic 045 --- CASE success/failure

/CASE |COMPUTER.*AIDED.*SOFTWARE.*ENGINEERING|\(CASE\)/    { s5a += 5; }
/SUCCESS|FAIL|SUCCEED|PRODUCTIVITY|IMPROVEMENT/ { s5b += 5; }

{ s5 = s5a * s5b;
  if (s5 > m5) m5 = s5;
  s5a *= .9;
  s5b *= .9; }

# topic 046 --- computer virus outbreak

/COMPUTER|INTERNET/                  { s6a += 5; }
/VIRUS|WORM/                       { s6b += 5; }
/OUTBREAK|ORGANIZATION|VICTIM|COMPANY|CORP|CO\.|INC\.|LTD\.|INCORPORATED/ { s6c += 5; }

{ s6 = s6a * s6b * s6c;
  if (s6 > m6) m6 = s6;
  s6a *= .9;
  s6b *= .9;
  s6c *= .9; }

# topic 047 --- computer system contracts over 1 M$

/COMPUTER|COMMUNICATION|INFORMATION/                    { s7a += 5; }
/CONTRACT|ACQUI|PROCURE|AWARD|PURCHASE/                { s7b += 5; }
/MILLION|000\,000|BILLION/                             { s7c += 5; }

{ s7 = s7a * s7b * s7c;
  if (s7 > m7) m7 = s7;
  s7a *= .9;
  s7b *= .9;
  s7c *=  9;}

# topic 048 --- purchase of modern communications equipment

/PBX| LAN |LOCAL AREA NET|NETWORK MANAGEMENT| FAX/                  { s8a += 5; }
/PURCHASE|ACQUI|PROCURE|AWARD|CONTACT/                             { s8b += 5; }

{ s8 = s8a * s8b;
  if (s8 > m8) m8 = s8;
  s8a *= .9;
  s8b *= .9; }

# topic 049 --- supercomputer operation/programming/purchase

/SUPERCOMPUTER|CRAY|IBM.*3090/                  { s9a += 5; }
/PURCHASE|ACQUI|PROCURE|AWARD|CONTACT|RESEARCH|OPERAT|PROGRAM|CENTER/    { s9b += 5; }

{ s9 = s9a * s9b ;
  if (s9 > m9) m9 = s9;
  s9a *= .9;
  s9b *= .9; }

# topic 050 --- military interest in Virtual Reality

/VIRTUAL REALITY| VR |\(VR\)|CYBERSPACE/ { s10a += 5; }
/MILITARY|ARMY|ARPA|NAVY|AIR FORCE|MARINE| DOD |DEFENSE|DEFENCE/ { s10b += 5; }

{ s10 = s10a * s10b ;
  if (s10 > m10) m10 = s10;
  s10a *= .9;
  s10b *= .9; }
```

# Workshop on:
# Use of Natural Language Processing at TREC

*[summary by: David Lewis and Alan Smeaton]*

A working group of individuals concerned with the impact and use of natural language processing (NLP) techniques in TREC-1 met in two sessions for lively discussion. The first question addressed was what exactly it meant to say that an information retrieval (IR) system uses NLP. There were some interesting disagreements about which TREC systems could be considered NLP-based. It was agreed there were a number of levels of NLP techniques that might be used in IR systems, including traditional distinctions between lexical, morphological, syntactic, and other levels. It was also agreed that there are many "boxes" or linguistic processes which could be useful for information retrieval including lexicons and gazetteers, syntactic analyzers, knowledge bases, etc. As such boxes become more robust and widely available, one or more may be plugged into IR systems with varying degrees of attention paid to linguistic issues, making the dividing line between NLP and non-NLP IR systems increasingly fuzzy.

The second issue discussed was whether NLP techniques had had an impact on retrieval quality in the TREC-1 experiments. The consensus was that it was impossible to tell from the results presented. Groups did not for the most part present results on controlled comparisons between using and not using their NLP components. Furthermore, the significant number of alternative retrieval models (vector, probabilistic, connectionist, and other) precluded making NLP vs. non-NLP comparisons across groups.

The discussion turned to the challenges of experimentation with NLP in the context of TREC-1, revealing ample reasons that TREC-1 participants should not be faulted for the lack of controlled experiments. All groups found getting their system to operate on 2 gigabytes of text very challenging. The tight schedule and limited funding available led to a "one mistake syndrome"---the limited amount of time available allowed groups to make only one mistake if they were to get their results in on time. Groups using NLP techniques were doubly challenged, given that the generally higher computational costs of these methods compared to traditional word-based indexing.

It was noted that, with the exception of the University of Massachusetts TIPSTER group, there was little work presented on focusing NLP on queries rather than documents. This is an area of research where the computational demands are much less than in applying NLP to document texts, and richness and subtlety of the TREC/TIPSTER topic descriptions would suggests that such an approach could have significant payoffs. It was agreed that this was an important area of exploration for TREC-2.

The issue of sharing of resources among TREC participants was discussed and the consensus is that although NLP work is generically similar, there do exist fine differences in NLP approaches which makes sharing of resources like dictionaries and parsers, troublesome at least. Still, everyone agreed that sharing is desirable when the technological and legal (copyright, etc.) barriers can be overcome.

Another interesting topic raised was the issue of how the effectiveness of NLP methods in IR changes when one scales up from traditional small IR test collections to gigabyte scale databases. Once again, the conclusion reached was that the TREC-1 results simply do not tell us this. We were unable to distinguish the "large collection factor" from other factors like long vs. short queries, long vs short documents and different query types, as well as other factors discussed above.

A tentative hypothesis put forward by some was that NLP-based methods may be best for paragraph or sub-document retrieval (termed "nugget extraction" during discussions) and that more traditional methods may be better for more general types of queries. It was suggested that testing this hypothesis, and in general getting a real understanding of the effect of NLP techniques on IR, would require a more careful analysis of the kinds of queries used (suggestions were made about how the query set might be improved or augmented), as well as details of how relevance judgments are made and what parts of documents are relevant.

In conclusion, it was acknowledged that the emphasis of researchers in TREC-1 had quite reasonably been simply on getting their systems to work at all with such a large collection of text. It was hoped that for TREC-2 more controlled comparisons and detailed analyses of failures and successes could be done, to give us more insight into the strengths and weaknesses of NLP methods in IR.

# Workshop on:
# Automatically Generating Adhoc and Routing Queries

*[summary by: Susan T. Dumais, Bellcore, std@bellcore.com]*

About 20 people attended the two workshops on automatic query generation. Many different issues were addressed, and I've tried to organize the important points under a few general headings.

*Topic Statements:*
We spent some time initially talking about how the topics statements were developed, what retrieval scenarios they are representative of, and some consequences of this for research. The topic statements are much more detailed, structured, and specific than queries associated with most previous IR test collections, averaging about 150 words in length. Most topics (routing topics 001-025 and adhoc topics 051-100) require that fairly specific facts be retrieved. Routing topics 026-050 are more general. The topic statements were generated by subject domain experts and reformulated using search results from two different retrieval systems. While this might be characteristic of routing applications or of dedicated searchers, there was some question about how likely more casual users would be to generate such queries. There was some interest in developing a companion set of shorter topic descriptions that could be used to better explore the effects of term expansion, feedback, and iterative query formulation. In contrast, there was also some interest in having expert human searchers carry out much deeper searches for a few topics in order to cast a wider net and increase the variety of documents retrieved.

*Term Extraction:*
Most of the fields in the topic description were used, and there was some evidence that the <concept> field was the most useful. Almost all systems used a stop-list and some kind of stemmer. A few systems recognized and tagged common abbreviations or acronyms, proper names, company names, place names, etc. Everyone agreed that a compendium of this information would be a valuable common resource. Many systems used differential term weighting. Typically weights derived from a statistical analyses of the documents were also used to weight query terms. Term weights sometimes depended on the topic field or syntactic slot the term occupied.

About half of the systems used phrases in addition to single words. Phrases were usually derived by simple statistical means using word adjacency (or co-occurrence withing k positions), with high thresholds on overall frequency of occurrence to limit the number of phrases. Some systems used syntactic analysis to discover phrases, but most of these groups did not automatically generate their queries. Phrases appeared to improve performance somewhat by increasing both precision and (somewhat unexpectedly) recall.

*Term expansion:*
Term expansion has long been used to increase recall by making the search query more comprehensive. Not all relations are equally useful in expansion, and the most commonly used relation was synonymy. Queries were expanded using several different sources of information - a thesaurus to generate semantic categories; a general, manually-constructed lexical system (wordnet); associations automatically derived from an analysis of word usage in the documents or

smaller syntactic units; and automatic pronoun disambiguation.

Relevance feedback is closely related to term expansion. It is not fully automatic in the sense that human judgements about the relevance of some small number of documents are required. However, the routing queries were specifically designed to take advantage of relevance judgements from a training corpus. More importantly, many of the same issues that arise in term expansion also occur in the context of relevance feedback. The most common implementation of relevance feedback was to modify the query by adding some words from relevant documents. For the TREC experiments as few as 5 words and as many as 250 words were added, with most systems adding from 10-30 words. Some systems also modified term weights, used information about words in non-relevant documents, and gave less weight to added words (compared with words in the original query).

There were few comparisons of term expansion (or feedback) compared to no expansion in the same system. Feedback improvements were somewhat smaller than expected based on experiments with smaller test collections. It is too early to tell for sure, but part of this may simply be that the original queries were very good.

The single common theme in the discussion of query expansion was *be careful*! Results were quite variable - appropriate term expansion can improve recall, but inappropriate expansion can just as easily harm performance. One major problem is that expansion is not easily limited to the intended meaning of a word. Some groups first disambiguated the word sense by hand before automatic expansion; others used automatic heuristics for disambiguation with some success. Other methods discussed to help limit undesirable associations included: expanding only "hot spots"; matching on smaller subtexts; giving less weight to added words relative to original query words; limiting the total number of words added; limiting the syntactic or semantic relations of added words; and limiting the influence that any single word can have in overall similarity.

*Miscellaneous observations:*
Few systems did anything more than extract single words and phrases. A few systems removed negated words (often by hand), and a few systems automatically generated Boolean queries.

Some groups used what might be called a "two-pass method", first using a standard global match to obtain a smaller group of documents which then receive more detailed processing. Some of the more detailed processing involved breaking the query down into smaller sub-units for matching.

*Summary:*
There were few really novel methods used for automatically generating either adhoc or routing queries. There are now some general and fairly comprehensive lexical resources that might be useful. The problems with over-expanding queries were quite noticeable in the TREC application. Systems that automatically generated queries often performed quite well compared to other systems. However, there were few direct comparisons of manual vs. automatic query generation, or of individual components (term expansion vs no expansion) within a system, and this is what is needed to understand the usefulness of such methods. Hopefully this will happen in TREC-2.

# Workshop on:
# Machine Learning and Relevance Feedback

*[summary by: Norbert Fuhr and Stephen Robertson]*

Note: This report covers the two separate workshop sessions on the topic.

A machine might use a number of different sources of information from which to learn. One major source, but not the only one, is relevance feedback information. Others include for example the user's selection of terms or information about the user's context or background.

One can also distinguish possible objects of a machine learning process -- what it might learn about. First, the object may be the current topic or query only -- this is the usual domain of relevance feedback. Second, the machine may learn about other topics or queries. This clearly requires a degree of abstraction. For example, the behaviour of a specific term in respect of one query is unlikely to tell us much about its behaviour in relation to a different query; however, it may tell us something about the behaviour of other terms sharing certain characteristics with it.

Somewhere in between these two kinds of object is a third: the machine may learn about this particular user, in a way that is applicable to his or her subsequent queries or uses of the system. It is not possible to study this kind of machine learning within the framework of the TREC experiment, because of the way in which topics and relevance judgments have been obtained. However, there is some work on such systems.

We should also ask the purpose of machine learning. In the context of TREC, this should presumably be the improvement of retrieval performance.

It is arguable whether some of the processes included in the above discussion actually qualify as Machine Learning in the sense in which the phrase is usually used: for example, a form of "learning" which affects only the next iteration of a single search hardly counts as ML. However, in an IR context there appears to be some continuity between such systems and those which come closer to "proper" ML.

One area of concern is the extraction of features. If there is a candidate set of features, then relevance feedback can contribute to their selection, but the original identification of features in text is a much more difficult task.

Another point worth noting for TREC is the desirability of considering learning methods in the context of interactive systems. The present TREC experimental design is not suited for evaluating interactive systems; it would be of great value in this context if the experimental design could be adapted to allow such evaluation.

For the areas where machine learning (ML) methods can be applied within IR, one may distinguish between models and representations. For retrieval, first representations of documents and queries (e.g. both as sets of terms) have to be formed. Based on these representations, models are applied in order to compute the retrieval status value for query-document pairs. In order to improve the representations, learning methods may be applied e.g. for the detection of phrases or for word sense disambiguation. Examples of learning methods used within models are regression methods, classification trees or genetic algorithms.

The features which ML methods are based upon may be derived from the text, but also from additional attributes (e.g. the source of a document may already indicate that it is not relevant for certain queries). Most ML methods assume that the features to be considered form a set, whereas tree- or graph-like structures have to be mapped onto this flat structure

first. The level of abstraction used in the definition of the features plays an important role for the applicability or non-applicability of certain learning methods, since most methods require a certain amount of data. In general, a higher level of abstraction yields more leaning data; on the other hand, the decision resulting from the learning algorithm may become too unspecific. For this reason, there is a need for different levels of abstraction, from which one may choose the most appropriate one for the actual circumstances. However, for text there are effectively only two levels of abstraction, namely either the term itself or its statistical parameters (e.g. within-document-frequency and inverse document frequency). A possible intermediate level would be sets of synonym terms.

In order to improve the effectiveness of ML methods for a given learning sample, prior knowledge plays an important role. For example, we may assume that the weight of a term with respect to a document is a monotonic function of its within-document-frequency. For this reason, a regression method which is implicitly based on this type of prior knowledge is more appropriate than e.g. a classification tree which makes no such assumption.

We may distinguish different sources of learning data. Most important, we have relevance feedback data. Here one may think of different levels of response, either by using multivalued relevance scales or by indicating important paragraphs within a long document. For some applications, it may also be necessary to get more specific feedback with respect to internal decisions of the system. For example, for tuning a phrase detection algorithm, it would be useful to get decisions about each specific phrase. As a third possible source of learning data, the combination of different sources of knowledge (e.g. thesauri and corpora) also might yield new information.

For the TREC initiative, there are two possible improvements which would ease the further application of ML methods. First, relevance feedback data should be enriched by indicating the most important paragraph of the document. As a precondition, there should some method for identifying single paragraphs, e.g. by additional SGML-like tags. Since the assessors will not give the requested type of judgements, the TREC participants would have to do this job, and NIST should act as collector and distributor for these judgements.

# Workshop on: Evaluation Issues

Evaluation has two distinct but complementary purposes. The first purpose of evaluation is to allow researchers and system developers to understand how their system is working, hopefully to improve its performance. The second purpose is to allow groups outside a system to understand its strengths and weaknesses and to do cross-system comparison. TREC has a need for both types of evaluation and the workshop discussed the general issue of evaluation and specifically how to improve evaluation in TREC-2.

The evaluation measures used in TREC-1 were standard recall and precision measures for each set of system results, plus a listing for each topic showing the best, worst, and median performance on that topic. These listings were helpful to TREC groups in finding which topics were easiest (or hardest) for their systems to handle. The standard recall and precision measures allowed some cross-system comparison, although any absolute ranking of systems is impossible because of the different levels of user and/or developer effort that must be part of a full evaluation.

Several improvements were suggested for TREC-2 (all of which will be implemented).

1. More documents need to be submitted for use in evaluation. The artificially low cutoff of 200 documents meant that the recall/precision figures were accurate only to about 40% recall, as all documents retrieved after rank 200 were marked non-relevant for evaluation purposes. Whereas all evaluation results were equally hurt by this, a higher cutoff (500 or 1000 documents) would allow better performance measurement accuracy.
2. Better methods are needed to deal with systems working on a variable thresholding method, where a threshold is set for each topic and far fewer than 200 documents could be submitted for evaluation for some topics. This particularly applies to routing.
3. Clearer definitions of automatic, manual, and feedback methods for constructing queries are needed.

Additionally several suggestions were made that are harder to implement (or possibly even to define), but would be helpful to researchers and system developers.

1. Provide two levels of relevance judgments, with the levels being "on topic" and "meets all criteria". The effects of the complex narrative in the topic could then be more easily separated from searching effects. However this might cause more subjective relevance assessment and create too many relevant documents for some topics.
2. Find some method of marking relevant paragraphs or sentences. This would be particularly useful for machine learning algorithms including relevance feedback.

Finally workshop participants had several suggestions to increase the general "learning level" of the conference.

1. Strongly urge all participants to obtain the evaluation programs from Chris Buckley to allow more internal evaluation.
2. Strongly urge all participants to attempt to modularize their various techniques to separate the effects of each technique. For example, it was suggested that all routing results using any type of training also discuss their baseline performance without training. The internal use of the TREC evaluation program could be done for this.

# APPENDIX A

This appendix contains tables of results for all the TREC-1 participants except the TIPSTER panel, whose tables appear in Appendix B. The tables in Appendix A and Appendix B show various measures of the performance on the adhoc and routing tasks. The adhoc results come first, followed by the routing results, with the tables in the same order as the presentation order of the papers. The definitions of the evaluation measures are given in the Overview, section 4, and readers unfamiliar with these measures should read that section first.

Care should be taken in comparing the tables across systems. These measures show performance only, with no measure of user or system effort. Additionally, because of misunderstandings about the query categories, some results may reflect manual adjustments of the queries even though the results are not formally categorized as feedback results.

The tables contain four major boxes of statistics and two graphs.

Box 1 -- Summary Statistics

line 1 -- unique run identifier, data subset, and query construction method used

Data subset
- full (disks 1 and 2 for adhoc, disk 2 for routing)
- category B (the official subset of data, 1/4 of the data using the Wall Street Journal articles)
- disk 1 only
- disk 2 only
- wsj, disk 1 only (Wall Street Journal from disk 1)
- wsj, disk 2 only (Wall Street Journal from disk 2)

Query construction method
- automatic (method 1)
- manual (method 2)
- feedback (method 3, frozen evaluation used)

line 2 -- Number of topics included in averages.

line 3 -- Total number of documents retrieved over all topics. Here, "retrieved" means having a rank less 200.

line 4 -- Total number of relevant documents for all topics in the collection (whether retrieved or not).

line 5 -- Total number of relevant retrieved documents for this run.

Box 2 -- Recall Level Averages

lines 1-11 -- The average over all topics of the precision at each of the 11 recall points given. Note that this is interpolated precision: e.g., for a particular topic, if the precision at 0.50 recall is greater than the precision at 0.40 recall, then the precision at 0.50 recall is used for both the 0.50 and 0.40 recall levels.

line 12 -- The average precision based on the 11 recall points in lines 1-11.

line 13 -- The average precision based on 3 intermediate recall points (0.2, 0.5, and 0.8).

Box 3 -- Document Level Averages

lines 1-5 -- The average recall and precision after the given number of documents have been retrieved.

Box 4 -- ROC Averages

lines 1-10 -- The average probability of detection (recall) at a fixed probability of false alarm rate (fallout). Ten equally spaced false alarm points in the high precision end of the curve were used, and for each topic, the highest detection value in the region surrounding that false alarm point is selected. The table shows the averages of these points across all topics.

Graph 1 -- Recall-Precision Curve

This is a plot of the data shown in Box 2.

Graph 2 -- Normal Deviate - Detection-False Alarm

This is a plot of the data shown in Box 4, but plotted on probability scales. See Overview, section 4.2 for a reference explaining these plots.

| Summary Statistics | |
|---|---|
| Run number | cityal–full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 9989 |
| Relevant | 16400 |
| Rel_ret | 2722 |

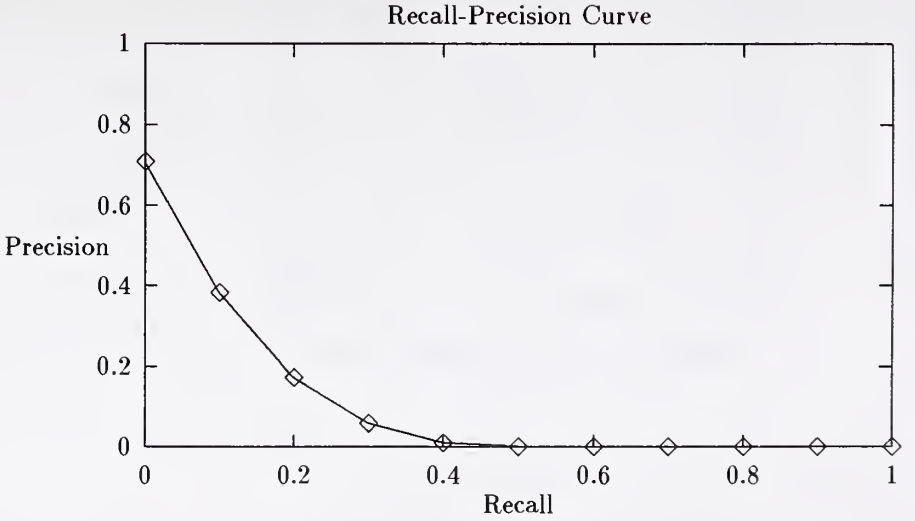
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7089 |
| 0.10 | 0.3855 |
| 0.20 | 0.1727 |
| 0.30 | 0.0594 |
| 0.40 | 0.0089 |
| 0.50 | 0.0000 |
| 0.60 | 0.0000 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

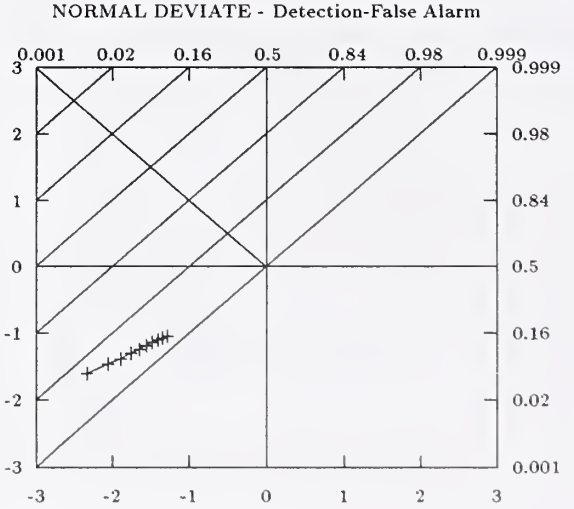
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1214 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0576 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0122 | 0.4960 |
| at 15 docs | 0.0298 | 0.4600 |
| at 30 docs | 0.0504 | 0.4207 |
| at 100 docs | 0.1276 | 0.3462 |
| at 200 docs | 0.1892 | 0.2722 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0379 |
| 0.00001 | 0.0555 |
| 0.00002 | 0.0734 |
| 0.00003 | 0.0847 |
| 0.00004 | 0.0985 |
| 0.00005 | 0.1086 |
| 0.00006 | 0.1200 |
| 0.00007 | 0.1291 |
| 0.00008 | 0.1377 |
| 0.00009 | 0.1442 |



| Summary Statistics | |
|---|---|
| Run number | citym1–full, manual |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 9898 |
| Relevant | 16400 |
| Rel_ret | 3042 |

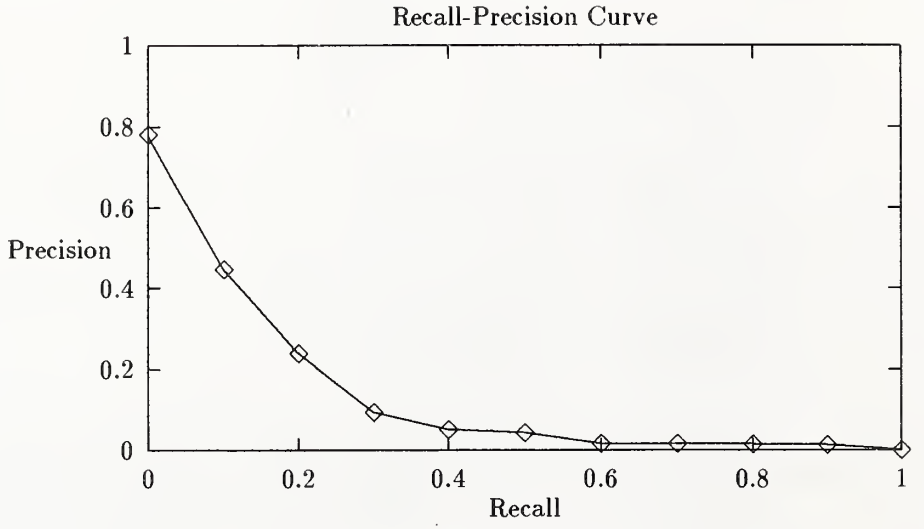
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7813 |
| 0.10 | 0.4481 |
| 0.20 | 0.2399 |
| 0.30 | 0.0938 |
| 0.40 | 0.0506 |
| 0.50 | 0.0425 |
| 0.60 | 0.0166 |
| 0.70 | 0.0154 |
| 0.80 | 0.0147 |
| 0.90 | 0.0133 |
| 1.00 | 0.0000 |

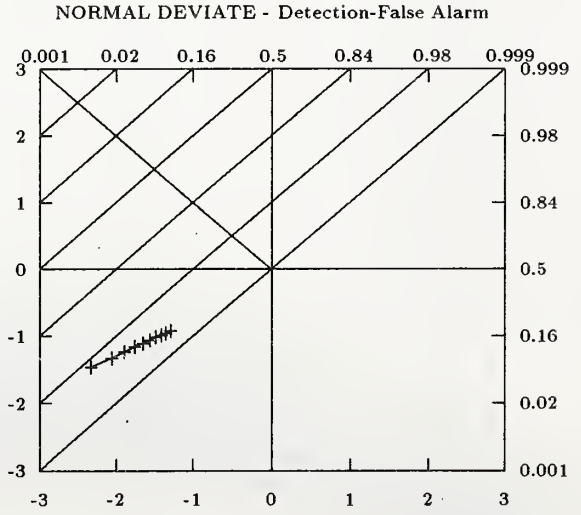
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1560 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0990 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0129 | 0.5760 |
| at 15 docs | 0.0328 | 0.5000 |
| at 30 docs | 0.0593 | 0.4573 |
| at 100 docs | 0.1514 | 0.3818 |
| at 200 docs | 0.2224 | 0.3042 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0525 |
| 0.00001 | 0.0735 |
| 0.00002 | 0.0940 |
| 0.00003 | 0.1097 |
| 0.00004 | 0.1243 |
| 0.00005 | 0.1360 |
| 0.00006 | 0.1456 |
| 0.00007 | 0.1565 |
| 0.00008 | 0.1631 |
| 0.00009 | 0.1715 |



| Summary Statistics | |
|---|---|
| Run number | citym2–full, feedback |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 10829 |
| Relevant | 16400 |
| Rel_ret | 3431 |

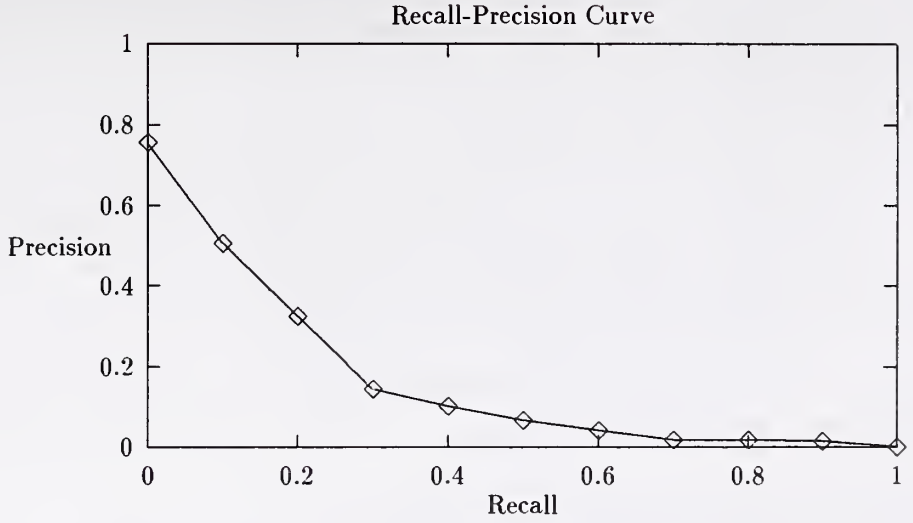
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7569 |
| 0.10 | 0.5072 |
| 0.20 | 0.3262 |
| 0.30 | 0.1442 |
| 0.40 | 0.1016 |
| 0.50 | 0.0667 |
| 0.60 | 0.0418 |
| 0.70 | 0.0174 |
| 0.80 | 0.0170 |
| 0.90 | 0.0143 |
| 1.00 | 0.0000 |

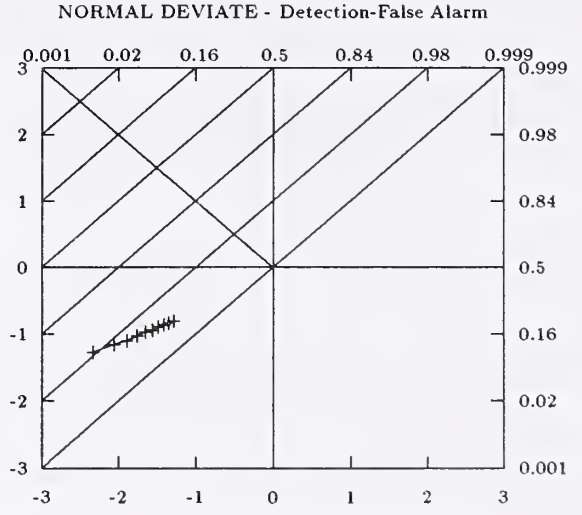
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1812 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1366 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0126 | 0.5800 |
| at 15 docs | 0.0353 | 0.5333 |
| at 30 docs | 0.0654 | 0.5193 |
| at 100 docs | 0.1662 | 0.4304 |
| at 200 docs | 0.2407 | 0.3300 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0795 |
| 0.00001 | 0.1028 |
| 0.00002 | 0.1245 |
| 0.00003 | 0.1377 |
| 0.00004 | 0.1556 |
| 0.00005 | 0.1684 |
| 0.00006 | 0.1766 |
| 0.00007 | 0.1878 |
| 0.00008 | 0.1954 |
| 0.00009 | 0.2047 |



| Summary Statistics | |
|---|---|
| Run number | upitt3–disk2 only, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 5383 |
| Relevant | 5923 |
| Rel_ret | 1249 |

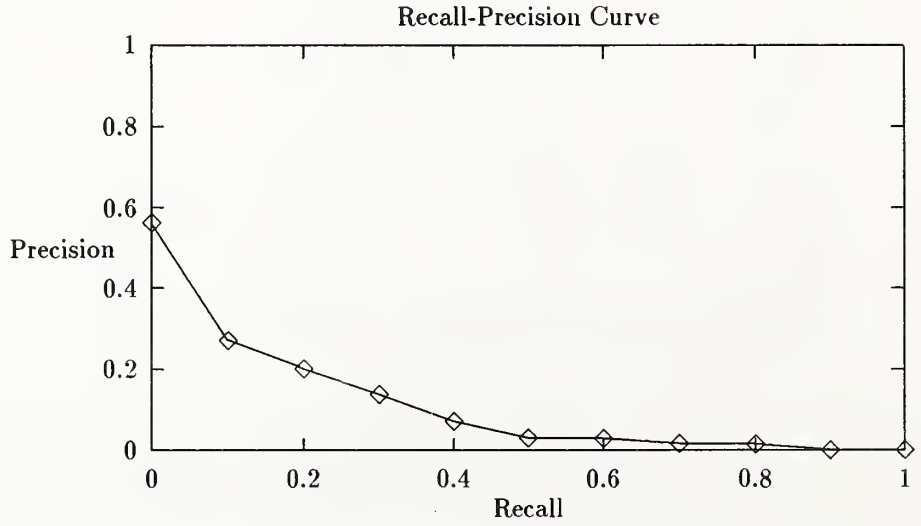
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.5622 |
| 0.10 | 0.2723 |
| 0.20 | 0.2025 |
| 0.30 | 0.1382 |
| 0.40 | 0.0708 |
| 0.50 | 0.0309 |
| 0.60 | 0.0279 |
| 0.70 | 0.0155 |
| 0.80 | 0.0136 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

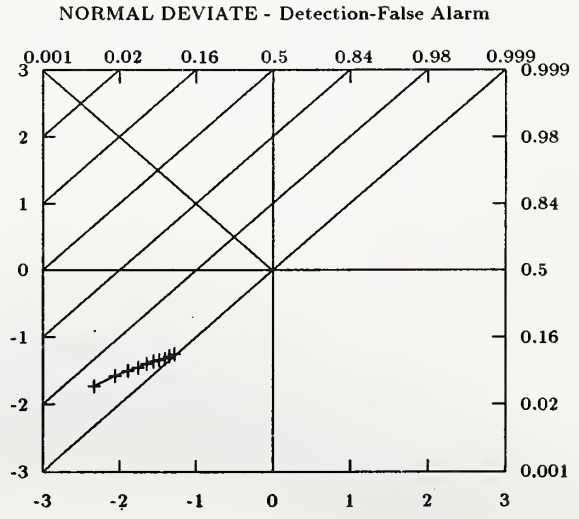
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1213 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0824 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0192 | 0.3280 |
| at 15 docs | 0.0509 | 0.2867 |
| at 30 docs | 0.0849 | 0.2533 |
| at 100 docs | 0.1753 | 0.1844 |
| at 200 docs | 0.2206 | 0.1249 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0303 |
| 0.00001 | 0.0426 |
| 0.00002 | 0.0582 |
| 0.00003 | 0.0675 |
| 0.00004 | 0.0745 |
| 0.00005 | 0.0819 |
| 0.00006 | 0.0881 |
| 0.00007 | 0.0912 |
| 0.00008 | 0.0946 |
| 0.00009 | 0.1017 |



| Summary Statistics | |
|---|---|
| Run number | crnla2–full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 10000 |
| Relevant | 16400 |
| Rel_ret | 3345 |

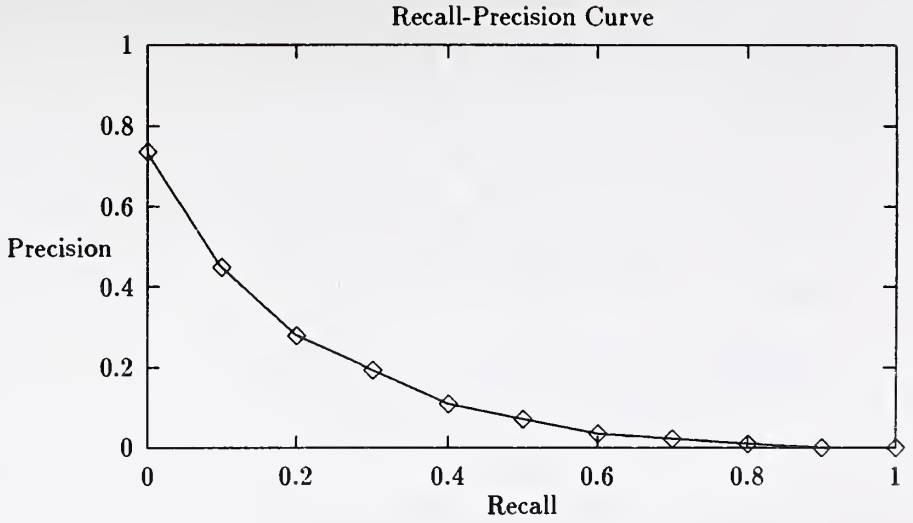
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7367 |
| 0.10 | 0.4506 |
| 0.20 | 0.2803 |
| 0.30 | 0.1945 |
| 0.40 | 0.1102 |
| 0.50 | 0.0714 |
| 0.60 | 0.0357 |
| 0.70 | 0.0229 |
| 0.80 | 0.0095 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

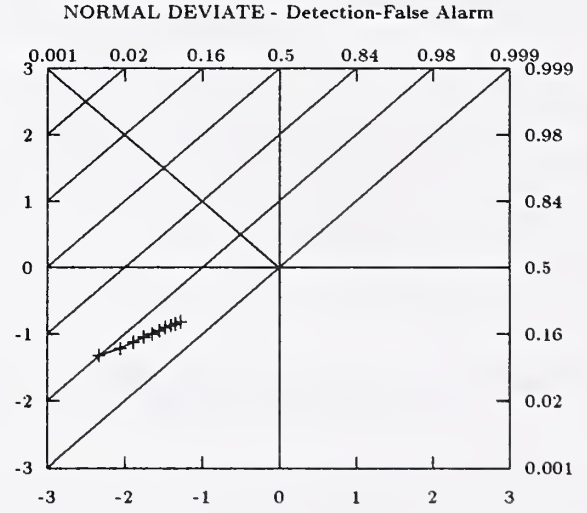
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1738 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1204 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0123 | 0.5120 |
| at 15 docs | 0.0337 | 0.4773 |
| at 30 docs | 0.0613 | 0.4400 |
| at 100 docs | 0.1661 | 0.3972 |
| at 200 docs | 0.2572 | 0.3345 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0624 |
| 0.00001 | 0.0945 |
| 0.00002 | 0.1143 |
| 0.00003 | 0.1340 |
| 0.00004 | 0.1498 |
| 0.00005 | 0.1608 |
| 0.00006 | 0.1737 |
| 0.00007 | 0.1854 |
| 0.00008 | 0.1940 |
| 0.00009 | 0.2025 |



| Summary Statistics | |
|---|---|
| Run number | crnlp2–full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 10000 |
| Relevant | 16400 |
| Rel_ret | 3793 |

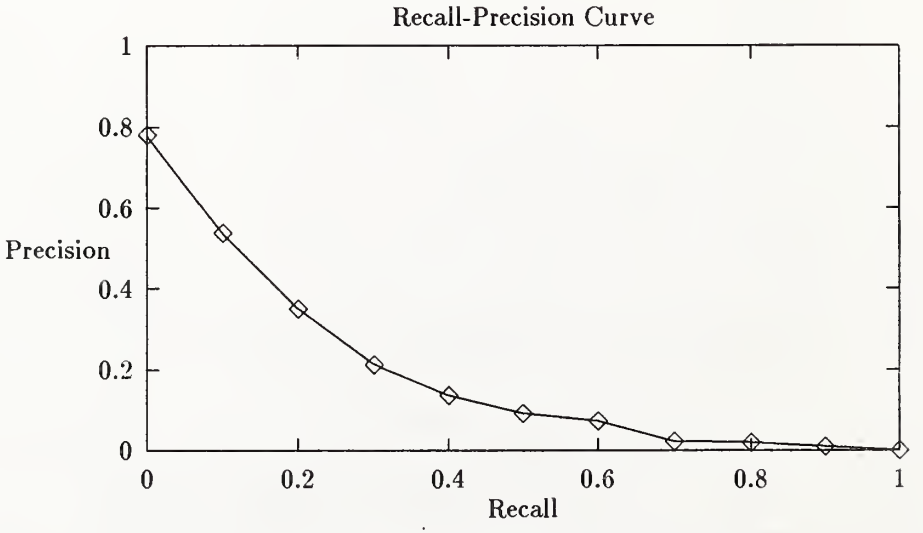
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7803 |
| 0.10 | 0.5389 |
| 0.20 | 0.3507 |
| 0.30 | 0.2134 |
| 0.40 | 0.1366 |
| 0.50 | 0.0917 |
| 0.60 | 0.0724 |
| 0.70 | 0.0229 |
| 0.80 | 0.0187 |
| 0.90 | 0.0098 |
| 1.00 | 0.0000 |

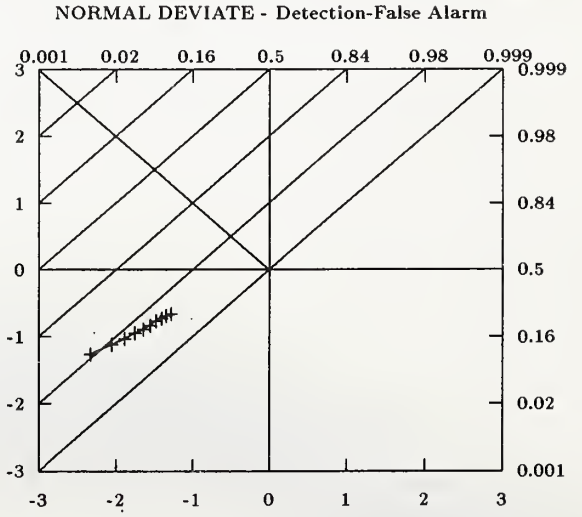
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.2032 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1537 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0145 | 0.5560 |
| at 15 docs | 0.0412 | 0.5400 |
| at 30 docs | 0.0764 | 0.5187 |
| at 100 docs | 0.1931 | 0.4414 |
| at 200 docs | 0.3001 | 0.3793 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0759 |
| 0.00001 | 0.1047 |
| 0.00002 | 0.1335 |
| 0.00003 | 0.1528 |
| 0.00004 | 0.1727 |
| 0.00005 | 0.1893 |
| 0.00006 | 0.2051 |
| 0.00007 | 0.2244 |
| 0.00008 | 0.2362 |
| 0.00009 | 0.2483 |



| Summary Statistics | |
|---|---|
| Run number | Brkly2–full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 10000 |
| Relevant | 16400 |
| Rel_ret | 3394 |

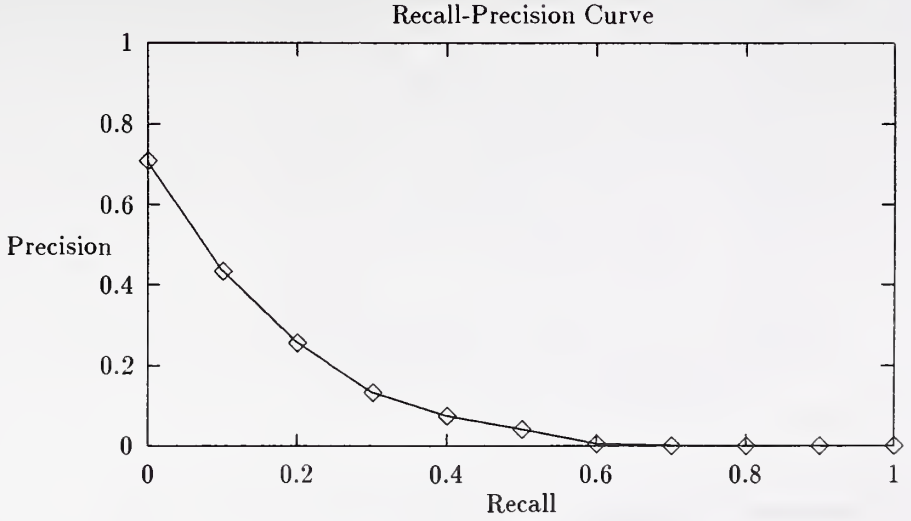
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7093 |
| 0.10 | 0.4362 |
| 0.20 | 0.2583 |
| 0.30 | 0.1333 |
| 0.40 | 0.0754 |
| 0.50 | 0.0408 |
| 0.60 | 0.0045 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

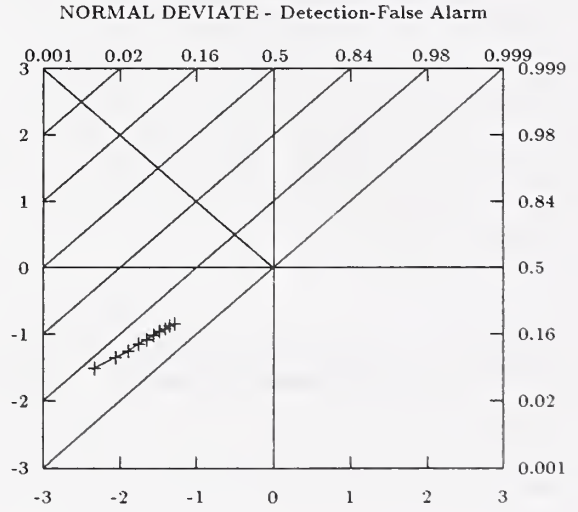
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1507 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0997 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0091 | 0.4400 |
| at 15 docs | 0.0288 | 0.4573 |
| at 30 docs | 0.0545 | 0.4533 |
| at 100 docs | 0.1592 | 0.4080 |
| at 200 docs | 0.2550 | 0.3394 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0403 |
| 0.00001 | 0.0664 |
| 0.00002 | 0.0894 |
| 0.00003 | 0.1064 |
| 0.00004 | 0.1281 |
| 0.00005 | 0.1418 |
| 0.00006 | 0.1568 |
| 0.00007 | 0.1717 |
| 0.00008 | 0.1824 |
| 0.00009 | 0.1938 |



| Summary Statistics | |
|---|---|
| Run number | fuhra2-full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 10000 |
| Relevant | 16400 |
| Rel_ret | 3553 |

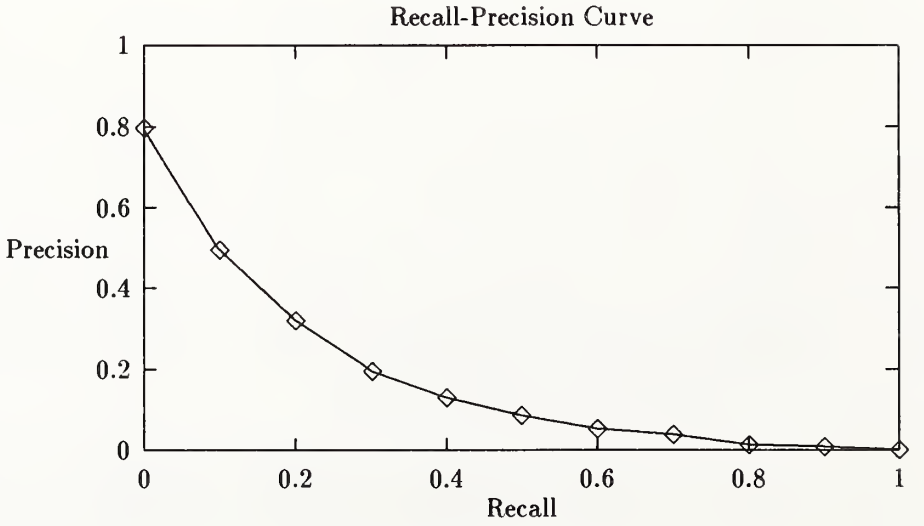
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7966 |
| 0.10 | 0.4968 |
| 0.20 | 0.3204 |
| 0.30 | 0.1961 |
| 0.40 | 0.1302 |
| 0.50 | 0.0864 |
| 0.60 | 0.0527 |
| 0.70 | 0.0376 |
| 0.80 | 0.0128 |
| 0.90 | 0.0072 |
| 1.00 | 0.0000 |

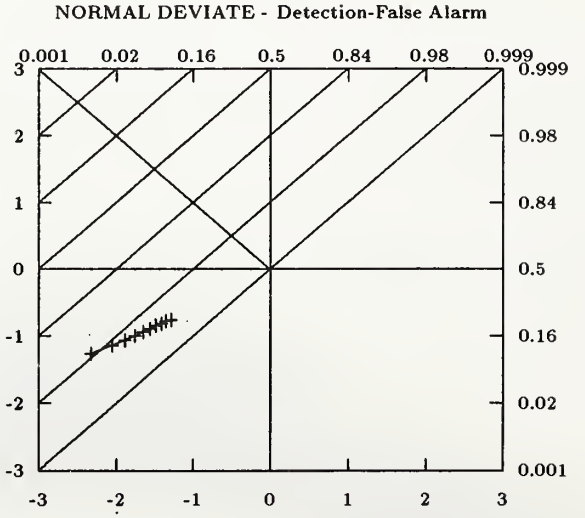
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1943 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1399 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0138 | 0.5800 |
| at 15 docs | 0.0363 | 0.5173 |
| at 30 docs | 0.0675 | 0.5127 |
| at 100 docs | 0.1697 | 0.4232 |
| at 200 docs | 0.2666 | 0.3553 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0718 |
| 0.00001 | 0.1040 |
| 0.00002 | 0.1295 |
| 0.00003 | 0.1460 |
| 0.00004 | 0.1628 |
| 0.00005 | 0.1768 |
| 0.00006 | 0.1895 |
| 0.00007 | 0.2027 |
| 0.00008 | 0.2114 |
| 0.00009 | 0.2214 |



| Summary Statistics | |
|---|---|
| Run number | fuhrp2-full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 10000 |
| Relevant | 16400 |
| Rel_ret | 3613 |

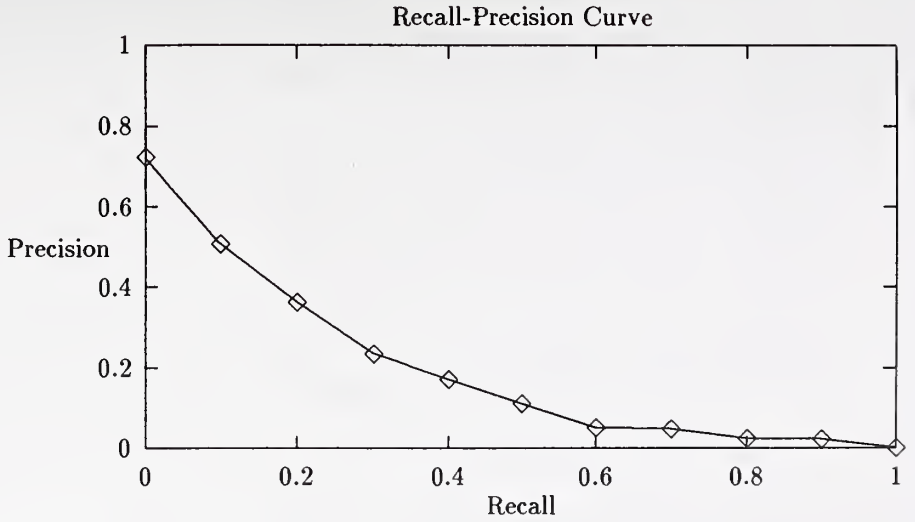
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7236 |
| 0.10 | 0.5083 |
| 0.20 | 0.3641 |
| 0.30 | 0.2350 |
| 0.40 | 0.1722 |
| 0.50 | 0.1116 |
| 0.60 | 0.0508 |
| 0.70 | 0.0479 |
| 0.80 | 0.0234 |
| 0.90 | 0.0228 |
| 1.00 | 0.0000 |

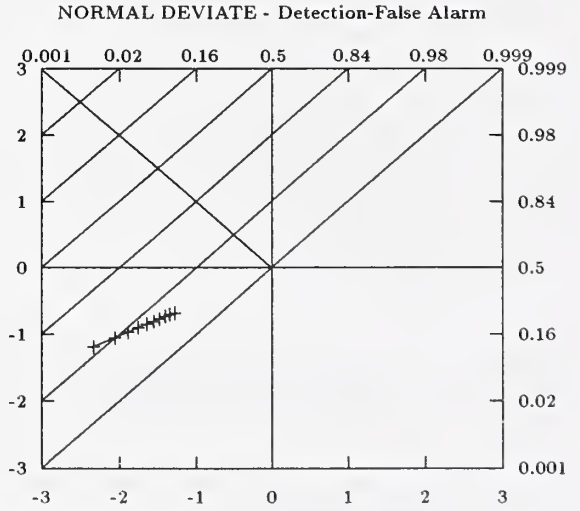
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.2054 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1664 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0142 | 0.5440 |
| at 15 docs | 0.0393 | 0.5293 |
| at 30 docs | 0.0740 | 0.5140 |
| at 100 docs | 0.1950 | 0.4468 |
| at 200 docs | 0.2829 | 0.3613 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0841 |
| 0.00001 | 0.1194 |
| 0.00002 | 0.1477 |
| 0.00003 | 0.1678 |
| 0.00004 | 0.1865 |
| 0.00005 | 0.2027 |
| 0.00006 | 0.2140 |
| 0.00007 | 0.2254 |
| 0.00008 | 0.2346 |
| 0.00009 | 0.2439 |



| Summary Statistics | |
|---|---|
| Run number | danyu1–full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 9588 |
| Relevant | 16400 |
| Rel_ret | 1514 |

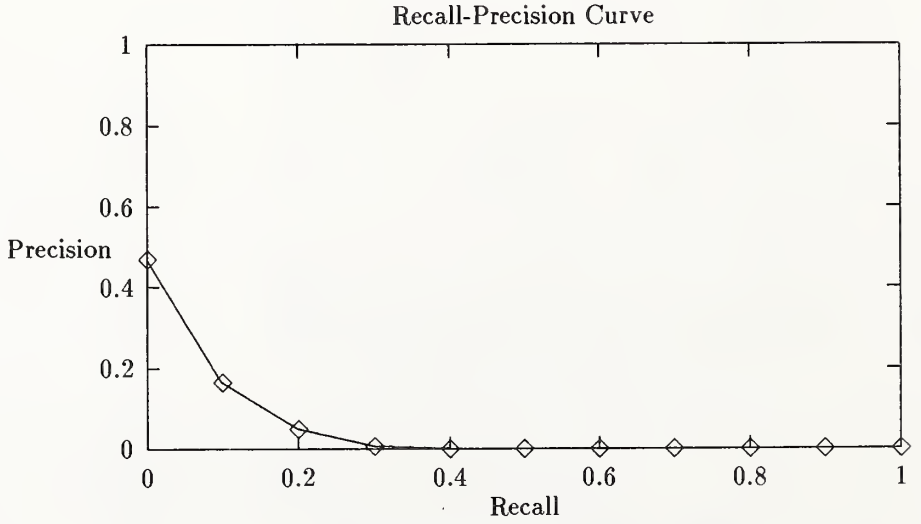
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.4710 |
| 0.10 | 0.1652 |
| 0.20 | 0.0497 |
| 0.30 | 0.0059 |
| 0.40 | 0.0000 |
| 0.50 | 0.0000 |
| 0.60 | 0.0000 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

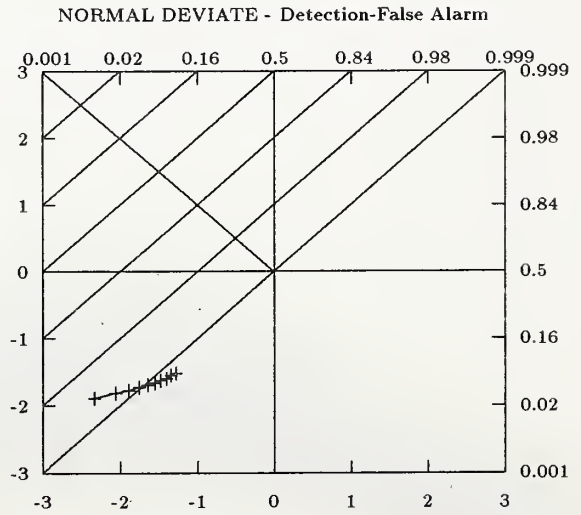
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.0629 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0166 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0065 | 0.2760 |
| at 15 docs | 0.0138 | 0.2253 |
| at 30 docs | 0.0259 | 0.2347 |
| at 100 docs | 0.0578 | 0.1818 |
| at 200 docs | 0.0961 | 0.1514 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0191 |
| 0.00001 | 0.0297 |
| 0.00002 | 0.0350 |
| 0.00003 | 0.0386 |
| 0.00004 | 0.0423 |
| 0.00005 | 0.0458 |
| 0.00006 | 0.0493 |
| 0.00007 | 0.0525 |
| 0.00008 | 0.0563 |
| 0.00009 | 0.0621 |



| Summary Statistics | |
|---|---|
| Run number | lsiac01–full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 9995 |
| Relevant | 16400 |
| Rel_ret | 2786 |

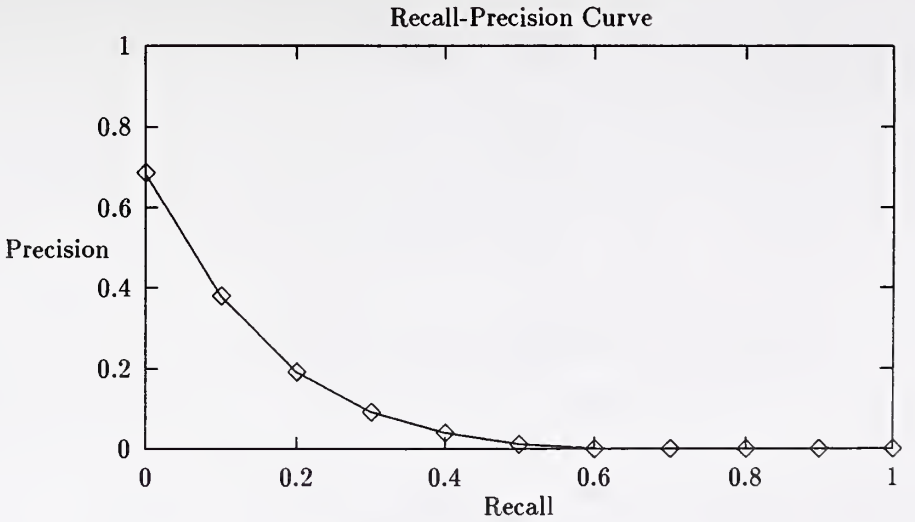
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.6864 |
| 0.10 | 0.3808 |
| 0.20 | 0.1926 |
| 0.30 | 0.0924 |
| 0.40 | 0.0391 |
| 0.50 | 0.0104 |
| 0.60 | 0.0000 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

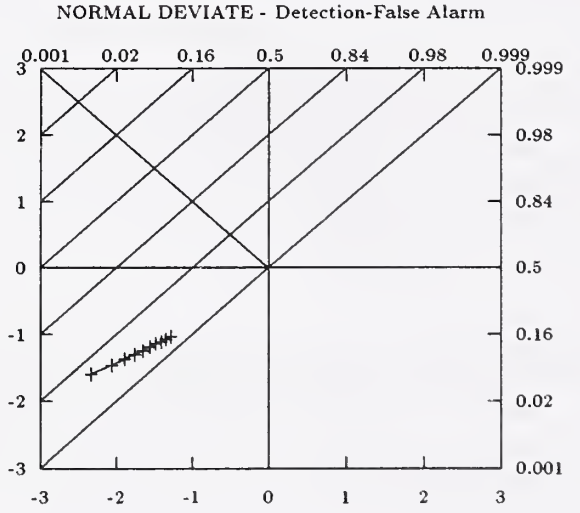
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1274 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0677 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0096 | 0.4640 |
| at 15 docs | 0.0243 | 0.4107 |
| at 30 docs | 0.0458 | 0.3967 |
| at 100 docs | 0.1247 | 0.3446 |
| at 200 docs | 0.1983 | 0.2786 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0393 |
| 0.00001 | 0.0559 |
| 0.00002 | 0.0729 |
| 0.00003 | 0.0867 |
| 0.00004 | 0.0988 |
| 0.00005 | 0.1092 |
| 0.00006 | 0.1196 |
| 0.00007 | 0.1283 |
| 0.00008 | 0.1357 |
| 0.00009 | 0.1442 |



| Summary Statistics | |
|---|---|
| Run number | lsiaz01–full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 9993 |
| Relevant | 16400 |
| Rel_ret | 2469 |

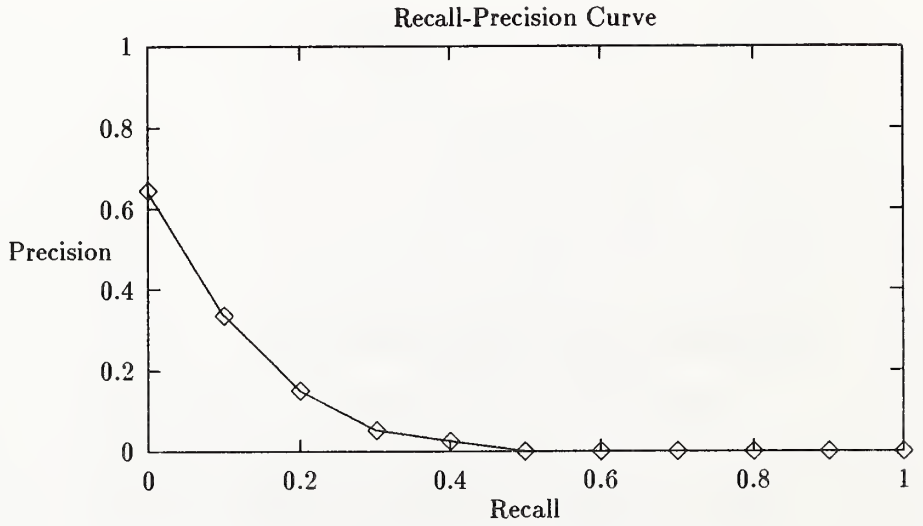
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.6448 |
| 0.10 | 0.3367 |
| 0.20 | 0.1510 |
| 0.30 | 0.0524 |
| 0.40 | 0.0248 |
| 0.50 | 0.0000 |
| 0.60 | 0.0000 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

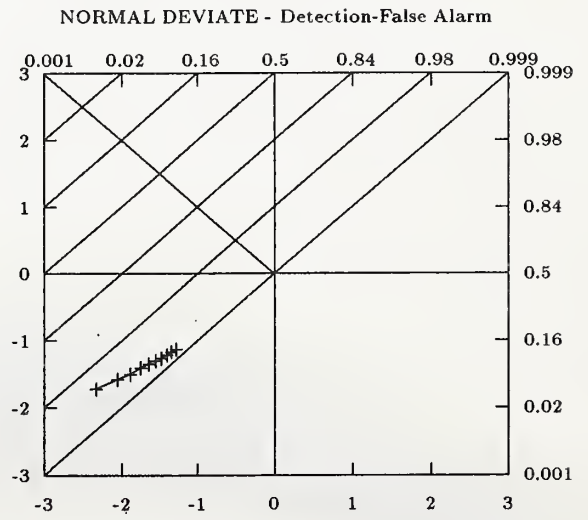
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1100 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0503 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0085 | 0.4320 |
| at 15 docs | 0.0229 | 0.3840 |
| at 30 docs | 0.0391 | 0.3473 |
| at 100 docs | 0.1063 | 0.3010 |
| at 200 docs | 0.1745 | 0.2469 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0311 |
| 0.00001 | 0.0438 |
| 0.00002 | 0.0581 |
| 0.00003 | 0.0683 |
| 0.00004 | 0.0809 |
| 0.00005 | 0.0908 |
| 0.00006 | 0.0983 |
| 0.00007 | 0.1055 |
| 0.00008 | 0.1139 |
| 0.00009 | 0.1242 |



| Summary Statistics | |
|---|---|
| Run number | pircs1–category B, automatic |
| Num_queries | 23 |

| Total number of documents over all queries | |
|---|---|
| Retrieved | 4370 |
| Relevant | 3238 |
| Rel_ret | 1237 |

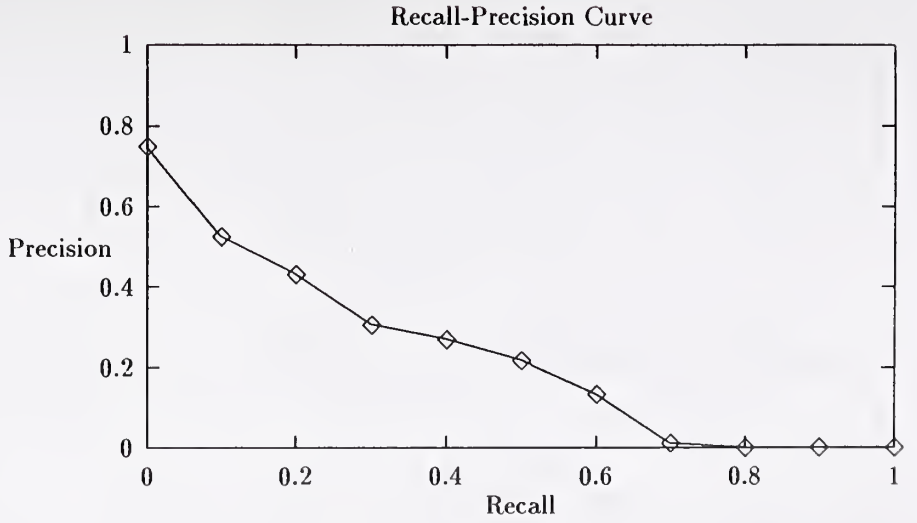
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7481 |
| 0.10 | 0.5259 |
| 0.20 | 0.4328 |
| 0.30 | 0.3071 |
| 0.40 | 0.2707 |
| 0.50 | 0.2173 |
| 0.60 | 0.1333 |
| 0.70 | 0.0105 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

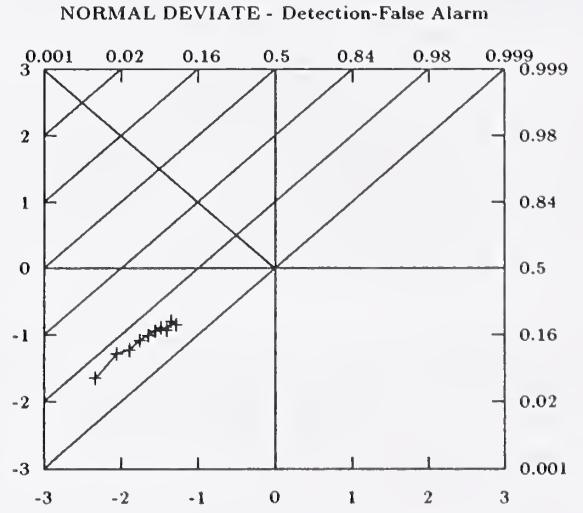
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.2405 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.2167 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0321 | 0.4522 |
| at 15 docs | 0.0938 | 0.4928 |
| at 30 docs | 0.1649 | 0.4507 |
| at 100 docs | 0.3595 | 0.3570 |
| at 200 docs | 0.5018 | 0.2689 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0454 |
| 0.00001 | 0.0505 |
| 0.00002 | 0.1007 |
| 0.00003 | 0.1111 |
| 0.00004 | 0.1418 |
| 0.00005 | 0.1588 |
| 0.00006 | 0.1771 |
| 0.00007 | 0.1890 |
| 0.00008 | 0.1797 |
| 0.00009 | 0.2151 |



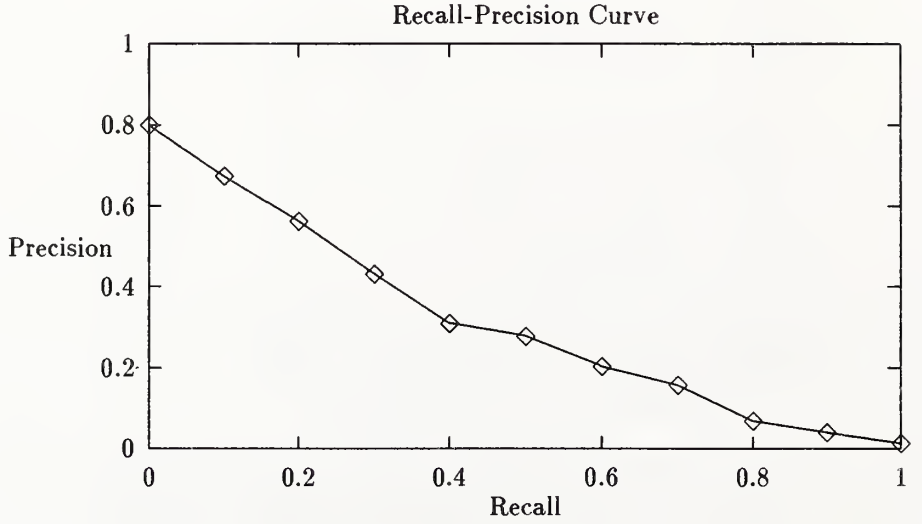



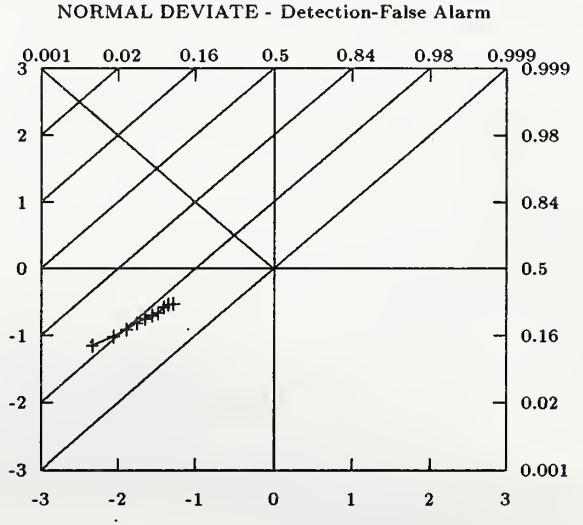
| Summary Statistics | |
|---|---|
| Run number | pircs2–category B, manual |
| Num_queries | 25 |
| **Total number of documents over all queries** | |
| Retrieved | 5000 |
| Relevant | 3318 |
| Rel_ret | 1414 |

| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7998 |
| 0.10 | 0.6742 |
| 0.20 | 0.5626 |
| 0.30 | 0.4322 |
| 0.40 | 0.3106 |
| 0.50 | 0.2783 |
| 0.60 | 0.2037 |
| 0.70 | 0.1580 |
| 0.80 | 0.0681 |
| 0.90 | 0.0396 |
| 1.00 | 0.0133 |
| **Average precision for all points** | |
| 11-pt Avg | 0.3219 |
| **Average precision for 3 intermediate points (0.20, 0.50, 0.80)** | |
| 3-pt Avg | 0.3030 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0665 | 0.6320 |
| at 15 docs | 0.1340 | 0.5307 |
| at 30 docs | 0.2156 | 0.4787 |
| at 100 docs | 0.4245 | 0.3680 |
| at 200 docs | 0.5876 | 0.2828 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.1084 |
| 0.00001 | 0.1273 |
| 0.00002 | 0.1565 |
| 0.00003 | 0.1829 |
| 0.00004 | 0.2107 |
| 0.00005 | 0.2313 |
| 0.00006 | 0.2436 |
| 0.00007 | 0.2574 |
| 0.00008 | 0.2840 |
| 0.00009 | 0.2969 |



| Summary Statistics | |
|---|---|
| Run number | pircs3–category B, feedback |
| Num_queries | 25 |
| **Total number of documents over all queries** | |
| Retrieved | 5230 |
| Relevant | 3318 |
| Rel_ret | 1526 |

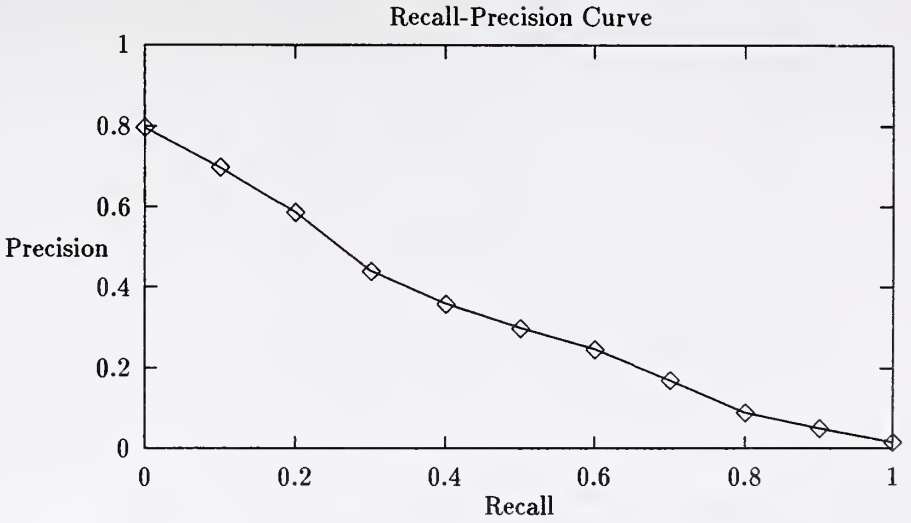
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7970 |
| 0.10 | 0.6978 |
| 0.20 | 0.5873 |
| 0.30 | 0.4398 |
| 0.40 | 0.3595 |
| 0.50 | 0.2982 |
| 0.60 | 0.2460 |
| 0.70 | 0.1700 |
| 0.80 | 0.0898 |
| 0.90 | 0.0489 |
| 1.00 | 0.0136 |

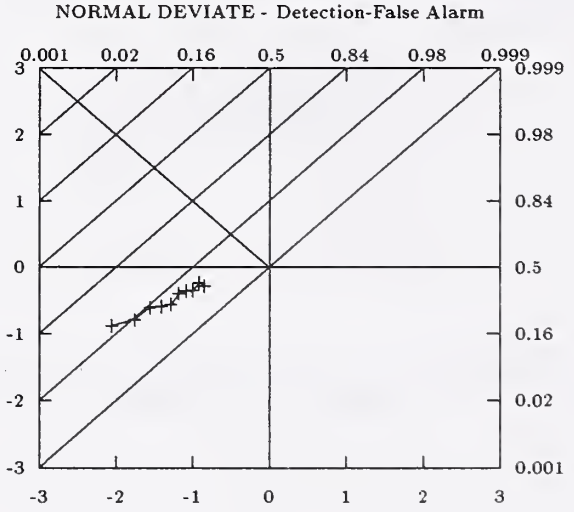
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.3407 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.3251 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0725 | 0.6400 |
| at 15 docs | 0.1325 | 0.5467 |
| at 30 docs | 0.2090 | 0.4920 |
| at 100 docs | 0.4559 | 0.3920 |
| at 200 docs | 0.6163 | 0.2970 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.1409 |
| 0.00002 | 0.1911 |
| 0.00004 | 0.2198 |
| 0.00006 | 0.2751 |
| 0.00008 | 0.2806 |
| 0.00010 | 0.2931 |
| 0.00012 | 0.3500 |
| 0.00014 | 0.3654 |
| 0.00016 | 0.3668 |
| 0.00018 | 0.4119 |
| 0.00020 | 0.3941 |



| Summary Statistics | |
|---|---|
| Run number | pircs4–category B, feedback |
| Num_queries | 25 |

| Total number of documents over all queries | |
|---|---|
| Retrieved | 5230 |
| Relevant | 3318 |
| Rel_ret | 1582 |

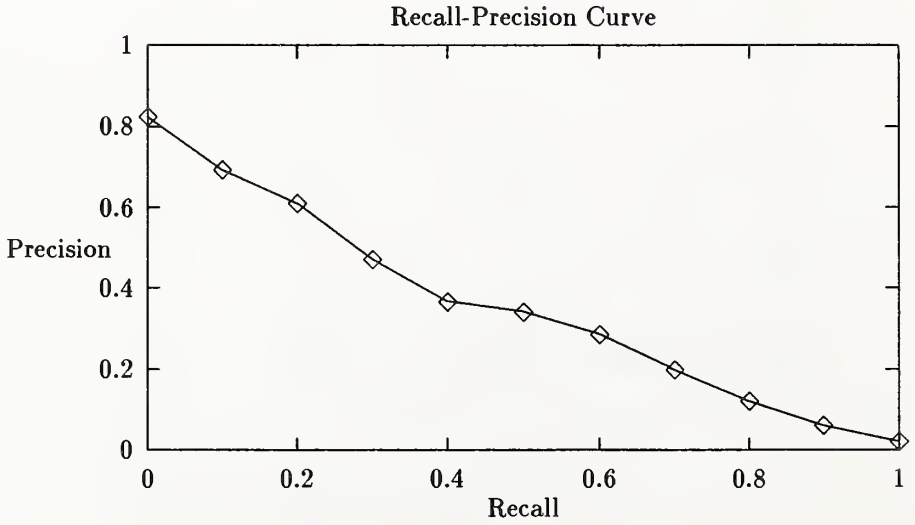
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.8242 |
| 0.10 | 0.6935 |
| 0.20 | 0.6105 |
| 0.30 | 0.4726 |
| 0.40 | 0.3672 |
| 0.50 | 0.3418 |
| 0.60 | 0.2869 |
| 0.70 | 0.1983 |
| 0.80 | 0.1206 |
| 0.90 | 0.0609 |
| 1.00 | 0.0207 |

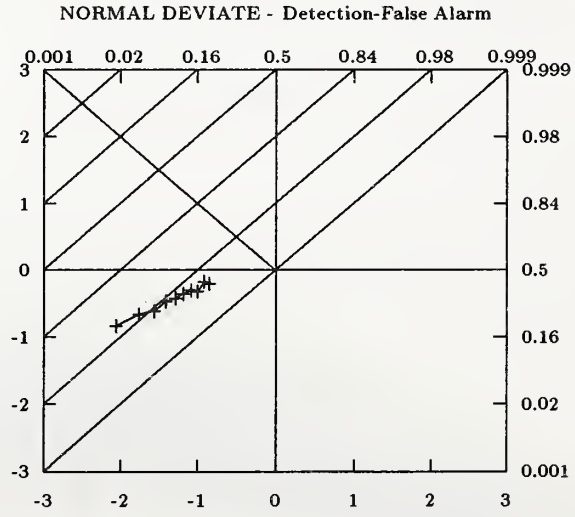
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.3634 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.3576 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0695 | 0.6240 |
| at 15 docs | 0.1389 | 0.5707 |
| at 30 docs | 0.2272 | 0.5187 |
| at 100 docs | 0.4774 | 0.4152 |
| at 200 docs | 0.6339 | 0.3074 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.1608 |
| 0.00002 | 0.2035 |
| 0.00004 | 0.2576 |
| 0.00006 | 0.2715 |
| 0.00008 | 0.3202 |
| 0.00010 | 0.3380 |
| 0.00012 | 0.3645 |
| 0.00014 | 0.3857 |
| 0.00016 | 0.3791 |
| 0.00018 | 0.4337 |
| 0.00020 | 0.4211 |



| Summary Statistics | |
|---|---|
| Run number | nyuir1–category B, automatic |
| Num_queries | 25 |

| Total number of documents over all queries | |
|---|---|
| Retrieved | 4986 |
| Relevant | 3318 |
| Rel_ret | 1161 |

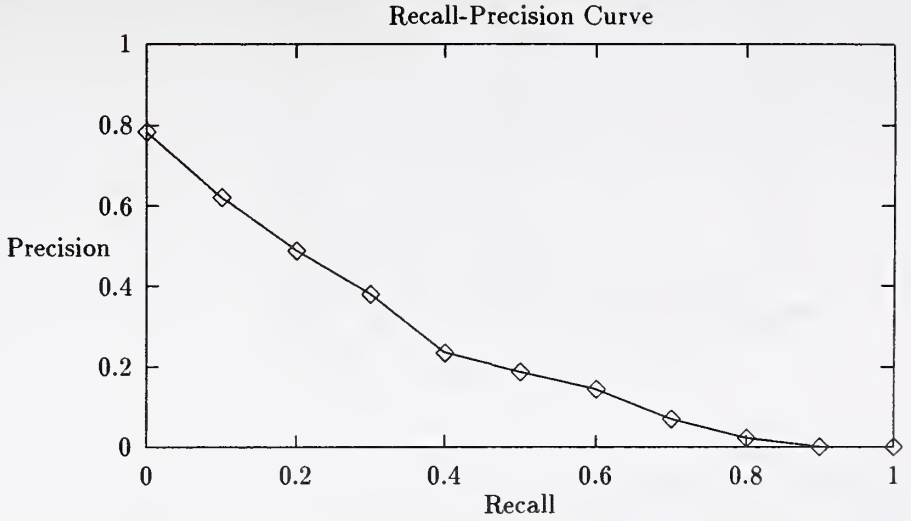
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7830 |
| 0.10 | 0.6216 |
| 0.20 | 0.4891 |
| 0.30 | 0.3815 |
| 0.40 | 0.2350 |
| 0.50 | 0.1871 |
| 0.60 | 0.1447 |
| 0.70 | 0.0705 |
| 0.80 | 0.0229 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

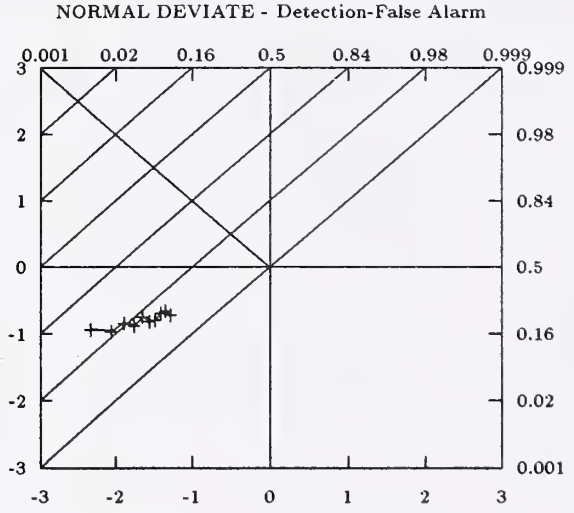
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.2669 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.2330 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0713 | 0.5680 |
| at 15 docs | 0.1326 | 0.5200 |
| at 30 docs | 0.1868 | 0.4320 |
| at 100 docs | 0.3350 | 0.3140 |
| at 200 docs | 0.4294 | 0.2322 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.1363 |
| 0.00001 | 0.1770 |
| 0.00002 | 0.1698 |
| 0.00003 | 0.2026 |
| 0.00004 | 0.1911 |
| 0.00005 | 0.2292 |
| 0.00006 | 0.2109 |
| 0.00007 | 0.2149 |
| 0.00008 | 0.2491 |
| 0.00009 | 0.2578 |



| Summary Statistics | |
|---|---|
| Run number | nyuir2–category B, automatic |
| Num_queries | 25 |
| **Total number of documents over all queries** | |
| Retrieved | 4990 |
| Relevant | 3318 |
| Rel_ret | 958 |

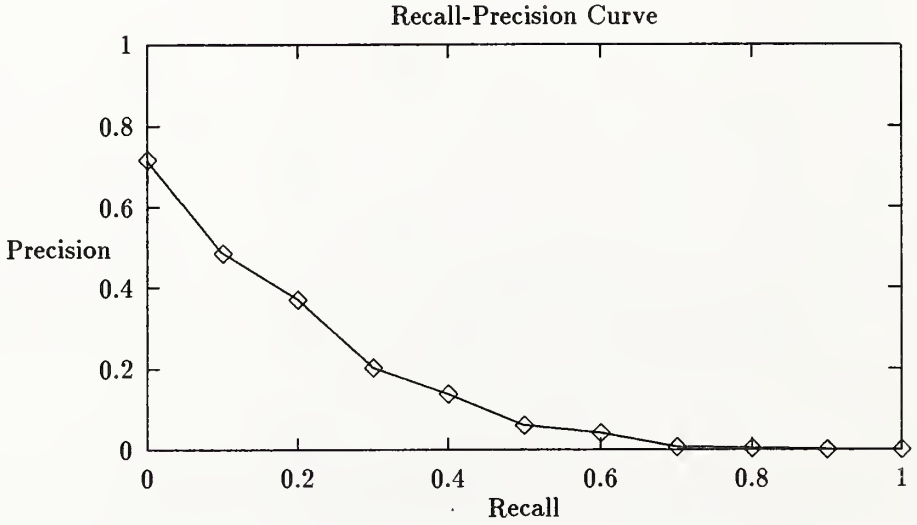
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7168 |
| 0.10 | 0.4858 |
| 0.20 | 0.3722 |
| 0.30 | 0.2032 |
| 0.40 | 0.1382 |
| 0.50 | 0.0599 |
| 0.60 | 0.0419 |
| 0.70 | 0.0058 |
| 0.80 | 0.0030 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

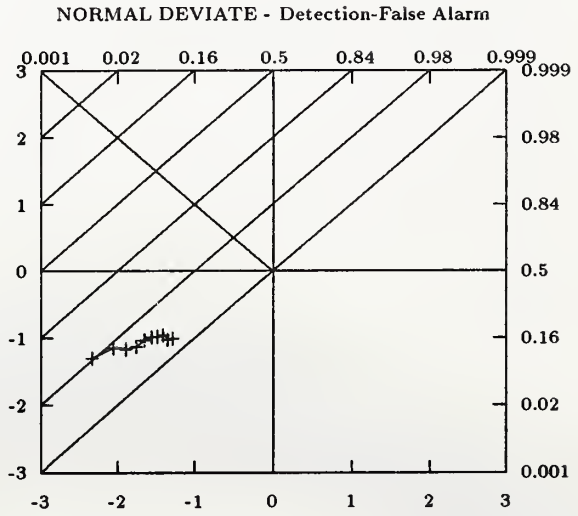
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1843 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1450 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0444 | 0.5200 |
| at 15 docs | 0.0929 | 0.4267 |
| at 30 docs | 0.1424 | 0.3627 |
| at 100 docs | 0.2550 | 0.2684 |
| at 200 docs | 0.3466 | 0.1916 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0754 |
| 0.00001 | 0.0983 |
| 0.00002 | 0.1269 |
| 0.00003 | 0.1227 |
| 0.00004 | 0.1305 |
| 0.00005 | 0.1531 |
| 0.00006 | 0.1639 |
| 0.00007 | 0.1668 |
| 0.00008 | 0.1717 |
| 0.00009 | 0.1549 |



| Summary Statistics | |
|---|---|
| Run number | nyuir3–category B, automatic |
| Num_queries | 25 |

| Total number of documents over all queries | |
|---|---|
| Retrieved | 4989 |
| Relevant | 3318 |
| Rel_ret | 1039 |

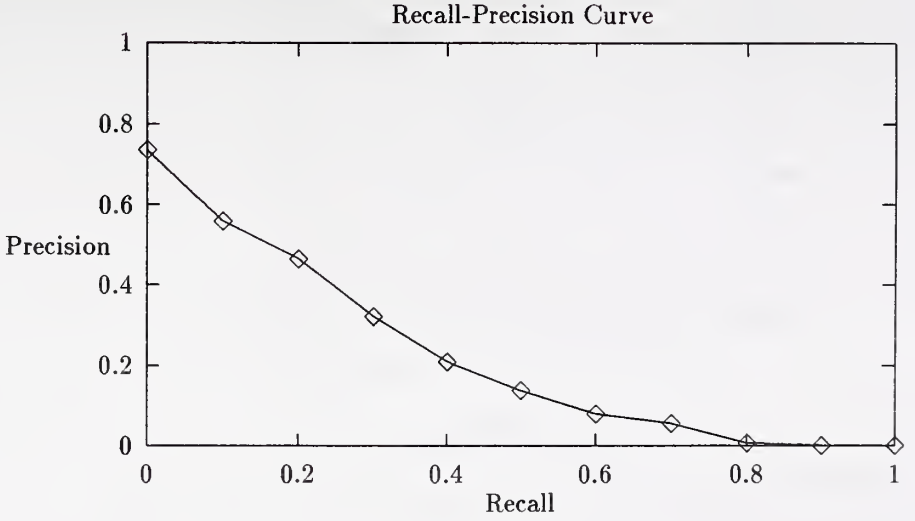
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7368 |
| 0.10 | 0.5592 |
| 0.20 | 0.4660 |
| 0.30 | 0.3235 |
| 0.40 | 0.2099 |
| 0.50 | 0.1385 |
| 0.60 | 0.0801 |
| 0.70 | 0.0560 |
| 0.80 | 0.0056 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

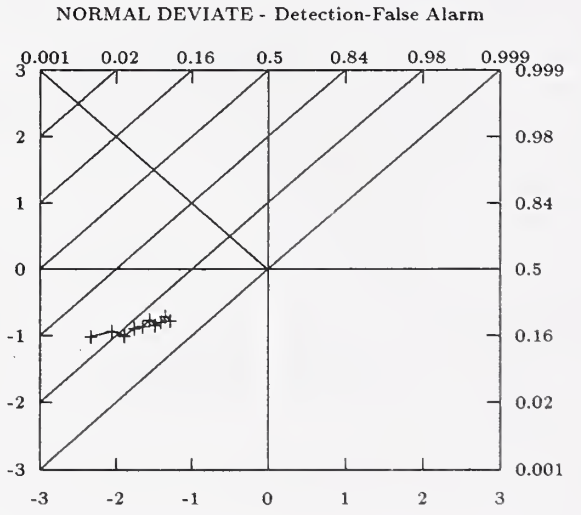
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.2341 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.2034 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0571 | 0.5360 |
| at 15 docs | 0.1297 | 0.4907 |
| at 30 docs | 0.1734 | 0.4147 |
| at 100 docs | 0.2918 | 0.2916 |
| at 200 docs | 0.3908 | 0.2078 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.1289 |
| 0.00001 | 0.1575 |
| 0.00002 | 0.1761 |
| 0.00003 | 0.1594 |
| 0.00004 | 0.1880 |
| 0.00005 | 0.1966 |
| 0.00006 | 0.2254 |
| 0.00007 | 0.2033 |
| 0.00008 | 0.2169 |
| 0.00009 | 0.2424 |



| Summary Statistics | |
|---|---|
| Run number | OCLC_1–wsj, disk2 only, manual |
| Num_queries | 50 |

| Total number of documents over all queries | |
|---|---|
| Retrieved | 10000 |
| Relevant | 2172 |
| Rel_ret | 1274 |

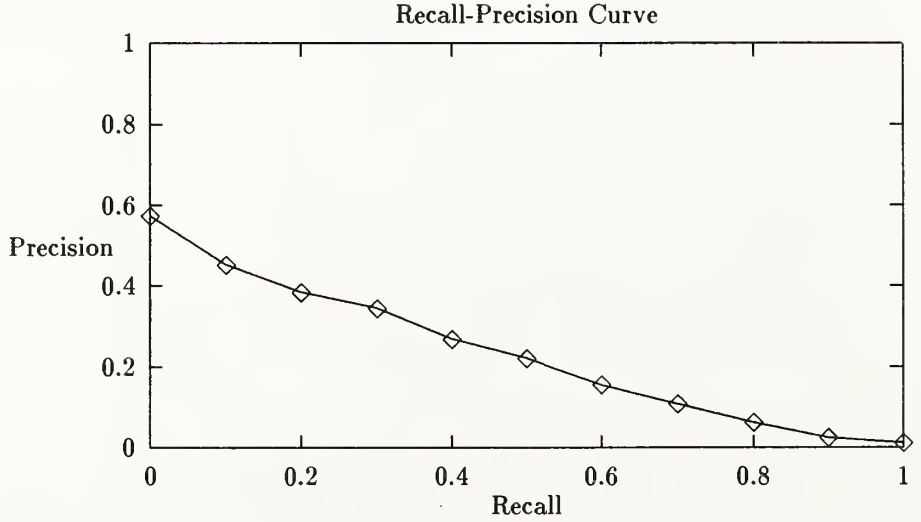
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.5744 |
| 0.10 | 0.4535 |
| 0.20 | 0.3841 |
| 0.30 | 0.3443 |
| 0.40 | 0.2679 |
| 0.50 | 0.2202 |
| 0.60 | 0.1561 |
| 0.70 | 0.1083 |
| 0.80 | 0.0623 |
| 0.90 | 0.0231 |
| 1.00 | 0.0114 |

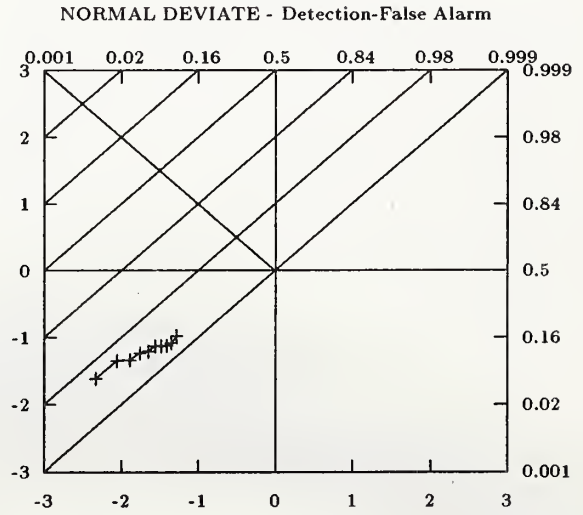
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.2369 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.2222 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0726 | 0.3600 |
| at 15 docs | 0.1514 | 0.3093 |
| at 30 docs | 0.2541 | 0.2753 |
| at 100 docs | 0.4897 | 0.2018 |
| at 200 docs | 0.5930 | 0.1274 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0479 |
| 0.00001 | 0.0538 |
| 0.00002 | 0.0907 |
| 0.00003 | 0.0909 |
| 0.00004 | 0.1104 |
| 0.00005 | 0.1161 |
| 0.00006 | 0.1313 |
| 0.00007 | 0.1315 |
| 0.00008 | 0.1327 |
| 0.00009 | 0.1419 |



| Summary Statistics | |
|---|---|
| Run number | citri2–full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 10000 |
| Relevant | 16400 |
| Rel_ret | 2348 |

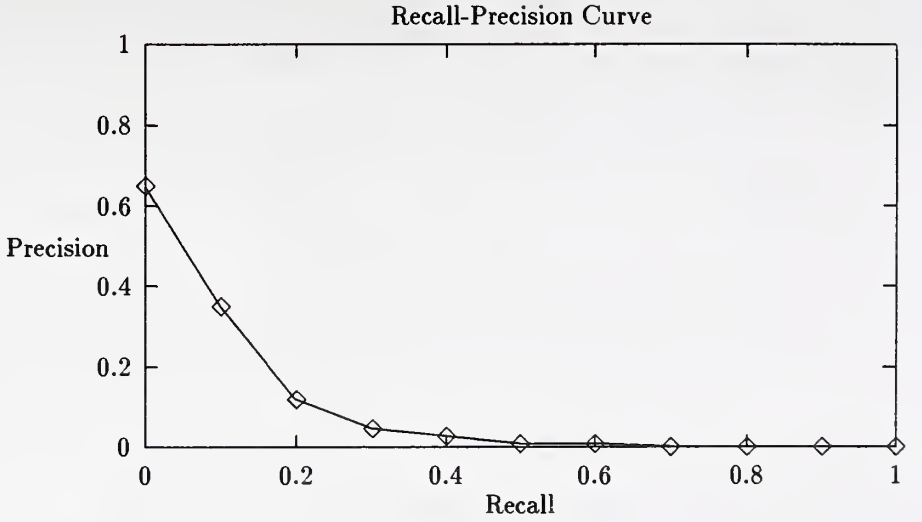
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.6496 |
| 0.10 | 0.3515 |
| 0.20 | 0.1189 |
| 0.30 | 0.0460 |
| 0.40 | 0.0274 |
| 0.50 | 0.0081 |
| 0.60 | 0.0075 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

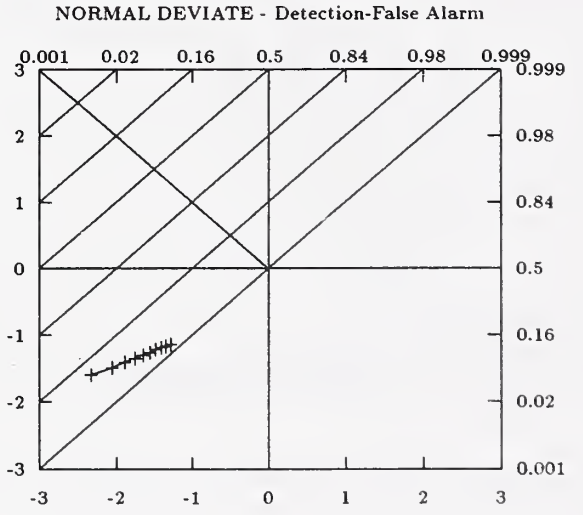
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1099 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0423 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0102 | 0.4440 |
| at 15 docs | 0.0280 | 0.4293 |
| at 30 docs | 0.0483 | 0.3900 |
| at 100 docs | 0.1165 | 0.3198 |
| at 200 docs | 0.1668 | 0.2348 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0404 |
| 0.00001 | 0.0559 |
| 0.00002 | 0.0686 |
| 0.00003 | 0.0810 |
| 0.00004 | 0.0899 |
| 0.00005 | 0.0979 |
| 0.00006 | 0.1056 |
| 0.00007 | 0.1137 |
| 0.00008 | 0.1192 |
| 0.00009 | 0.1239 |



| Summary Statistics | |
|---|---|
| Run number | citri3–full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 10000 |
| Relevant | 16400 |
| Rel_ret | 2011 |

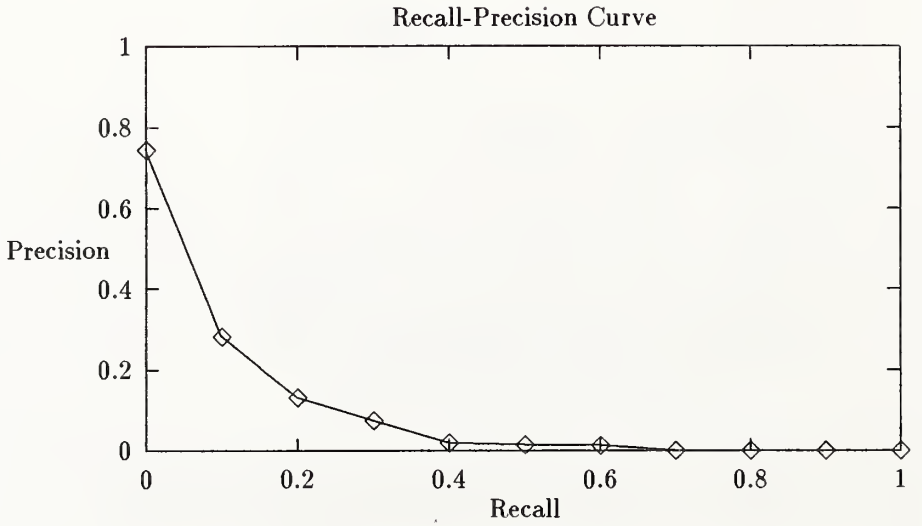
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7436 |
| 0.10 | 0.2834 |
| 0.20 | 0.1325 |
| 0.30 | 0.0742 |
| 0.40 | 0.0192 |
| 0.50 | 0.0147 |
| 0.60 | 0.0120 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

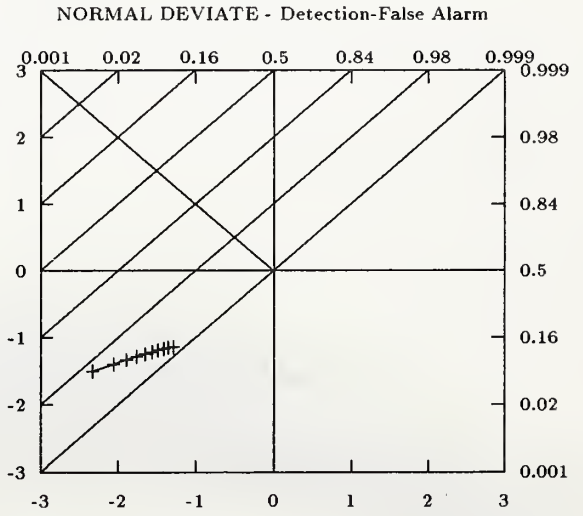
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1163 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0491 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0109 | 0.5000 |
| at 15 docs | 0.0289 | 0.4453 |
| at 30 docs | 0.0553 | 0.4320 |
| at 100 docs | 0.1162 | 0.3080 |
| at 200 docs | 0.1431 | 0.2011 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0480 |
| 0.00001 | 0.0675 |
| 0.00002 | 0.0824 |
| 0.00003 | 0.0931 |
| 0.00004 | 0.1019 |
| 0.00005 | 0.1083 |
| 0.00006 | 0.1140 |
| 0.00007 | 0.1200 |
| 0.00008 | 0.1241 |
| 0.00009 | 0.1269 |



| Summary Statistics | |
|---|---|
| Run number | clarta–full, manual |
| Num_queries | 50 |

| Total number of documents over all queries | |
|---|---|
| Retrieved | 10000 |
| Relevant | 16400 |
| Rel_ret | 2110 |

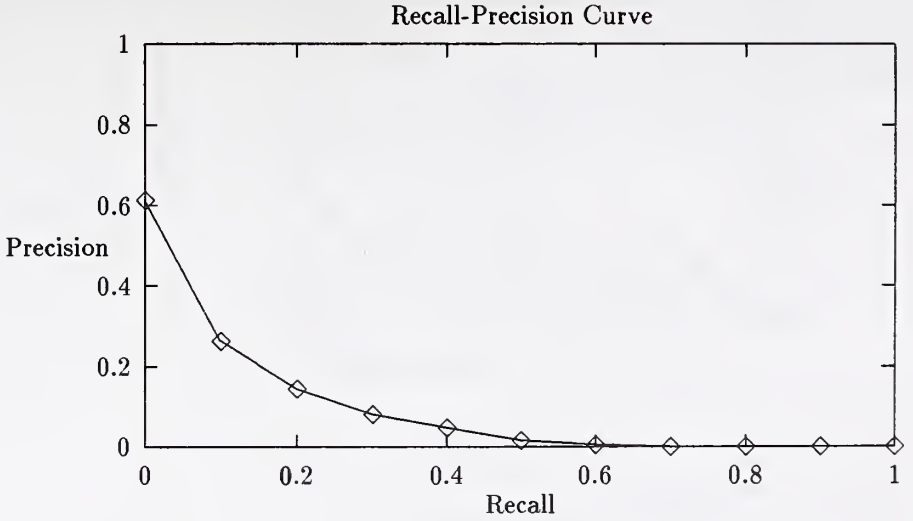
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.6138 |
| 0.10 | 0.2634 |
| 0.20 | 0.1445 |
| 0.30 | 0.0810 |
| 0.40 | 0.0479 |
| 0.50 | 0.0152 |
| 0.60 | 0.0052 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

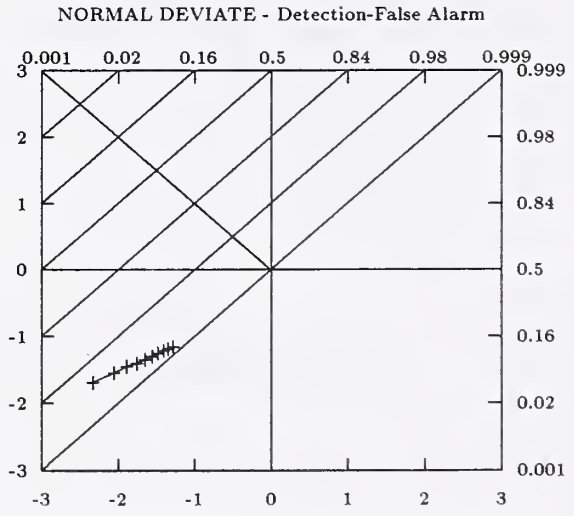
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1065 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0533 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0084 | 0.3920 |
| at 15 docs | 0.0219 | 0.3520 |
| at 30 docs | 0.0420 | 0.3467 |
| at 100 docs | 0.1051 | 0.2732 |
| at 200 docs | 0.1570 | 0.2110 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0316 |
| 0.00001 | 0.0460 |
| 0.00002 | 0.0619 |
| 0.00003 | 0.0736 |
| 0.00004 | 0.0810 |
| 0.00005 | 0.0901 |
| 0.00006 | 0.0975 |
| 0.00007 | 0.1068 |
| 0.00008 | 0.1133 |
| 0.00009 | 0.1201 |



| Summary Statistics | |
|---|---|
| Run number | clartb–full, manual |
| Num_queries | 50 |

| Total number of documents over all queries | |
|---|---|
| Retrieved | 10000 |
| Relevant | 16400 |
| Rel_ret | 3409 |

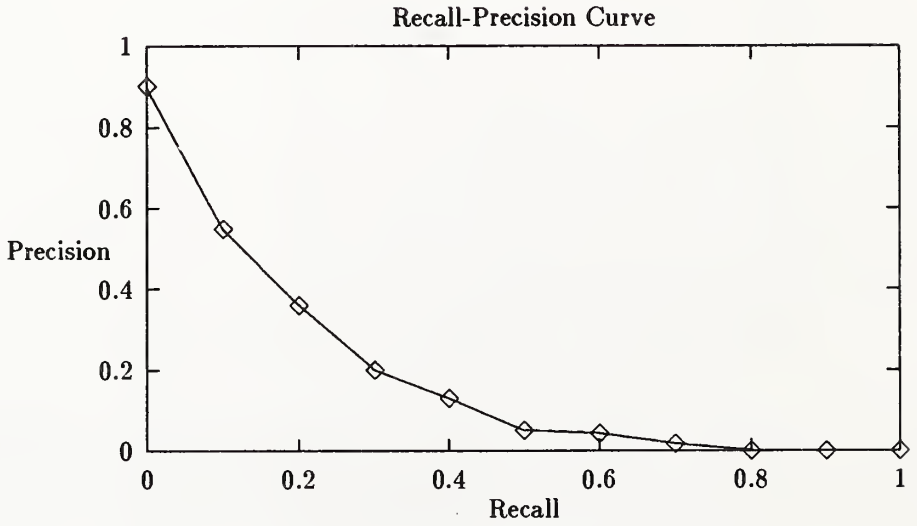
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.9010 |
| 0.10 | 0.5497 |
| 0.20 | 0.3609 |
| 0.30 | 0.1997 |
| 0.40 | 0.1304 |
| 0.50 | 0.0514 |
| 0.60 | 0.0423 |
| 0.70 | 0.0169 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

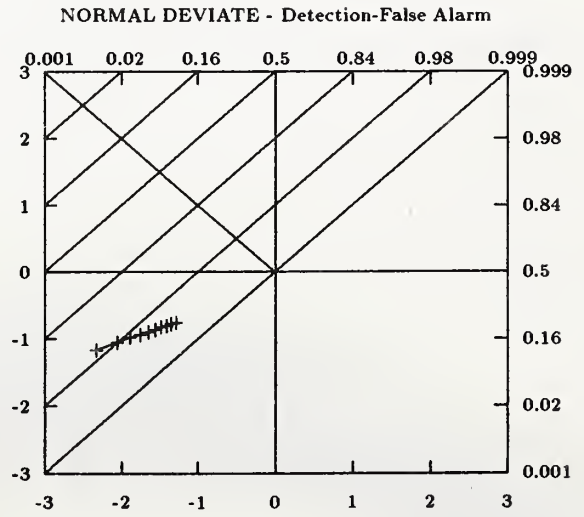
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.2048 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1374 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0179 | 0.7560 |
| at 15 docs | 0.0455 | 0.6533 |
| at 30 docs | 0.0804 | 0.6020 |
| at 100 docs | 0.1834 | 0.4570 |
| at 200 docs | 0.2567 | 0.3409 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0976 |
| 0.00001 | 0.1235 |
| 0.00002 | 0.1471 |
| 0.00003 | 0.1655 |
| 0.00004 | 0.1772 |
| 0.00005 | 0.1868 |
| 0.00006 | 0.1964 |
| 0.00007 | 0.2063 |
| 0.00008 | 0.2136 |
| 0.00009 | 0.2220 |



| Summary Statistics | |
|---|---|
| Run number | CnQst1–full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 10000 |
| Relevant | 16400 |
| Rel_ret | 2779 |

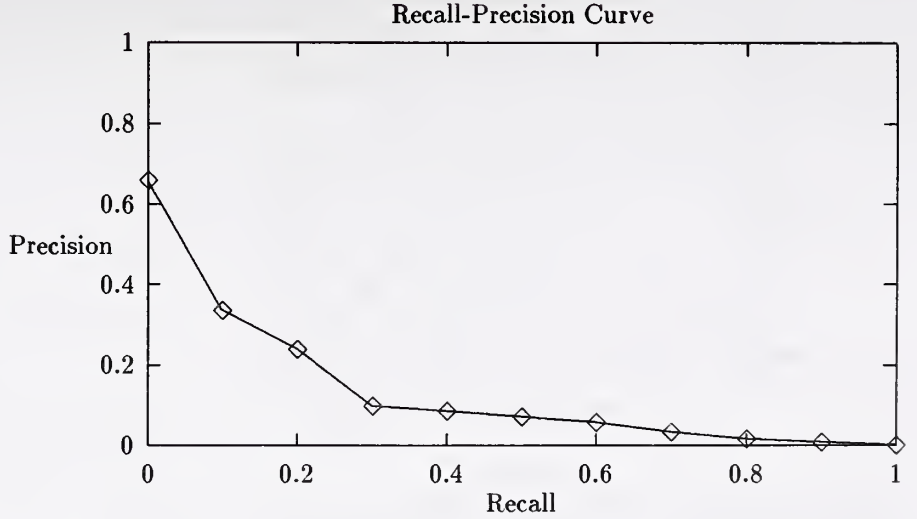
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.6593 |
| 0.10 | 0.3368 |
| 0.20 | 0.2395 |
| 0.30 | 0.0984 |
| 0.40 | 0.0862 |
| 0.50 | 0.0720 |
| 0.60 | 0.0568 |
| 0.70 | 0.0331 |
| 0.80 | 0.0164 |
| 0.90 | 0.0079 |
| 1.00 | 0.0000 |

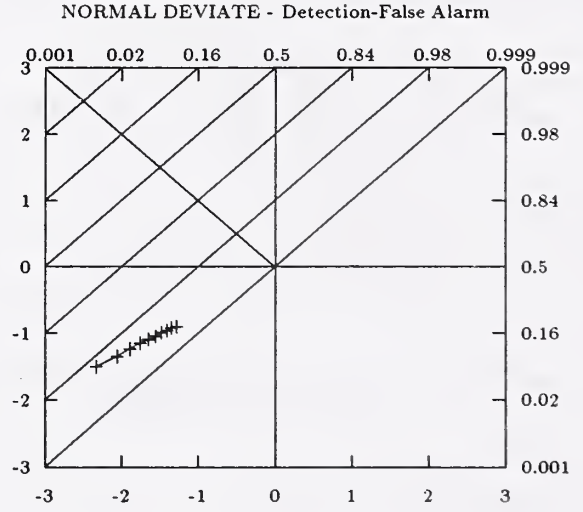
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1461 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1093 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0097 | 0.4160 |
| at 15 docs | 0.0276 | 0.3907 |
| at 30 docs | 0.0531 | 0.3800 |
| at 100 docs | 0.1467 | 0.3408 |
| at 200 docs | 0.2232 | 0.2779 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0364 |
| 0.00001 | 0.0682 |
| 0.00002 | 0.0901 |
| 0.00003 | 0.1097 |
| 0.00004 | 0.1270 |
| 0.00005 | 0.1398 |
| 0.00006 | 0.1506 |
| 0.00007 | 0.1641 |
| 0.00008 | 0.1723 |
| 0.00009 | 0.1801 |



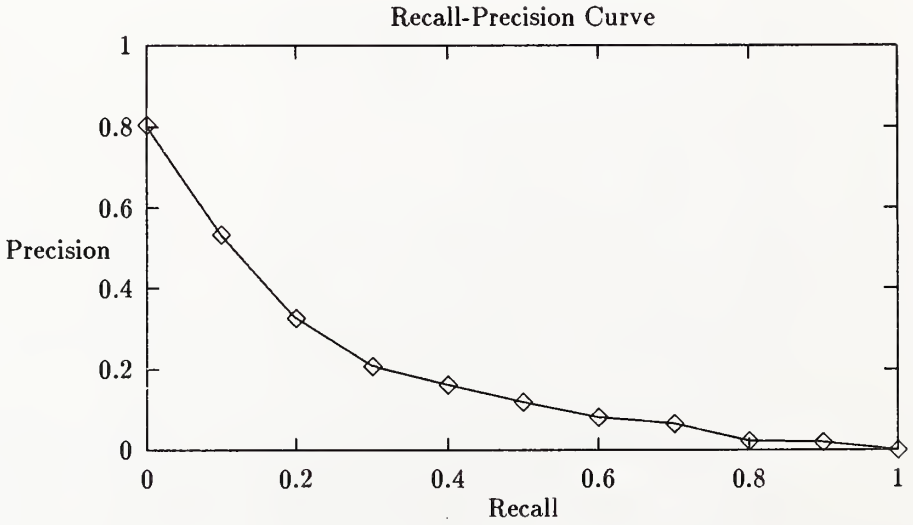



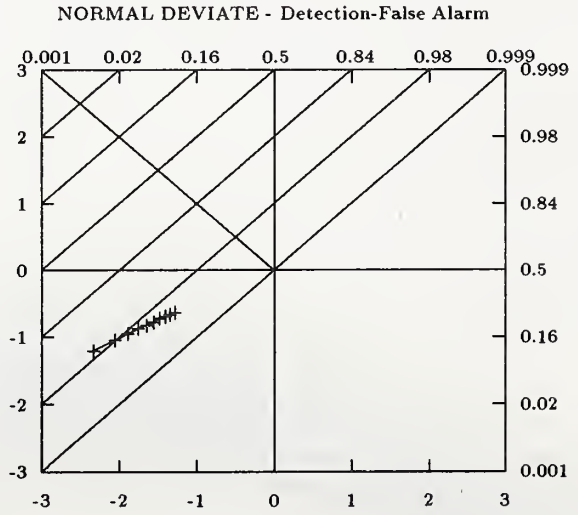
| Summary Statistics | |
|---|---|
| Run number | CnQst2–full, manual |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 10000 |
| Relevant | 16400 |
| Rel_ret | 3700 |

| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.8047 |
| 0.10 | 0.5346 |
| 0.20 | 0.3268 |
| 0.30 | 0.2078 |
| 0.40 | 0.1621 |
| 0.50 | 0.1192 |
| 0.60 | 0.0812 |
| 0.70 | 0.0644 |
| 0.80 | 0.0216 |
| 0.90 | 0.0187 |
| 1.00 | 0.0000 |
| **Average precision for all points** | |
| 11-pt Avg | 0.2128 |
| **Average precision for 3 intermediate points (0.20, 0.50, 0.80)** | |
| 3-pt Avg | 0.1559 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0126 | 0.5520 |
| at 15 docs | 0.0410 | 0.5653 |
| at 30 docs | 0.0733 | 0.5267 |
| at 100 docs | 0.2041 | 0.4730 |
| at 200 docs | 0.2960 | 0.3700 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0787 |
| 0.00001 | 0.1148 |
| 0.00002 | 0.1500 |
| 0.00003 | 0.1740 |
| 0.00004 | 0.1949 |
| 0.00005 | 0.2086 |
| 0.00006 | 0.2222 |
| 0.00007 | 0.2375 |
| 0.00008 | 0.2490 |
| 0.00009 | 0.2571 |



| Summary Statistics | |
|---|---|
| Run number | gecrd1–full, manual |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 7365 |
| Relevant | 16400 |
| Rel_ret | 3363 |

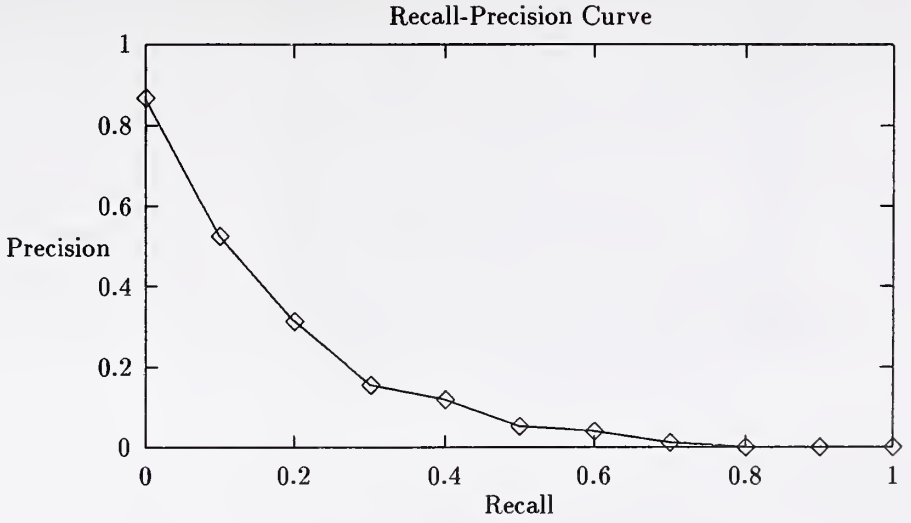
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.8682 |
| 0.10 | 0.5258 |
| 0.20 | 0.3154 |
| 0.30 | 0.1562 |
| 0.40 | 0.1196 |
| 0.50 | 0.0522 |
| 0.60 | 0.0395 |
| 0.70 | 0.0112 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

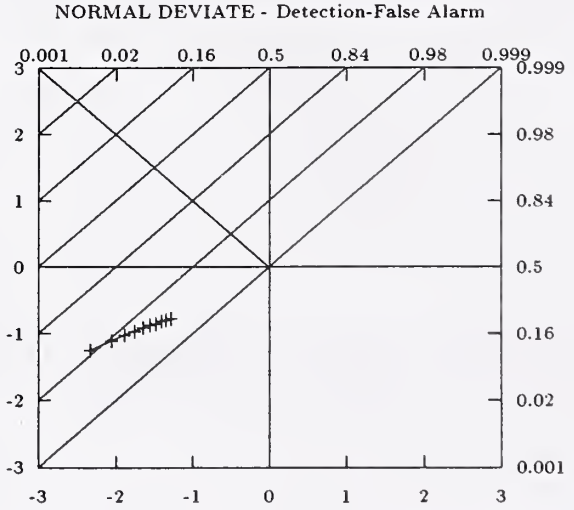
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1898 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1226 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0164 | 0.6640 |
| at 15 docs | 0.0413 | 0.6107 |
| at 30 docs | 0.0725 | 0.5593 |
| at 100 docs | 0.1813 | 0.4508 |
| at 200 docs | 0.2452 | 0.3363 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0824 |
| 0.00001 | 0.1065 |
| 0.00002 | 0.1352 |
| 0.00003 | 0.1554 |
| 0.00004 | 0.1698 |
| 0.00005 | 0.1831 |
| 0.00006 | 0.1933 |
| 0.00007 | 0.2002 |
| 0.00008 | 0.2089 |
| 0.00009 | 0.2166 |



| Summary Statistics | |
|---|---|
| Run number | gecrd2–full, manual |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 8146 |
| Relevant | 16400 |
| Rel_ret | 3689 |

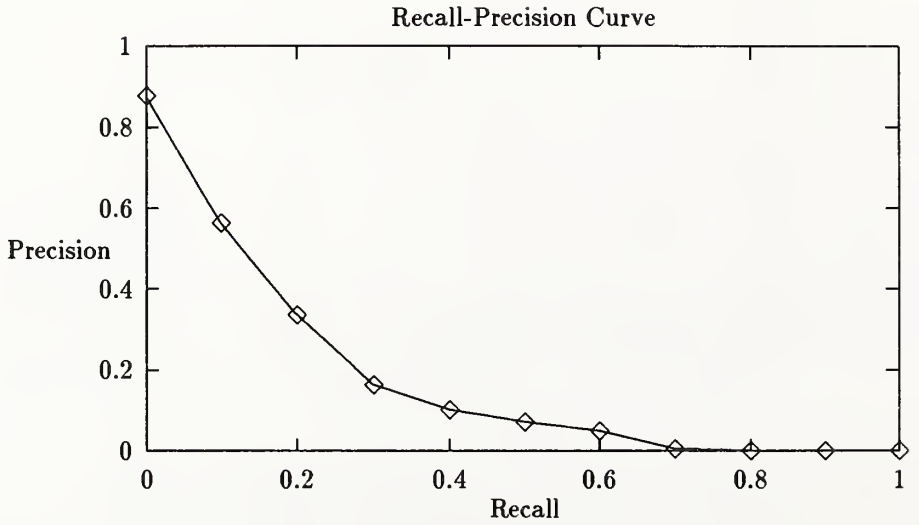
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.8769 |
| 0.10 | 0.5646 |
| 0.20 | 0.3367 |
| 0.30 | 0.1644 |
| 0.40 | 0.1013 |
| 0.50 | 0.0723 |
| 0.60 | 0.0495 |
| 0.70 | 0.0041 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

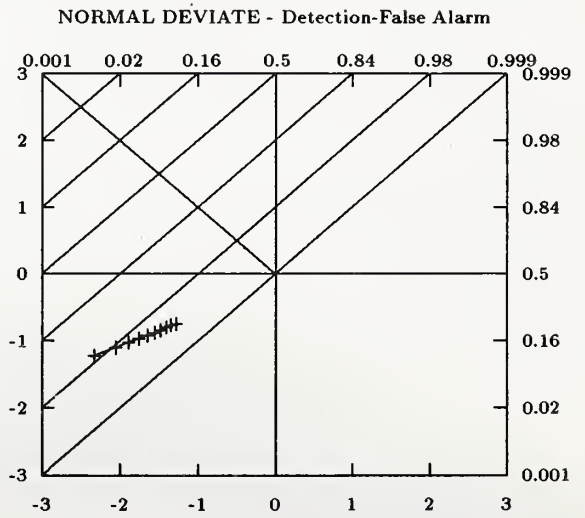
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1973 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1363 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0158 | 0.6480 |
| at 15 docs | 0.0439 | 0.6253 |
| at 30 docs | 0.0768 | 0.5753 |
| at 100 docs | 0.1852 | 0.4628 |
| at 200 docs | 0.2712 | 0.3689 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0807 |
| 0.00001 | 0.1120 |
| 0.00002 | 0.1383 |
| 0.00003 | 0.1544 |
| 0.00004 | 0.1698 |
| 0.00005 | 0.1802 |
| 0.00006 | 0.1912 |
| 0.00007 | 0.2024 |
| 0.00008 | 0.2168 |
| 0.00009 | 0.2249 |



| Summary Statistics | |
|---|---|
| Run number | vpidt2–wsj, disk1 only, manual |
| Num_queries | 50 |

| Total number of documents over all queries | |
|---|---|
| Retrieved | 10000 |
| Relevant | 4056 |
| Rel_ret | 1371 |

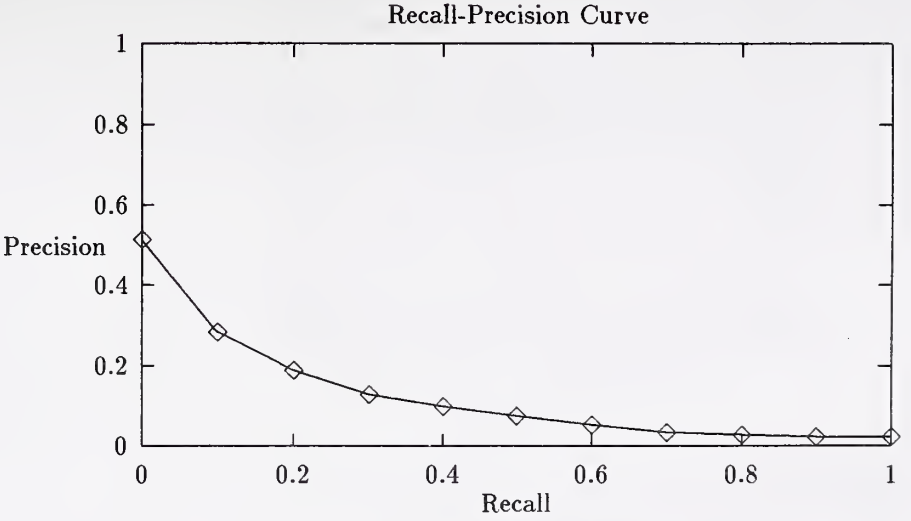
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.5153 |
| 0.10 | 0.2841 |
| 0.20 | 0.1898 |
| 0.30 | 0.1285 |
| 0.40 | 0.0988 |
| 0.50 | 0.0746 |
| 0.60 | 0.0527 |
| 0.70 | 0.0338 |
| 0.80 | 0.0271 |
| 0.90 | 0.0222 |
| 1.00 | 0.0222 |

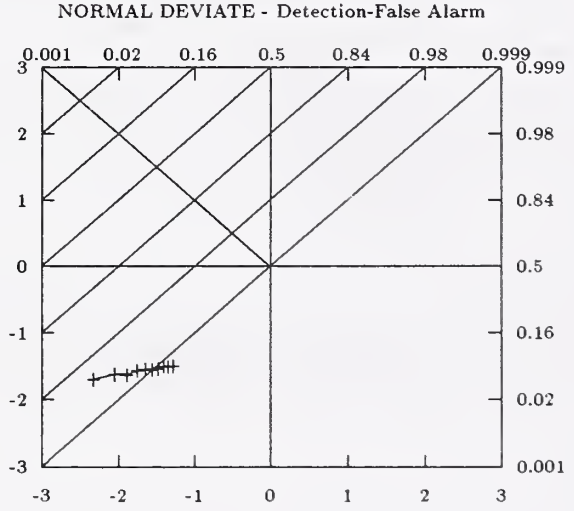
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1317 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0972 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0476 | 0.3160 |
| at 15 docs | 0.0640 | 0.2147 |
| at 30 docs | 0.0843 | 0.1760 |
| at 100 docs | 0.2050 | 0.1246 |
| at 200 docs | 0.4481 | 0.1371 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0391 |
| 0.00001 | 0.0452 |
| 0.00002 | 0.0529 |
| 0.00003 | 0.0522 |
| 0.00004 | 0.0595 |
| 0.00005 | 0.0616 |
| 0.00006 | 0.0607 |
| 0.00007 | 0.0633 |
| 0.00008 | 0.0677 |
| 0.00009 | 0.0661 |



| Summary Statistics | |
|---|---|
| Run number | fairs1–wsj, disk1 only, automatic |
| Num_queries | 50 |

| Total number of documents over all queries | |
|---|---|
| Retrieved | 10000 |
| Relevant | 4056 |
| Rel_ret | 1561 |

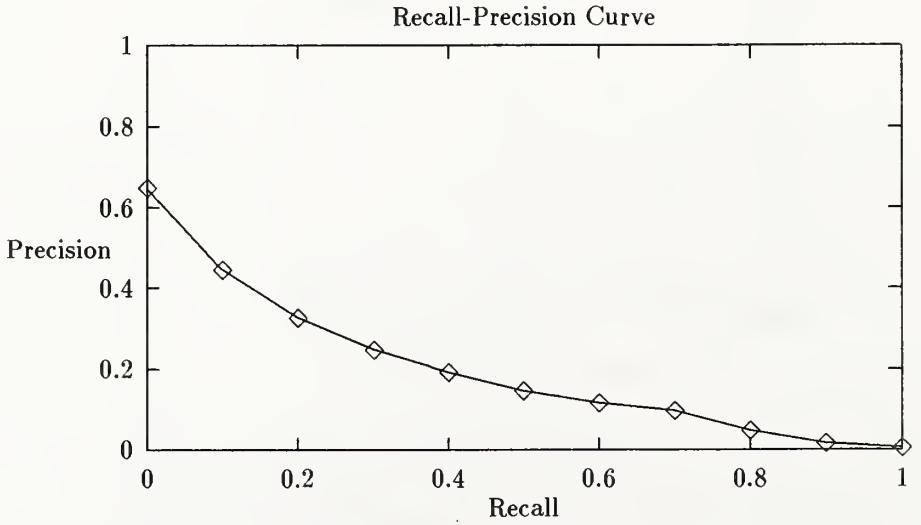
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.6492 |
| 0.10 | 0.4468 |
| 0.20 | 0.3278 |
| 0.30 | 0.2484 |
| 0.40 | 0.1916 |
| 0.50 | 0.1455 |
| 0.60 | 0.1159 |
| 0.70 | 0.0963 |
| 0.80 | 0.0474 |
| 0.90 | 0.0166 |
| 1.00 | 0.0055 |

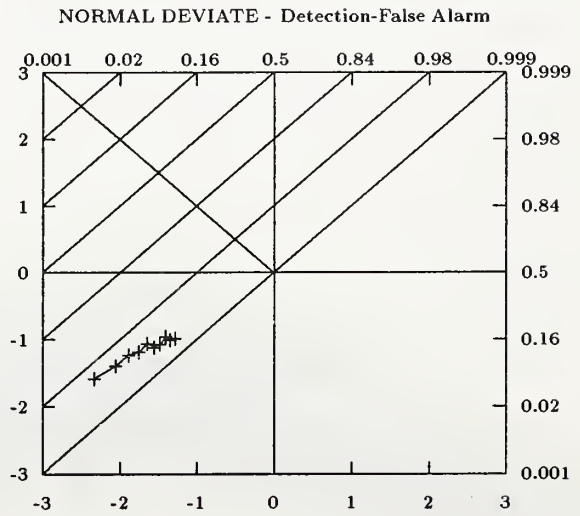
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.2083 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1736 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0769 | 0.4040 |
| at 15 docs | 0.1282 | 0.3387 |
| at 30 docs | 0.1764 | 0.2887 |
| at 100 docs | 0.3653 | 0.2252 |
| at 200 docs | 0.4633 | 0.1561 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0472 |
| 0.00001 | 0.0570 |
| 0.00002 | 0.0821 |
| 0.00003 | 0.1110 |
| 0.00004 | 0.1185 |
| 0.00005 | 0.1454 |
| 0.00006 | 0.1332 |
| 0.00007 | 0.1446 |
| 0.00008 | 0.1704 |
| 0.00009 | 0.1598 |



| Summary Statistics | |
|---|---|
| Run number | fairs2–wsj, disk2 only, automatic |
| Num_queries | 50 |

| Total number of documents over all queries | |
|---|---|
| Retrieved | 10000 |
| Relevant | 2172 |
| Rel_ret | 1188 |

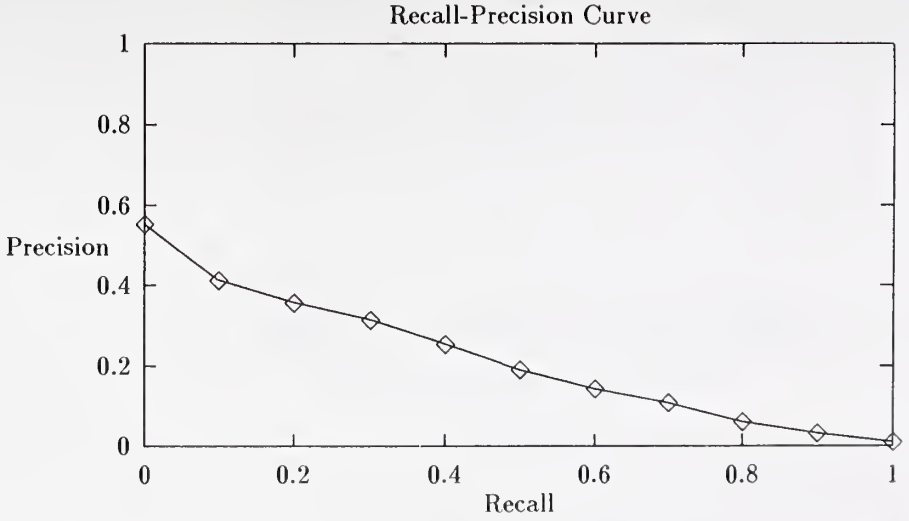
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.5538 |
| 0.10 | 0.4134 |
| 0.20 | 0.3582 |
| 0.30 | 0.3148 |
| 0.40 | 0.2551 |
| 0.50 | 0.1907 |
| 0.60 | 0.1427 |
| 0.70 | 0.1086 |
| 0.80 | 0.0603 |
| 0.90 | 0.0312 |
| 1.00 | 0.0092 |

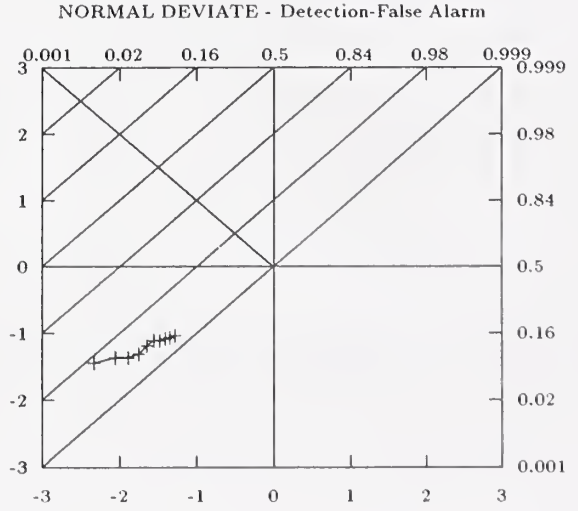
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.2216 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.2030 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0592 | 0.3440 |
| at 15 docs | 0.1500 | 0.2947 |
| at 30 docs | 0.2355 | 0.2673 |
| at 100 docs | 0.4410 | 0.1876 |
| at 200 docs | 0.5362 | 0.1188 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0622 |
| 0.00001 | 0.0758 |
| 0.00002 | 0.0873 |
| 0.00003 | 0.0873 |
| 0.00004 | 0.0964 |
| 0.00005 | 0.1198 |
| 0.00006 | 0.1350 |
| 0.00007 | 0.1350 |
| 0.00008 | 0.1421 |
| 0.00009 | 0.1470 |



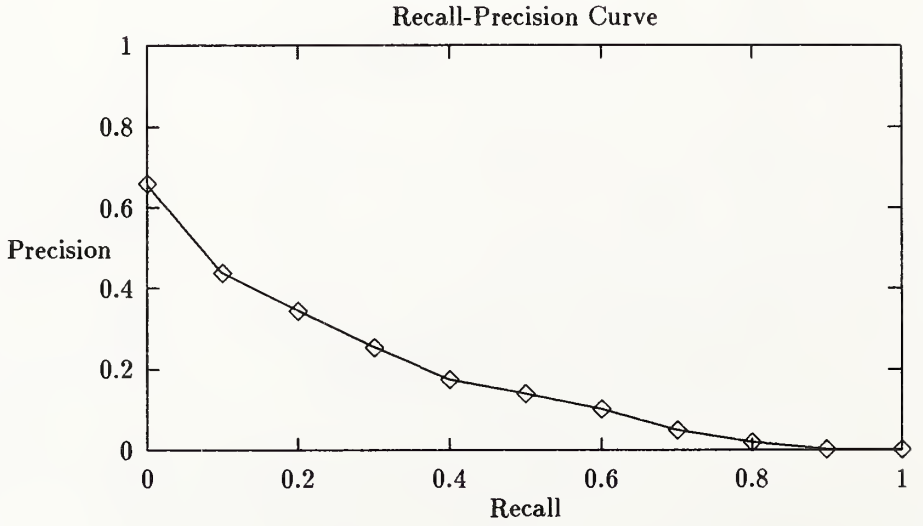



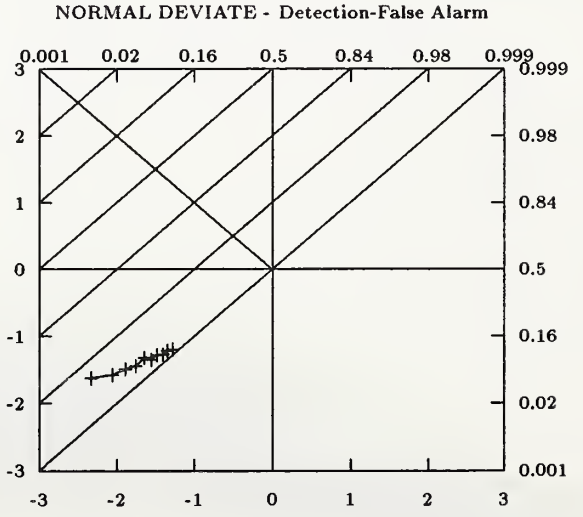
| Summary Statistics | |
|---|---|
| Run number | prceo1– |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 10000 |
| Relevant | 4056 |
| Rel_ret | 1666 |

| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.6598 |
| 0.10 | 0.4382 |
| 0.20 | 0.3451 |
| 0.30 | 0.2536 |
| 0.40 | 0.1753 |
| 0.50 | 0.1400 |
| 0.60 | 0.1016 |
| 0.70 | 0.0489 |
| 0.80 | 0.0185 |
| 0.90 | 0.0015 |
| 1.00 | 0.0015 |
| **Average precision for all points** | |
| 11-pt Avg | 0.1985 |
| **Average precision for 3 intermediate points (0.20, 0.50, 0.80)** | |
| 3-pt Avg | 0.1679 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0379 | 0.4040 |
| at 15 docs | 0.1193 | 0.3653 |
| at 30 docs | 0.1714 | 0.3033 |
| at 100 docs | 0.3438 | 0.2322 |
| at 200 docs | 0.4557 | 0.1666 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0388 |
| 0.00001 | 0.0534 |
| 0.00002 | 0.0582 |
| 0.00003 | 0.0690 |
| 0.00004 | 0.0753 |
| 0.00005 | 0.0943 |
| 0.00006 | 0.0901 |
| 0.00007 | 0.1012 |
| 0.00008 | 0.1033 |
| 0.00009 | 0.1143 |



| Summary Statistics | |
|---|---|
| Run number | siems1–full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 9992 |
| Relevant | 16400 |
| Rel_ret | 3393 |

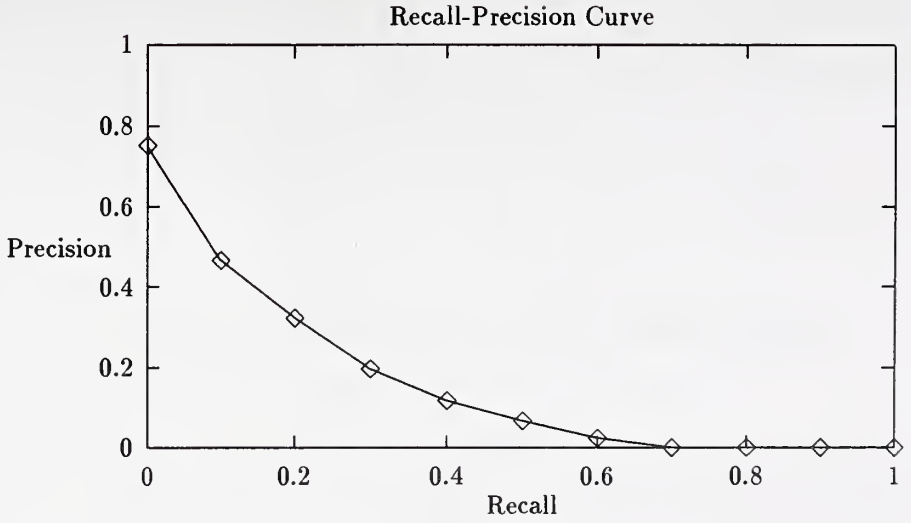
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7524 |
| 0.10 | 0.4670 |
| 0.20 | 0.3242 |
| 0.30 | 0.1973 |
| 0.40 | 0.1179 |
| 0.50 | 0.0672 |
| 0.60 | 0.0241 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

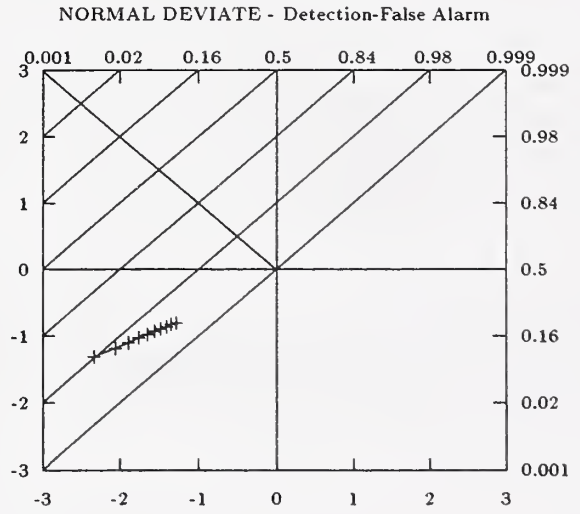
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1773 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1305 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0131 | 0.5200 |
| at 15 docs | 0.0344 | 0.4827 |
| at 30 docs | 0.0650 | 0.4687 |
| at 100 docs | 0.1687 | 0.4100 |
| at 200 docs | 0.2564 | 0.3393 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0660 |
| 0.00001 | 0.0959 |
| 0.00002 | 0.1194 |
| 0.00003 | 0.1369 |
| 0.00004 | 0.1558 |
| 0.00005 | 0.1675 |
| 0.00006 | 0.1775 |
| 0.00007 | 0.1897 |
| 0.00008 | 0.1971 |
| 0.00009 | 0.2065 |



| Summary Statistics | |
|---|---|
| Run number | cityr1–full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 10000 |
| Relevant | 15459 |
| Rel_ret | 3286 |

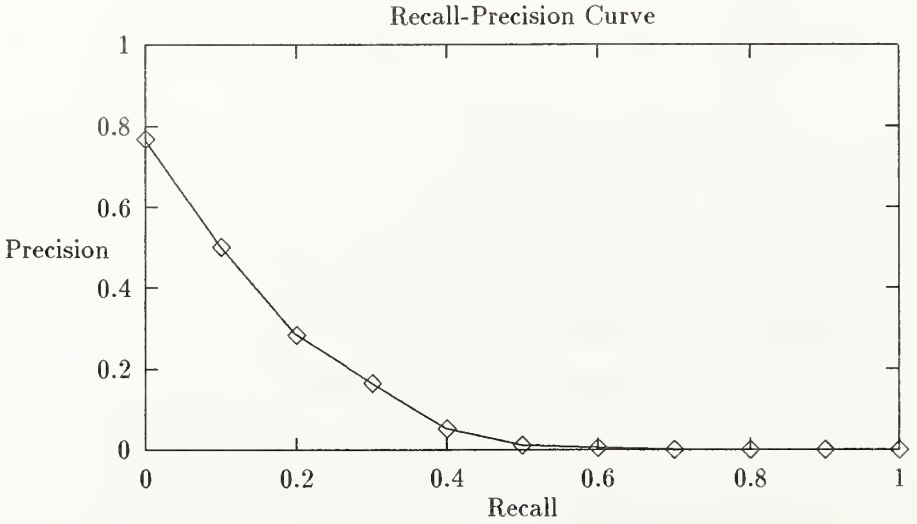
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7676 |
| 0.10 | 0.5016 |
| 0.20 | 0.2853 |
| 0.30 | 0.1646 |
| 0.40 | 0.0522 |
| 0.50 | 0.0105 |
| 0.60 | 0.0045 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

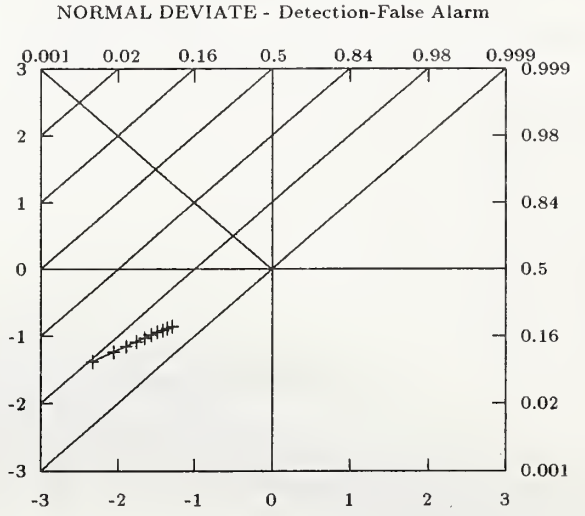
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1624 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0986 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0113 | 0.5480 |
| at 15 docs | 0.0352 | 0.5453 |
| at 30 docs | 0.0646 | 0.5093 |
| at 100 docs | 0.1653 | 0.4376 |
| at 200 docs | 0.2391 | 0.3286 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0607 |
| 0.00001 | 0.0857 |
| 0.00002 | 0.1103 |
| 0.00003 | 0.1268 |
| 0.00004 | 0.1422 |
| 0.00005 | 0.1547 |
| 0.00006 | 0.1655 |
| 0.00007 | 0.1752 |
| 0.00008 | 0.1834 |
| 0.00009 | 0.1911 |



| Summary Statistics | |
|---|---|
| Run number | upitt3–full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 7611 |
| Relevant | 15459 |
| Rel_ret | 1277 |

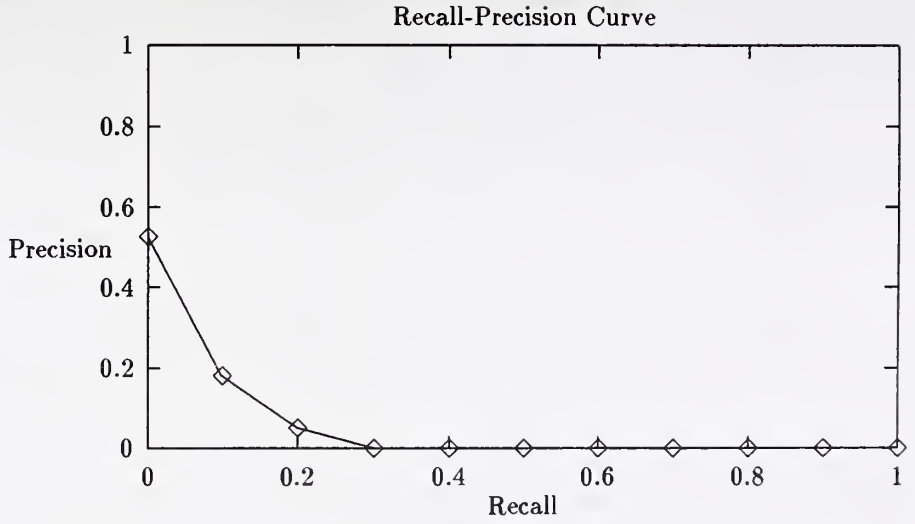
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.5285 |
| 0.10 | 0.1806 |
| 0.20 | 0.0511 |
| 0.30 | 0.0000 |
| 0.40 | 0.0000 |
| 0.50 | 0.0000 |
| 0.60 | 0.0000 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

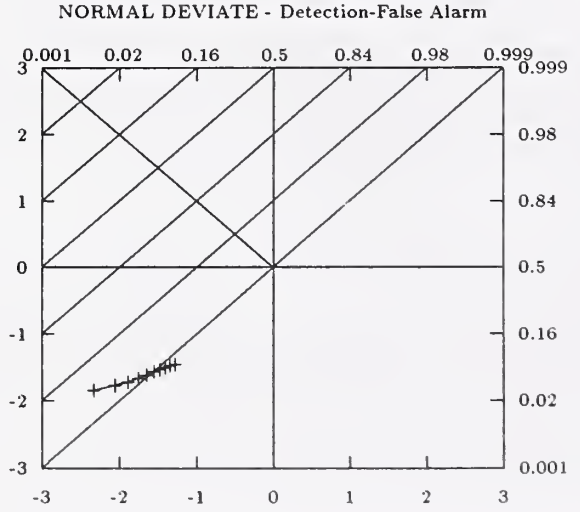
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.0691 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0170 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0074 | 0.3520 |
| at 15 docs | 0.0179 | 0.3013 |
| at 30 docs | 0.0307 | 0.2727 |
| at 100 docs | 0.0737 | 0.2102 |
| at 200 docs | 0.0889 | 0.1277 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0261 |
| 0.00001 | 0.0331 |
| 0.00002 | 0.0392 |
| 0.00003 | 0.0436 |
| 0.00004 | 0.0492 |
| 0.00005 | 0.0540 |
| 0.00006 | 0.0600 |
| 0.00007 | 0.0635 |
| 0.00008 | 0.0673 |
| 0.00009 | 0.0710 |



| Summary Statistics | |
|---|---|
| Run number | crnla2–full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 10000 |
| Relevant | 15459 |
| Rel_ret | 3399 |

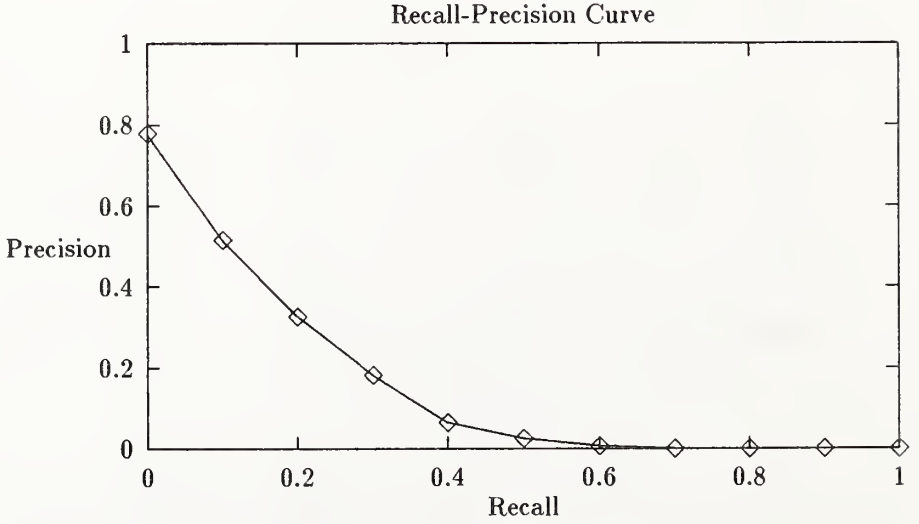
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7798 |
| 0.10 | 0.5162 |
| 0.20 | 0.3278 |
| 0.30 | 0.1823 |
| 0.40 | 0.0647 |
| 0.50 | 0.0248 |
| 0.60 | 0.0066 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

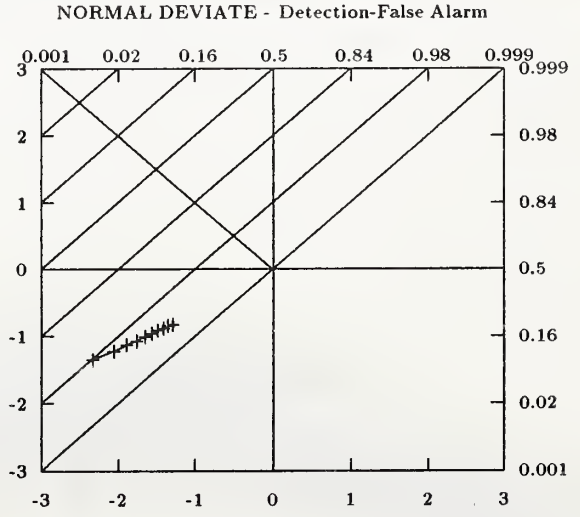
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1729 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1175 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0116 | 0.5640 |
| at 15 docs | 0.0349 | 0.5333 |
| at 30 docs | 0.0646 | 0.5113 |
| at 100 docs | 0.1690 | 0.4452 |
| at 200 docs | 0.2491 | 0.3399 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0679 |
| 0.00001 | 0.0901 |
| 0.00002 | 0.1120 |
| 0.00003 | 0.1313 |
| 0.00004 | 0.1447 |
| 0.00005 | 0.1583 |
| 0.00006 | 0.1712 |
| 0.00007 | 0.1828 |
| 0.00008 | 0.1926 |
| 0.00009 | 0.2018 |



| Summary Statistics | |
|---|---|
| Run number | fuhra2–full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 10000 |
| Relevant | 15459 |
| Rel_ret | 4221 |

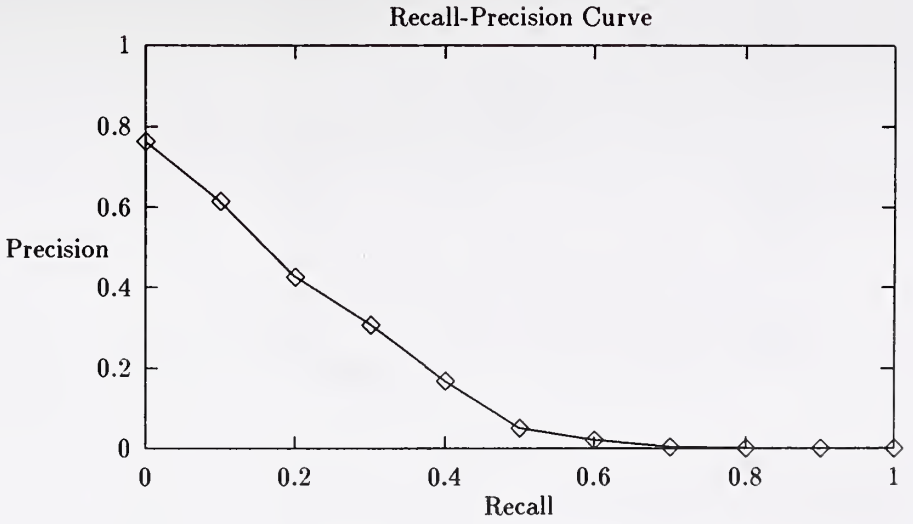
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7630 |
| 0.10 | 0.6153 |
| 0.20 | 0.4279 |
| 0.30 | 0.3089 |
| 0.40 | 0.1686 |
| 0.50 | 0.0516 |
| 0.60 | 0.0203 |
| 0.70 | 0.0036 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

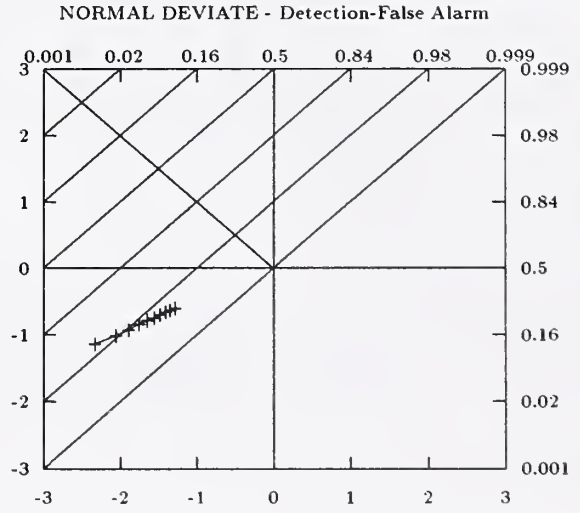
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.2145 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1598 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0124 | 0.6040 |
| at 15 docs | 0.0378 | 0.6000 |
| at 30 docs | 0.0760 | 0.5947 |
| at 100 docs | 0.2088 | 0.5328 |
| at 200 docs | 0.3121 | 0.4221 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0926 |
| 0.00001 | 0.1286 |
| 0.00002 | 0.1573 |
| 0.00003 | 0.1796 |
| 0.00004 | 0.2013 |
| 0.00005 | 0.2174 |
| 0.00006 | 0.2300 |
| 0.00007 | 0.2457 |
| 0.00008 | 0.2563 |
| 0.00009 | 0.2660 |



| Summary Statistics | |
|---|---|
| Run number | danyu1–full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 9999 |
| Relevant | 15459 |
| Rel_ret | 1253 |

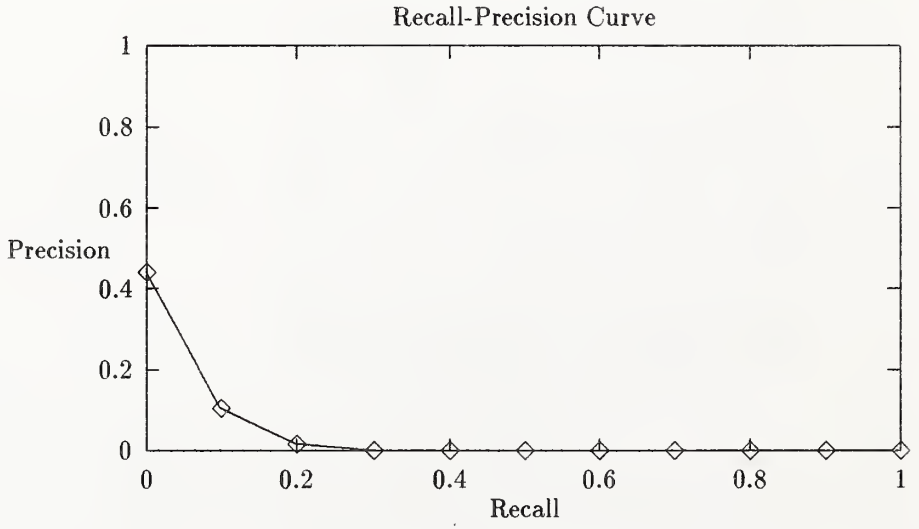
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.4415 |
| 0.10 | 0.1044 |
| 0.20 | 0.0154 |
| 0.30 | 0.0000 |
| 0.40 | 0.0000 |
| 0.50 | 0.0000 |
| 0.60 | 0.0000 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

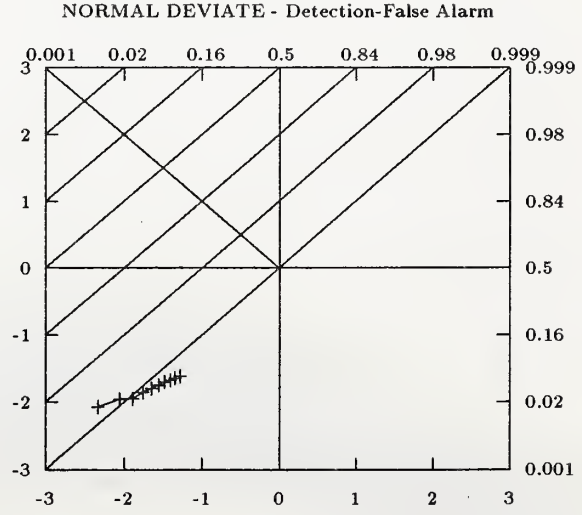
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.0510 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0051 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0041 | 0.2480 |
| at 15 docs | 0.0111 | 0.2240 |
| at 30 docs | 0.0192 | 0.1907 |
| at 100 docs | 0.0500 | 0.1536 |
| at 200 docs | 0.0804 | 0.1253 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0116 |
| 0.00001 | 0.0195 |
| 0.00002 | 0.0256 |
| 0.00003 | 0.0264 |
| 0.00004 | 0.0322 |
| 0.00005 | 0.0371 |
| 0.00006 | 0.0406 |
| 0.00007 | 0.0451 |
| 0.00008 | 0.0482 |
| 0.00009 | 0.0507 |



| Summary Statistics | |
|---|---|
| Run number | lsirc01–full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 10000 |
| Relevant | 15459 |
| Rel_ret | 2234 |

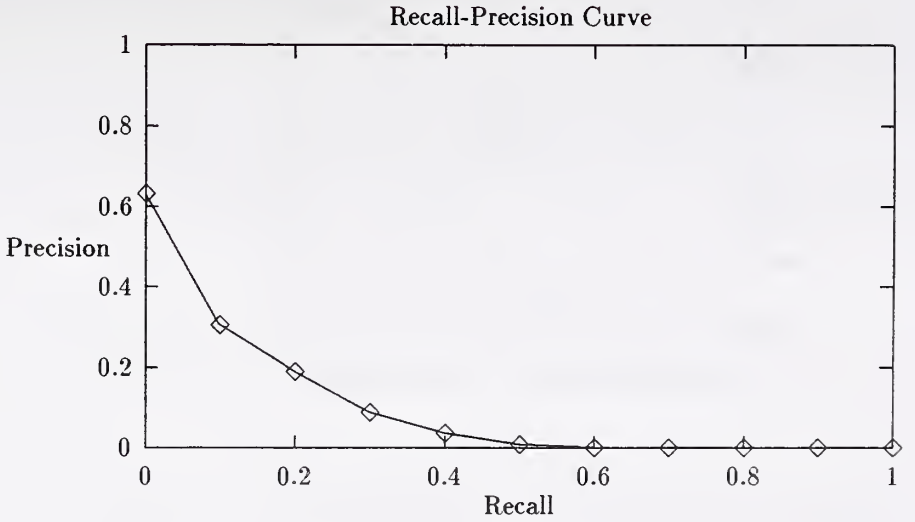
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.6323 |
| 0.10 | 0.3061 |
| 0.20 | 0.1901 |
| 0.30 | 0.0883 |
| 0.40 | 0.0370 |
| 0.50 | 0.0077 |
| 0.60 | 0.0000 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

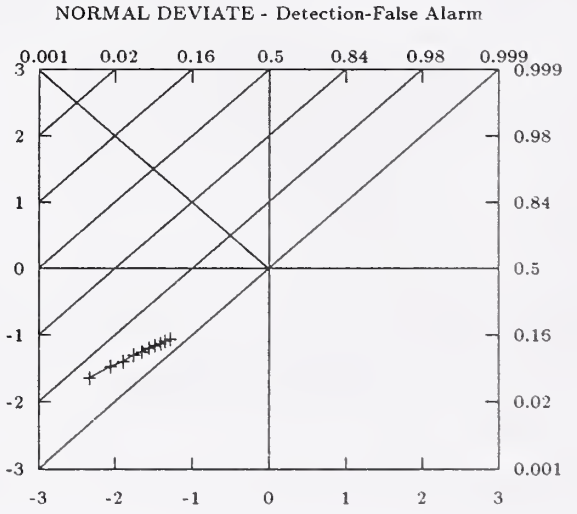
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1147 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0659 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0106 | 0.4160 |
| at 15 docs | 0.0286 | 0.3893 |
| at 30 docs | 0.0506 | 0.3573 |
| at 100 docs | 0.1273 | 0.3024 |
| at 200 docs | 0.1847 | 0.2234 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0418 |
| 0.00001 | 0.0513 |
| 0.00002 | 0.0719 |
| 0.00003 | 0.0828 |
| 0.00004 | 0.0977 |
| 0.00005 | 0.1069 |
| 0.00006 | 0.1170 |
| 0.00007 | 0.1259 |
| 0.00008 | 0.1312 |
| 0.00009 | 0.1389 |



| Summary Statistics | |
|---|---|
| Run number | lsirc02–full, automatic |
| Num_queries | 50 |

| Total number of documents over all queries | |
|---|---|
| Retrieved | 10000 |
| Relevant | 15459 |
| Rel_ret | 1837 |

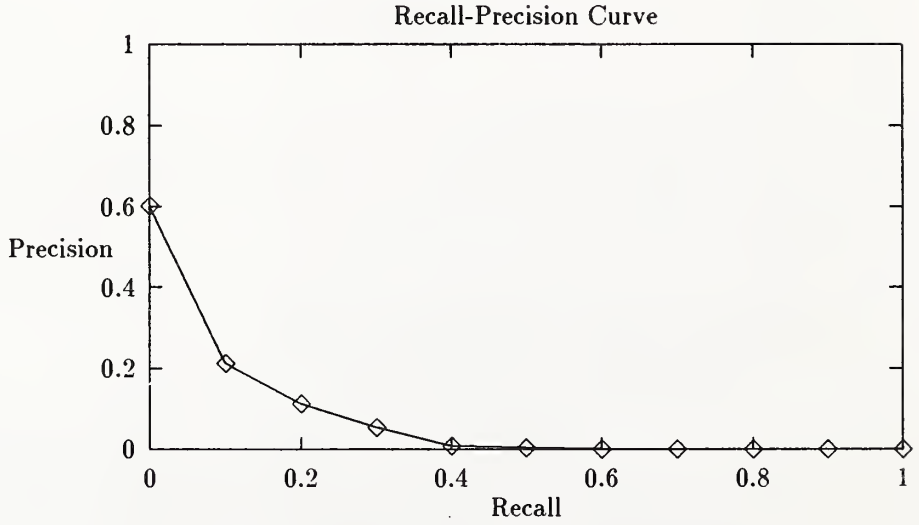
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.6033 |
| 0.10 | 0.2134 |
| 0.20 | 0.1125 |
| 0.30 | 0.0533 |
| 0.40 | 0.0072 |
| 0.50 | 0.0039 |
| 0.60 | 0.0000 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

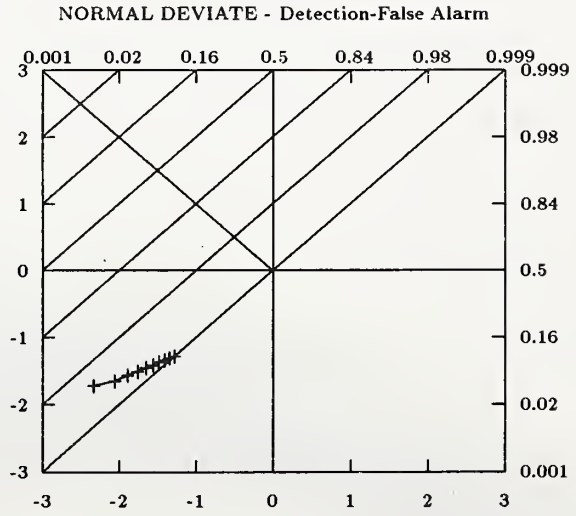
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.0903 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0388 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0080 | 0.3920 |
| at 15 docs | 0.0213 | 0.3360 |
| at 30 docs | 0.0399 | 0.3067 |
| at 100 docs | 0.0910 | 0.2448 |
| at 200 docs | 0.1388 | 0.1837 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0279 |
| 0.00001 | 0.0437 |
| 0.00002 | 0.0504 |
| 0.00003 | 0.0599 |
| 0.00004 | 0.0668 |
| 0.00005 | 0.0739 |
| 0.00006 | 0.0796 |
| 0.00007 | 0.0872 |
| 0.00008 | 0.0919 |
| 0.00009 | 0.0970 |



| Summary Statistics | |
|---|---|
| Run number | pircs1–category B, automatic |
| Num_queries | 25 |
| **Total number of documents over all queries** | |
| Retrieved | 5000 |
| Relevant | 3766 |
| Rel_ret | 1471 |

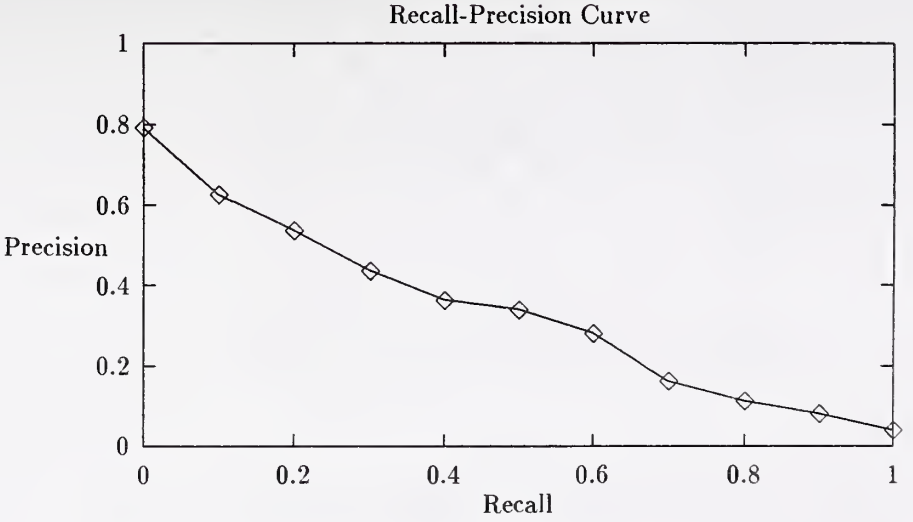
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7918 |
| 0.10 | 0.6256 |
| 0.20 | 0.5378 |
| 0.30 | 0.4373 |
| 0.40 | 0.3648 |
| 0.50 | 0.3395 |
| 0.60 | 0.2814 |
| 0.70 | 0.1627 |
| 0.80 | 0.1132 |
| 0.90 | 0.0804 |
| 1.00 | 0.0391 |

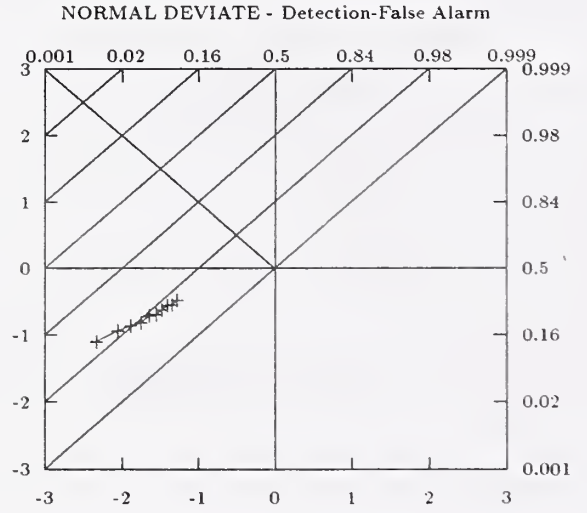
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.3430 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.3302 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0475 | 0.6080 |
| at 15 docs | 0.1448 | 0.5973 |
| at 30 docs | 0.2280 | 0.5213 |
| at 100 docs | 0.4434 | 0.3980 |
| at 200 docs | 0.5764 | 0.2942 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0974 |
| 0.00001 | 0.1370 |
| 0.00002 | 0.1767 |
| 0.00003 | 0.1966 |
| 0.00004 | 0.2096 |
| 0.00005 | 0.2426 |
| 0.00006 | 0.2476 |
| 0.00007 | 0.2694 |
| 0.00008 | 0.2910 |
| 0.00009 | 0.2947 |



| Summary Statistics | |
|---|---|
| Run number | pircs2–category B, manual |
| Num_queries | 25 |
| **Total number of documents over all queries** | |
| Retrieved | 5000 |
| Relevant | 3766 |
| Rel_ret | 1539 |

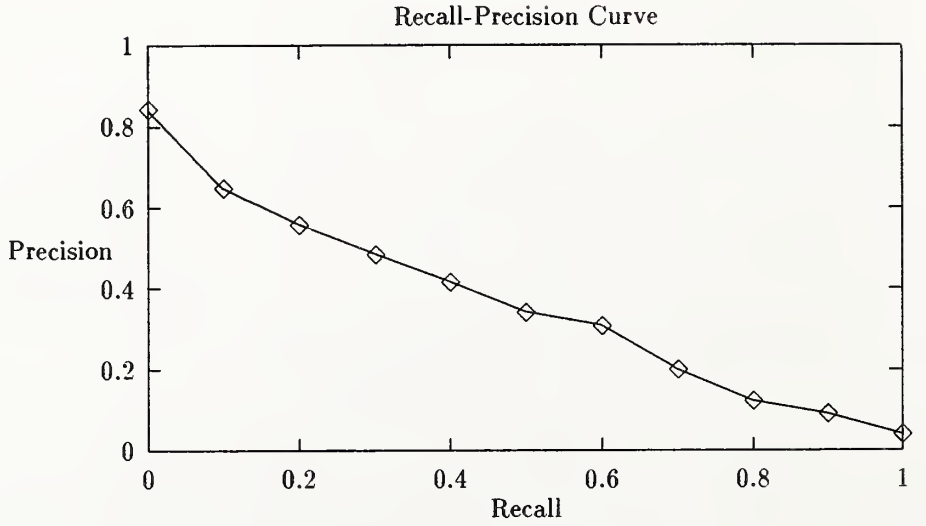
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.8428 |
| 0.10 | 0.6482 |
| 0.20 | 0.5581 |
| 0.30 | 0.4842 |
| 0.40 | 0.4171 |
| 0.50 | 0.3422 |
| 0.60 | 0.3079 |
| 0.70 | 0.2008 |
| 0.80 | 0.1220 |
| 0.90 | 0.0905 |
| 1.00 | 0.0404 |

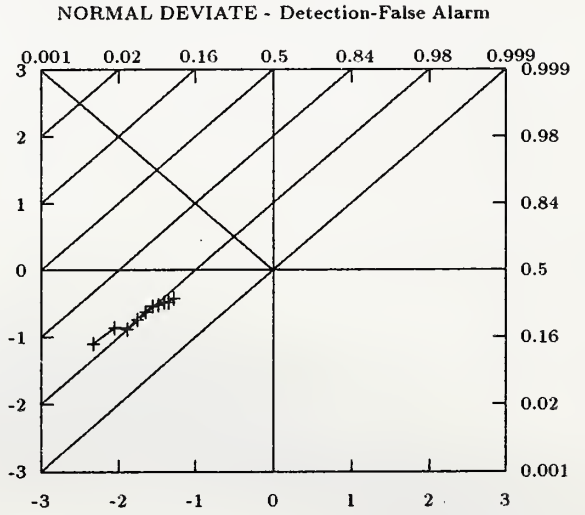
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.3686 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.3408 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0476 | 0.6400 |
| at 15 docs | 0.1408 | 0.6053 |
| at 30 docs | 0.2352 | 0.5400 |
| at 100 docs | 0.4649 | 0.4236 |
| at 200 docs | 0.5878 | 0.3078 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.1106 |
| 0.00001 | 0.1384 |
| 0.00002 | 0.1974 |
| 0.00003 | 0.1914 |
| 0.00004 | 0.2318 |
| 0.00005 | 0.2701 |
| 0.00006 | 0.2985 |
| 0.00007 | 0.3051 |
| 0.00008 | 0.3175 |
| 0.00009 | 0.3184 |



| Summary Statistics | |
|---|---|
| Run number | nyuir1–category B, automatic |
| Num_queries | 25 |
| **Total number of documents over all queries** | |
| Retrieved | 5000 |
| Relevant | 3766 |
| Rel_ret | 834 |

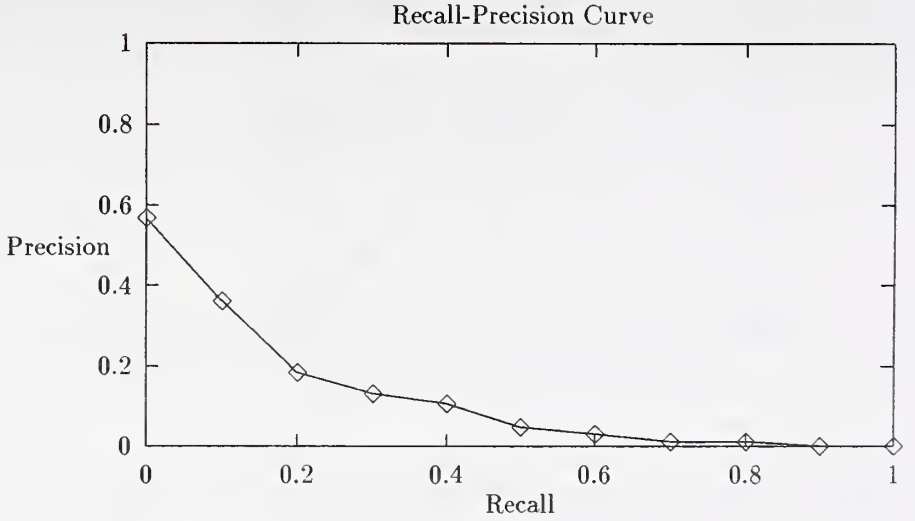
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.5691 |
| 0.10 | 0.3618 |
| 0.20 | 0.1846 |
| 0.30 | 0.1314 |
| 0.40 | 0.1065 |
| 0.50 | 0.0481 |
| 0.60 | 0.0295 |
| 0.70 | 0.0104 |
| 0.80 | 0.0104 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

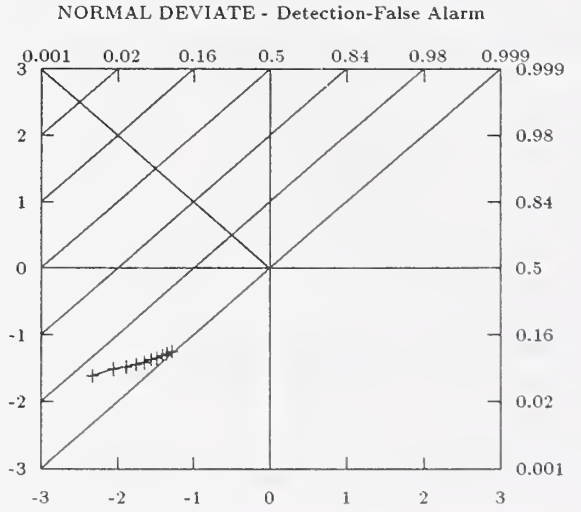
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1320 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0811 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0338 | 0.3360 |
| at 15 docs | 0.0671 | 0.2960 |
| at 30 docs | 0.0979 | 0.2787 |
| at 100 docs | 0.2117 | 0.2276 |
| at 200 docs | 0.2988 | 0.1668 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0397 |
| 0.00001 | 0.0537 |
| 0.00002 | 0.0666 |
| 0.00003 | 0.0698 |
| 0.00004 | 0.0762 |
| 0.00005 | 0.0801 |
| 0.00006 | 0.0872 |
| 0.00007 | 0.0899 |
| 0.00008 | 0.0963 |
| 0.00009 | 0.1008 |



| Summary Statistics | |
|---|---|
| Run number | UCFQA1–category B, automatic |
| Num_queries | 25 |

| Total number of documents over all queries | |
|---|---|
| Retrieved | 5000 |
| Relevant | 3766 |
| Rel_ret | 704 |

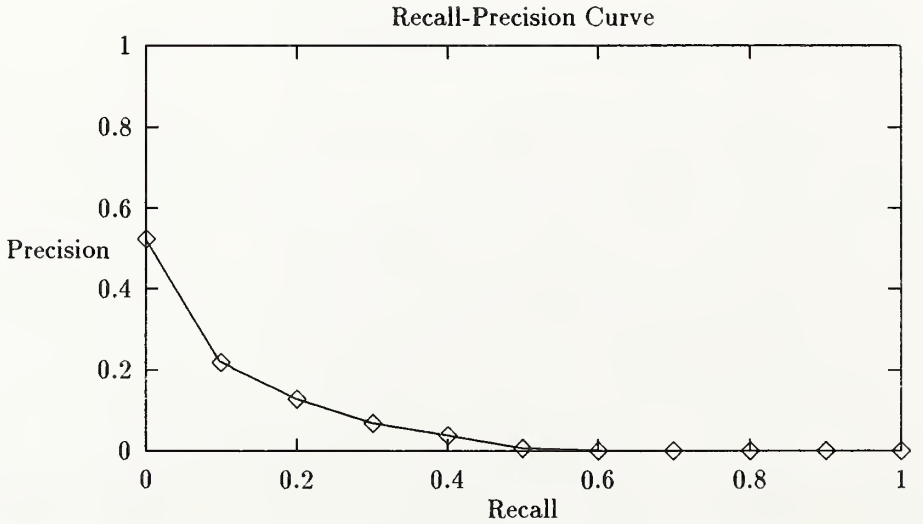
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.5248 |
| 0.10 | 0.2190 |
| 0.20 | 0.1293 |
| 0.30 | 0.0689 |
| 0.40 | 0.0389 |
| 0.50 | 0.0069 |
| 0.60 | 0.0000 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

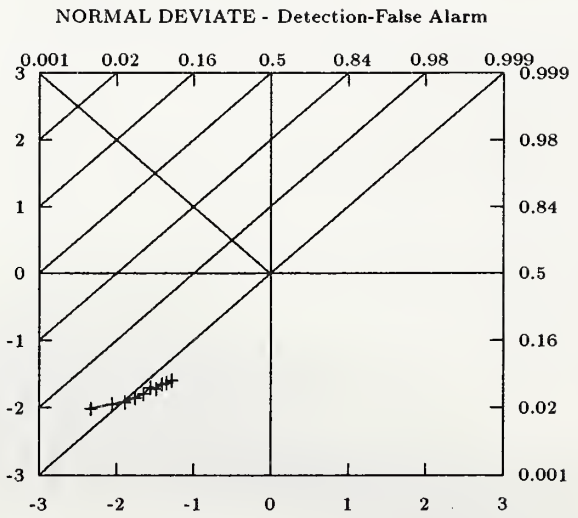
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.0898 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0454 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0116 | 0.2560 |
| at 15 docs | 0.0328 | 0.2080 |
| at 30 docs | 0.0559 | 0.1933 |
| at 100 docs | 0.1661 | 0.1860 |
| at 200 docs | 0.2446 | 0.1408 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0138 |
| 0.00001 | 0.0223 |
| 0.00002 | 0.0261 |
| 0.00003 | 0.0282 |
| 0.00004 | 0.0324 |
| 0.00005 | 0.0367 |
| 0.00006 | 0.0453 |
| 0.00007 | 0.0433 |
| 0.00008 | 0.0500 |
| 0.00009 | 0.0518 |



| Summary Statistics | |
|---|---|
| Run number | adsba1–category B, automatic |
| Num_queries | 25 |
| **Total number of documents over all queries** | |
| Retrieved | 4698 |
| Relevant | 3766 |
| Rel_ret | 572 |

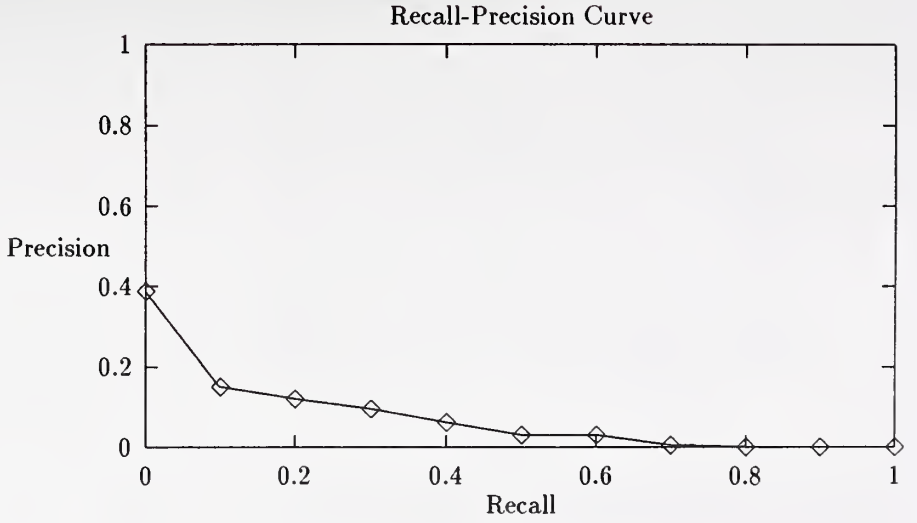
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.3889 |
| 0.10 | 0.1508 |
| 0.20 | 0.1210 |
| 0.30 | 0.0957 |
| 0.40 | 0.0623 |
| 0.50 | 0.0307 |
| 0.60 | 0.0301 |
| 0.70 | 0.0051 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

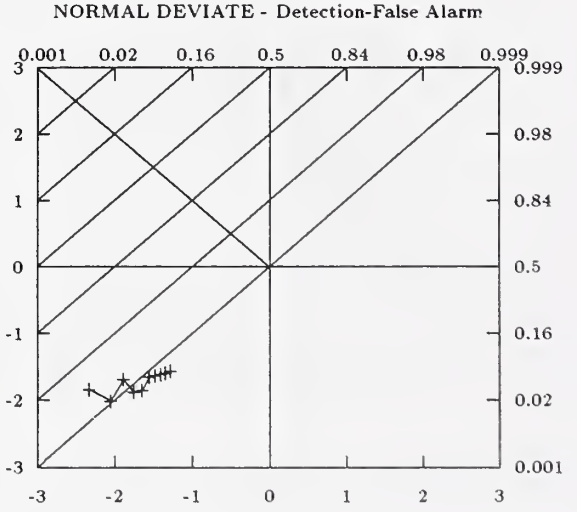
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.0804 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0506 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0118 | 0.1680 |
| at 15 docs | 0.0220 | 0.1467 |
| at 30 docs | 0.0442 | 0.1493 |
| at 100 docs | 0.1137 | 0.1244 |
| at 200 docs | 0.1771 | 0.1144 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0268 |
| 0.00001 | 0.0329 |
| 0.00002 | 0.0225 |
| 0.00003 | 0.0459 |
| 0.00004 | 0.0306 |
| 0.00005 | 0.0324 |
| 0.00006 | 0.0499 |
| 0.00007 | 0.0517 |
| 0.00008 | 0.0545 |
| 0.00009 | 0.0561 |



| Summary Statistics | |
|---|---|
| Run number | adsba2–category B, automatic |
| Num_queries | 25 |

| Total number of documents over all queries | |
|---|---|
| Retrieved | 4712 |
| Relevant | 3766 |
| Rel_ret | 559 |

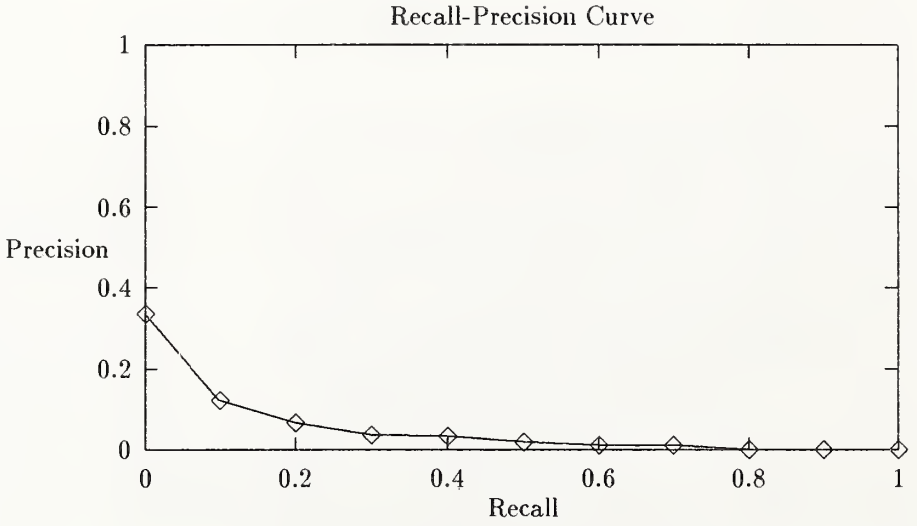
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.3378 |
| 0.10 | 0.1223 |
| 0.20 | 0.0661 |
| 0.30 | 0.0358 |
| 0.40 | 0.0338 |
| 0.50 | 0.0186 |
| 0.60 | 0.0116 |
| 0.70 | 0.0116 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

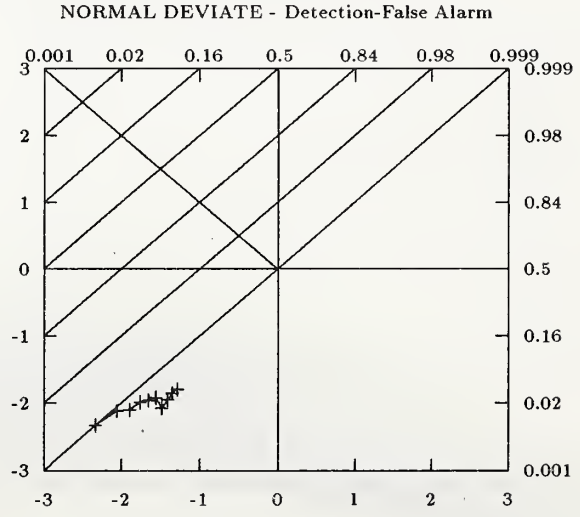
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.0580 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0282 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0075 | 0.1200 |
| at 15 docs | 0.0174 | 0.1173 |
| at 30 docs | 0.0367 | 0.1253 |
| at 100 docs | 0.0931 | 0.1084 |
| at 200 docs | 0.1778 | 0.1118 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0086 |
| 0.00001 | 0.0102 |
| 0.00002 | 0.0174 |
| 0.00003 | 0.0181 |
| 0.00004 | 0.0238 |
| 0.00005 | 0.0258 |
| 0.00006 | 0.0277 |
| 0.00007 | 0.0193 |
| 0.00008 | 0.0261 |
| 0.00009 | 0.0326 |



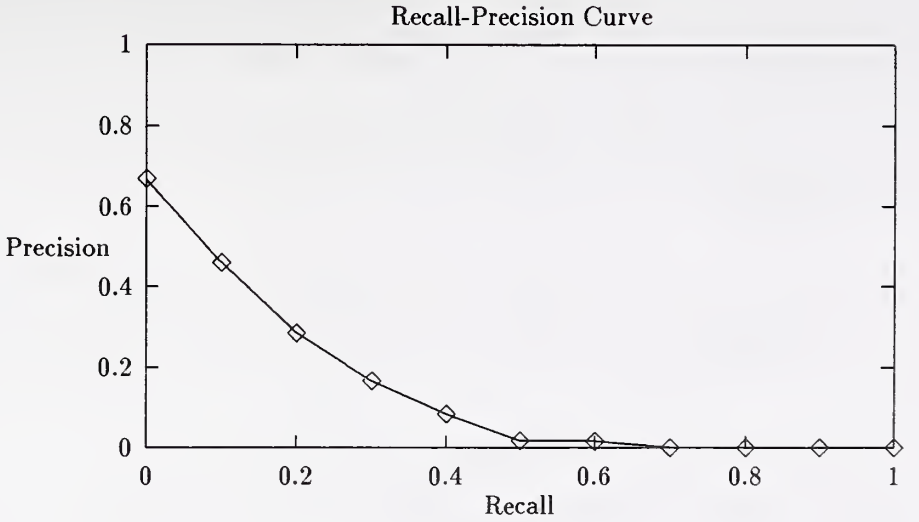



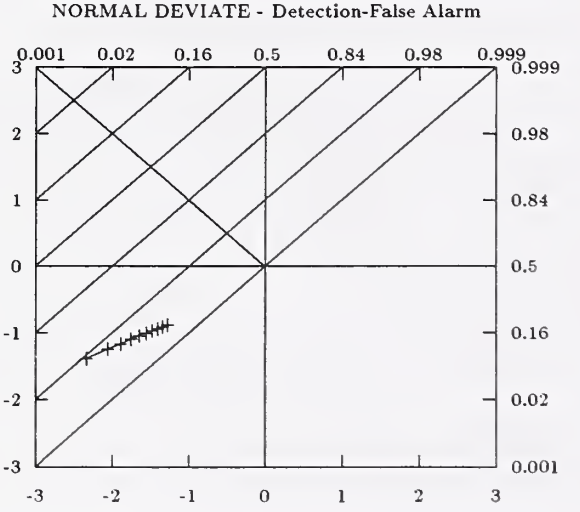
| Summary Statistics | |
|---|---|
| Run number | cpghc2-full, manual |
| Num_queries | 50 |
| Total number of documents over all queries | |
| Retrieved | 8531 |
| Relevant | 15459 |
| Rel_ret | 3168 |

| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.6690 |
| 0.10 | 0.4610 |
| 0.20 | 0.2867 |
| 0.30 | 0.1662 |
| 0.40 | 0.0839 |
| 0.50 | 0.0179 |
| 0.60 | 0.0164 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |
| Average precision for all points | |
| 11-pt Avg | 0.1547 |
| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
| 3-pt Avg | 0.1015 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0093 | 0.4720 |
| at 15 docs | 0.0291 | 0.4787 |
| at 30 docs | 0.0570 | 0.4713 |
| at 100 docs | 0.1474 | 0.4084 |
| at 200 docs | 0.2184 | 0.3168 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0557 |
| 0.00001 | 0.0844 |
| 0.00002 | 0.1084 |
| 0.00003 | 0.1237 |
| 0.00004 | 0.1388 |
| 0.00005 | 0.1495 |
| 0.00006 | 0.1599 |
| 0.00007 | 0.1687 |
| 0.00008 | 0.1768 |
| 0.00009 | 0.1844 |



| Summary Statistics | |
|---|---|
| Run number | cpgcn2–full, automatic |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 9468 |
| Relevant | 15459 |
| Rel_ret | 2907 |

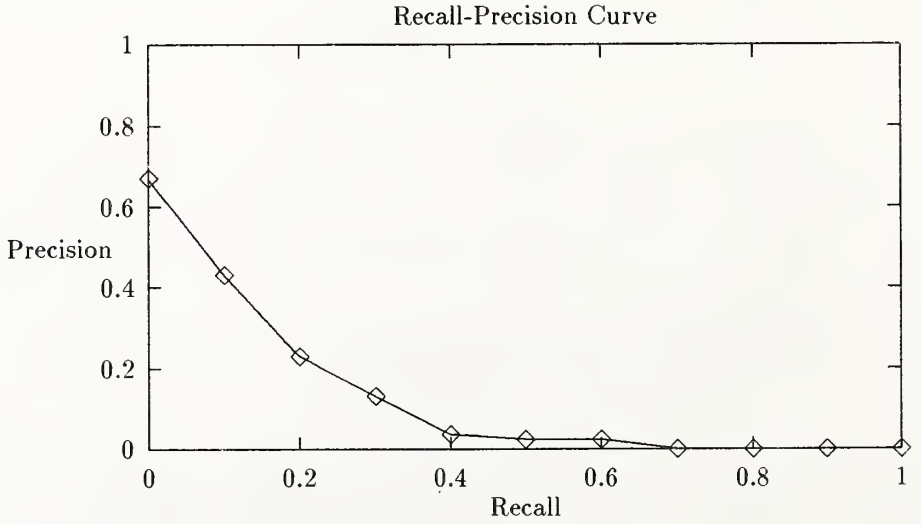
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.6706 |
| 0.10 | 0.4330 |
| 0.20 | 0.2312 |
| 0.30 | 0.1320 |
| 0.40 | 0.0371 |
| 0.50 | 0.0242 |
| 0.60 | 0.0237 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

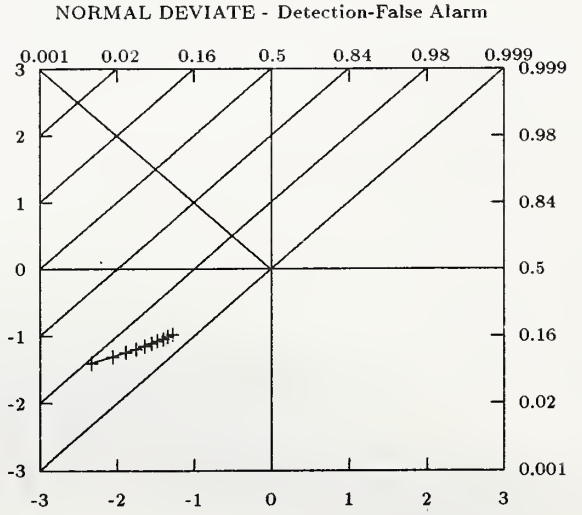
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1411 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0852 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0125 | 0.4600 |
| at 15 docs | 0.0330 | 0.4680 |
| at 30 docs | 0.0537 | 0.4260 |
| at 100 docs | 0.1389 | 0.3750 |
| at 200 docs | 0.2107 | 0.2907 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0660 |
| 0.00001 | 0.0808 |
| 0.00002 | 0.0964 |
| 0.00003 | 0.1080 |
| 0.00004 | 0.1176 |
| 0.00005 | 0.1273 |
| 0.00006 | 0.1347 |
| 0.00007 | 0.1438 |
| 0.00008 | 0.1512 |
| 0.00009 | 0.1602 |



| Summary Statistics | |
|---|---|
| Run number | clarta–full, manual |
| Num_queries | 50 |

| Total number of documents over all queries | |
|---|---|
| Retrieved | 10000 |
| Relevant | 15459 |
| Rel_ret | 2139 |

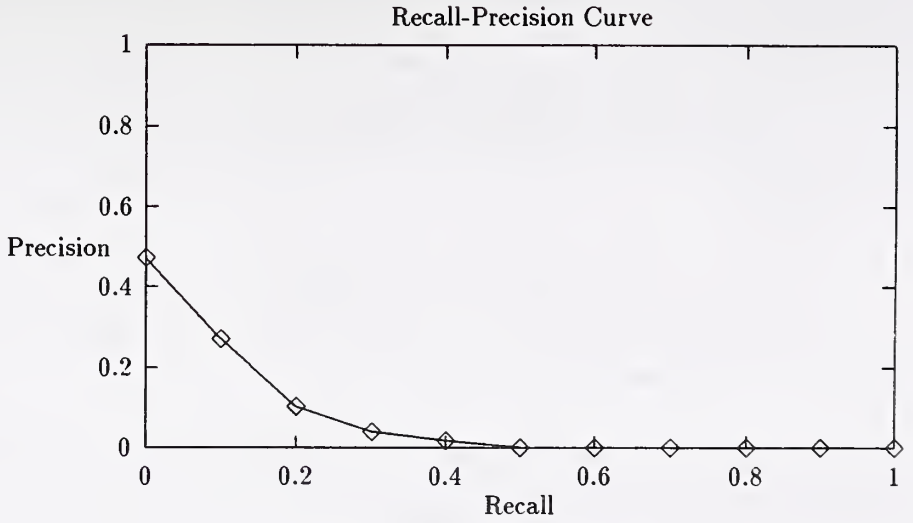
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.4736 |
| 0.10 | 0.2712 |
| 0.20 | 0.1033 |
| 0.30 | 0.0394 |
| 0.40 | 0.0170 |
| 0.50 | 0.0000 |
| 0.60 | 0.0000 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

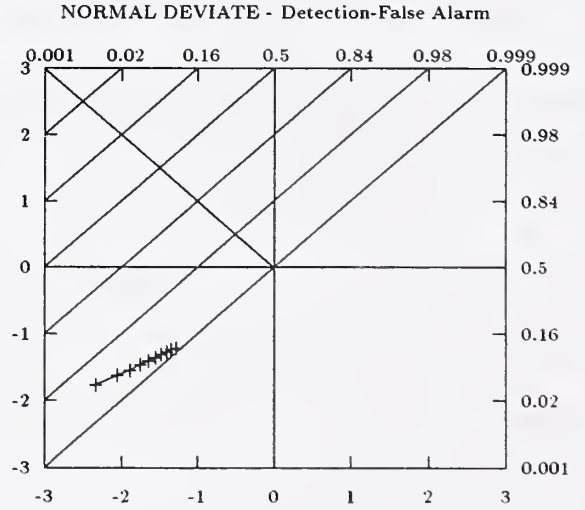
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.0822 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.0344 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0059 | 0.2560 |
| at 15 docs | 0.0205 | 0.3120 |
| at 30 docs | 0.0387 | 0.3173 |
| at 100 docs | 0.1002 | 0.2756 |
| at 200 docs | 0.1499 | 0.2139 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0267 |
| 0.00001 | 0.0391 |
| 0.00002 | 0.0531 |
| 0.00003 | 0.0620 |
| 0.00004 | 0.0733 |
| 0.00005 | 0.0809 |
| 0.00006 | 0.0883 |
| 0.00007 | 0.0969 |
| 0.00008 | 0.1036 |
| 0.00009 | 0.1095 |



| Summary Statistics | |
|---|---|
| Run number | clartb-full, manual |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 10000 |
| Relevant | 15459 |
| Rel_ret | 3427 |

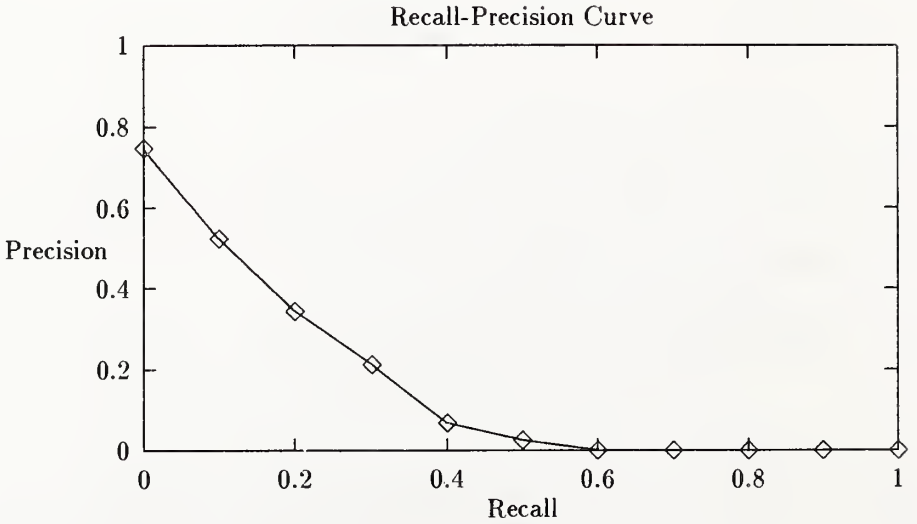
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7479 |
| 0.10 | 0.5245 |
| 0.20 | 0.3443 |
| 0.30 | 0.2125 |
| 0.40 | 0.0680 |
| 0.50 | 0.0259 |
| 0.60 | 0.0000 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

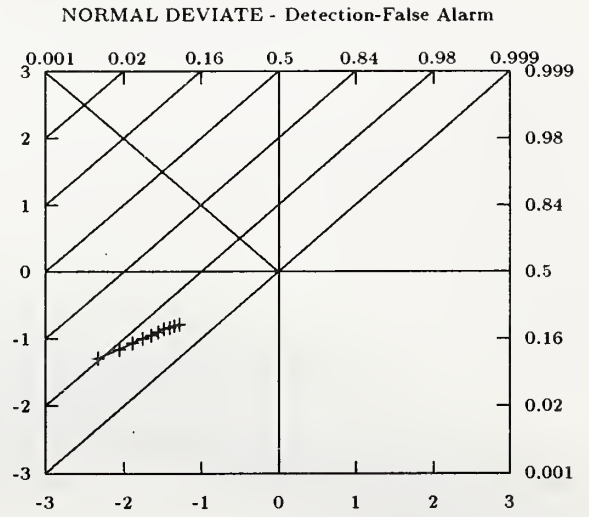
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1748 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1234 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0110 | 0.5120 |
| at 15 docs | 0.0360 | 0.5320 |
| at 30 docs | 0.0675 | 0.5140 |
| at 100 docs | 0.1778 | 0.4560 |
| at 200 docs | 0.2521 | 0.3427 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0717 |
| 0.00001 | 0.0990 |
| 0.00002 | 0.1251 |
| 0.00003 | 0.1450 |
| 0.00004 | 0.1617 |
| 0.00005 | 0.1747 |
| 0.00006 | 0.1863 |
| 0.00007 | 0.1981 |
| 0.00008 | 0.2052 |
| 0.00009 | 0.2132 |



| Summary Statistics | |
|---|---|
| Run number | gecrd1-full, manual |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 6911 |
| Relevant | 15459 |
| Rel_ret | 3308 |

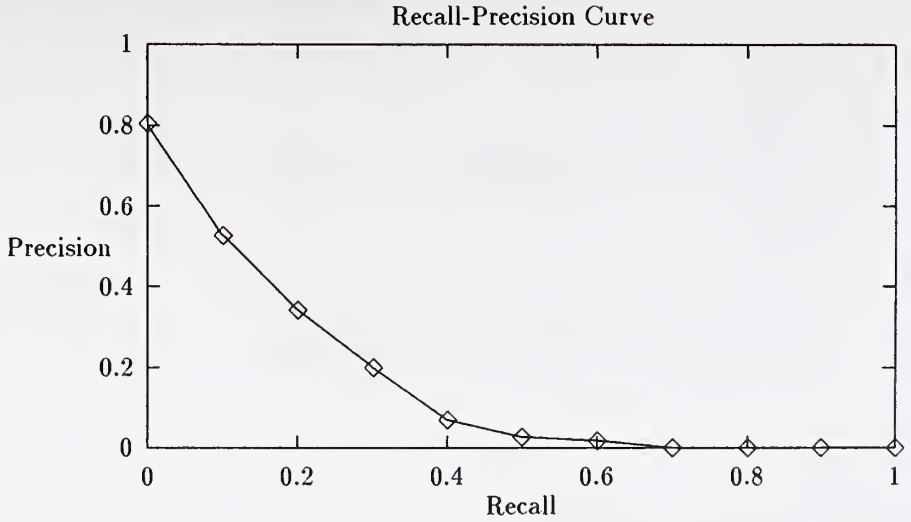
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.8044 |
| 0.10 | 0.5286 |
| 0.20 | 0.3427 |
| 0.30 | 0.2006 |
| 0.40 | 0.0703 |
| 0.50 | 0.0275 |
| 0.60 | 0.0171 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

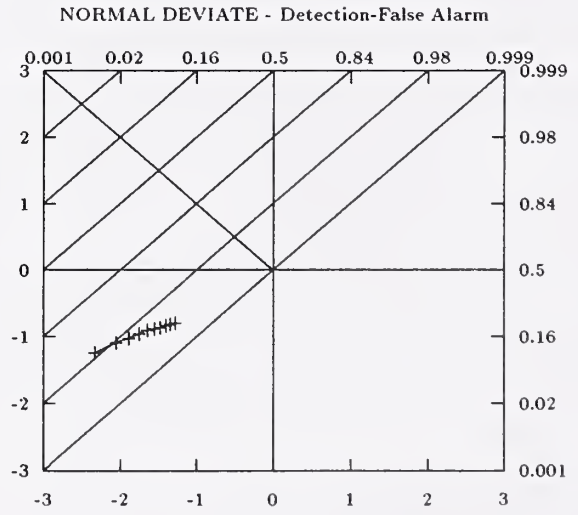
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1810 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1234 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0133 | 0.6080 |
| at 15 docs | 0.0359 | 0.5840 |
| at 30 docs | 0.0720 | 0.5747 |
| at 100 docs | 0.1713 | 0.4486 |
| at 200 docs | 0.2377 | 0.3308 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0674 |
| 0.00001 | 0.1082 |
| 0.00002 | 0.1394 |
| 0.00003 | 0.1558 |
| 0.00004 | 0.1720 |
| 0.00005 | 0.1859 |
| 0.00006 | 0.1912 |
| 0.00007 | 0.1976 |
| 0.00008 | 0.2040 |
| 0.00009 | 0.2105 |



| Summary Statistics | |
|---|---|
| Run number | gecrd2–full, manual |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 7820 |
| Relevant | 15459 |
| Rel_ret | 3581 |

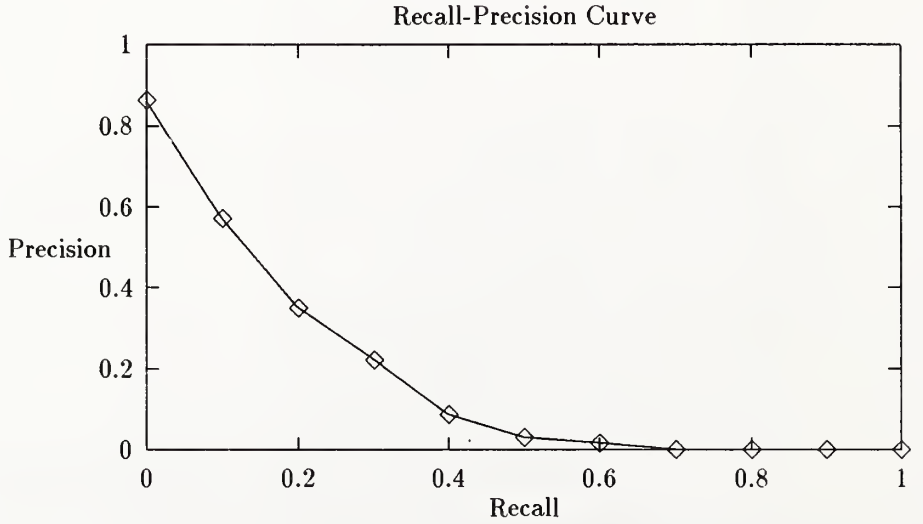
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.8637 |
| 0.10 | 0.5725 |
| 0.20 | 0.3519 |
| 0.30 | 0.2218 |
| 0.40 | 0.0869 |
| 0.50 | 0.0299 |
| 0.60 | 0.0157 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

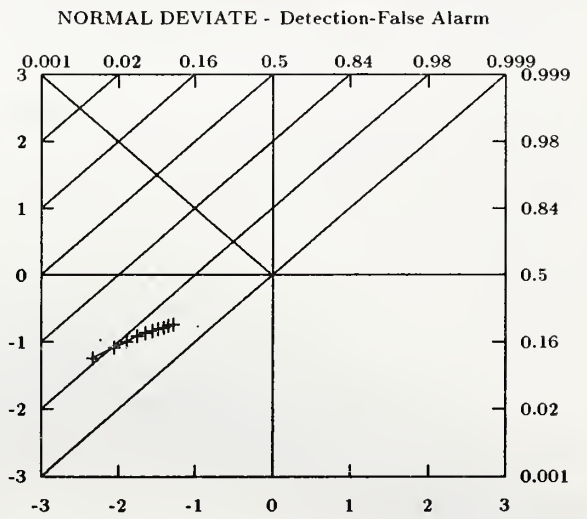
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1948 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1273 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0161 | 0.6880 |
| at 15 docs | 0.0417 | 0.6387 |
| at 30 docs | 0.0752 | 0.5953 |
| at 100 docs | 0.1832 | 0.4672 |
| at 200 docs | 0.2633 | 0.3581 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0773 |
| 0.00001 | 0.1087 |
| 0.00002 | 0.1414 |
| 0.00003 | 0.1629 |
| 0.00004 | 0.1847 |
| 0.00005 | 0.1966 |
| 0.00006 | 0.2044 |
| 0.00007 | 0.2129 |
| 0.00008 | 0.2174 |
| 0.00009 | 0.2262 |



| Summary Statistics | |
|---|---|
| Run number | trw1–full, manual |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 6830 |
| Relevant | 15459 |
| Rel_ret | 3206 |

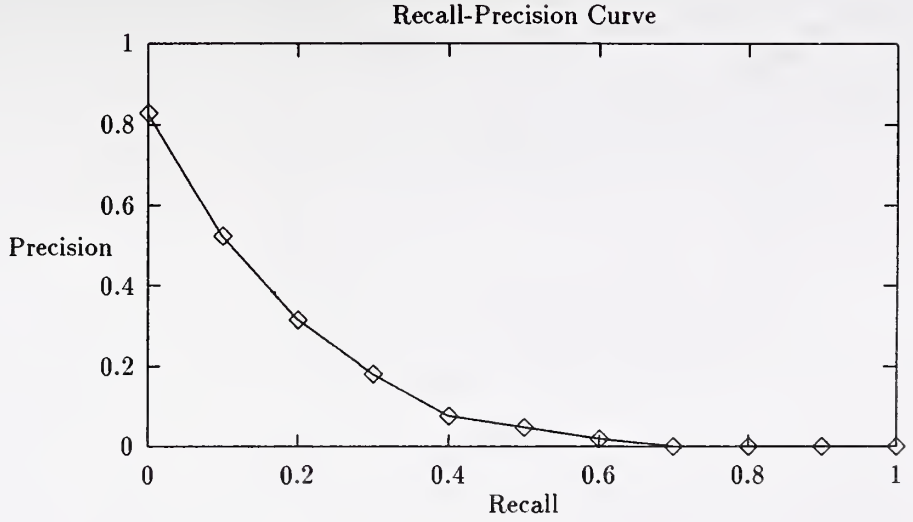
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.8281 |
| 0.10 | 0.5248 |
| 0.20 | 0.3157 |
| 0.30 | 0.1816 |
| 0.40 | 0.0764 |
| 0.50 | 0.0474 |
| 0.60 | 0.0198 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

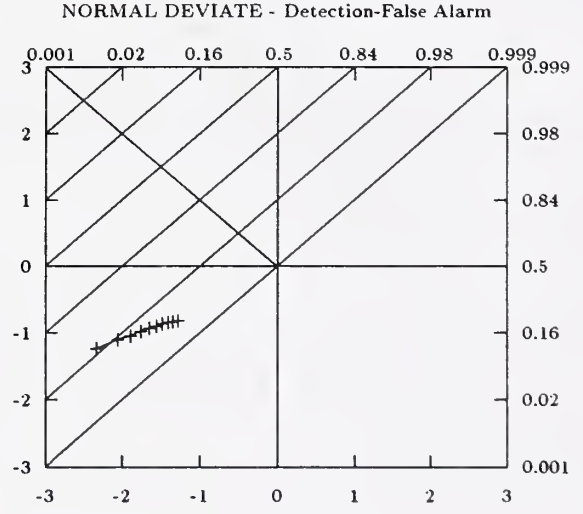
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1813 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1210 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0174 | 0.6160 |
| at 15 docs | 0.0388 | 0.5547 |
| at 30 docs | 0.0720 | 0.5453 |
| at 100 docs | 0.1678 | 0.4294 |
| at 200 docs | 0.2332 | 0.3206 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0762 |
| 0.00001 | 0.1095 |
| 0.00002 | 0.1364 |
| 0.00003 | 0.1504 |
| 0.00004 | 0.1661 |
| 0.00005 | 0.1783 |
| 0.00006 | 0.1863 |
| 0.00007 | 0.1960 |
| 0.00008 | 0.2010 |
| 0.00009 | 0.2045 |



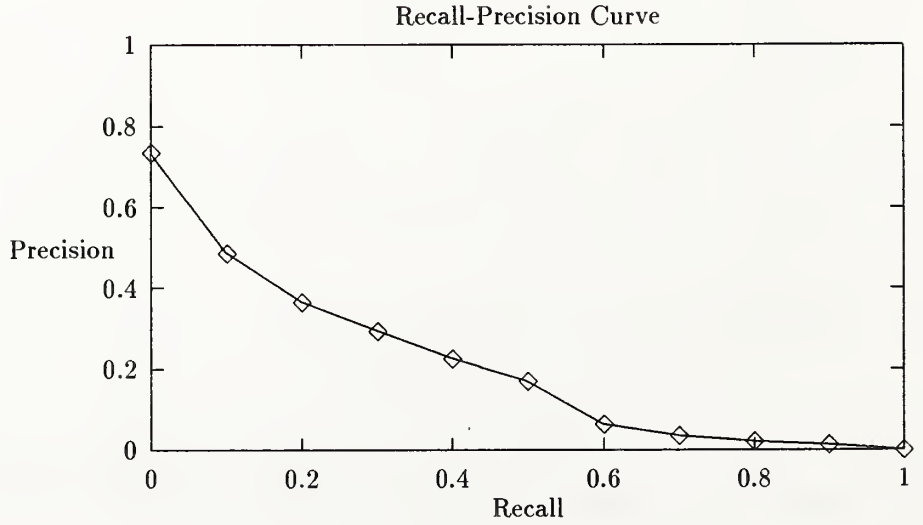



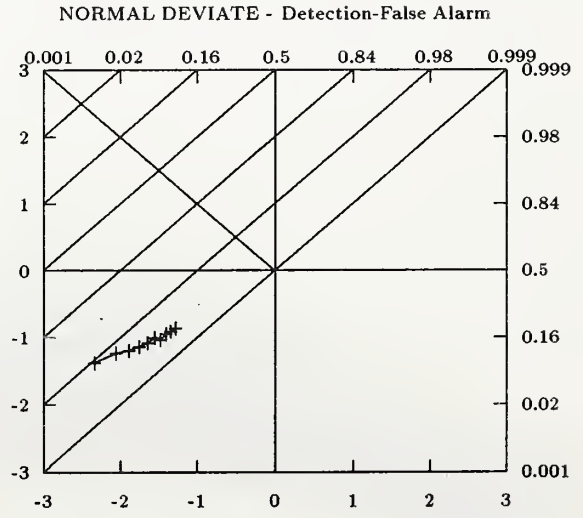
| Summary Statistics | |
|---|---|
| Run number | fairs1–category B, automatic |
| Num_queries | 25 |
| **Total number of documents over all queries** | |
| Retrieved | 5000 |
| Relevant | 3766 |
| Rel_ret | 1124 |

| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7339 |
| 0.10 | 0.4871 |
| 0.20 | 0.3661 |
| 0.30 | 0.2942 |
| 0.40 | 0.2256 |
| 0.50 | 0.1707 |
| 0.60 | 0.0639 |
| 0.70 | 0.0352 |
| 0.80 | 0.0208 |
| 0.90 | 0.0128 |
| 1.00 | 0.0000 |
| **Average precision for all points** | |
| 11-pt Avg | 0.2191 |
| **Average precision for 3 intermediate points (0.20, 0.50, 0.80)** | |
| 3-pt Avg | 0.1858 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0370 | 0.4560 |
| at 15 docs | 0.0918 | 0.4480 |
| at 30 docs | 0.1514 | 0.4200 |
| at 100 docs | 0.3166 | 0.3136 |
| at 200 docs | 0.4206 | 0.2248 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0641 |
| 0.00001 | 0.0853 |
| 0.00002 | 0.1094 |
| 0.00003 | 0.1166 |
| 0.00004 | 0.1299 |
| 0.00005 | 0.1419 |
| 0.00006 | 0.1599 |
| 0.00007 | 0.1532 |
| 0.00008 | 0.1732 |
| 0.00009 | 0.1834 |



| Summary Statistics | |
|---|---|
| Run number | paraz1–full, manual |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 8823 |
| Relevant | 15459 |
| Rel_ret | 3503 |

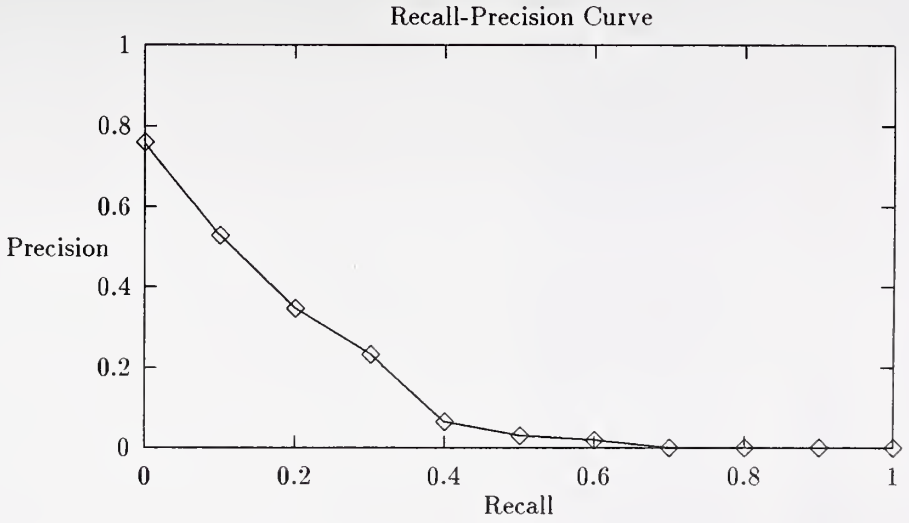
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.7604 |
| 0.10 | 0.5288 |
| 0.20 | 0.3483 |
| 0.30 | 0.2336 |
| 0.40 | 0.0647 |
| 0.50 | 0.0302 |
| 0.60 | 0.0187 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

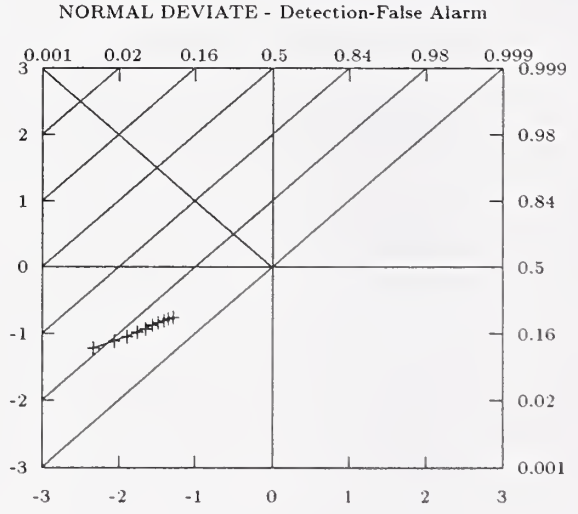
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1804 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1262 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0141 | 0.5720 |
| at 15 docs | 0.0356 | 0.5467 |
| at 30 docs | 0.0693 | 0.5353 |
| at 100 docs | 0.1825 | 0.4744 |
| at 200 docs | 0.2553 | 0.3503 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0799 |
| 0.00001 | 0.1129 |
| 0.00002 | 0.1358 |
| 0.00003 | 0.1497 |
| 0.00004 | 0.1670 |
| 0.00005 | 0.1788 |
| 0.00006 | 0.1928 |
| 0.00007 | 0.2058 |
| 0.00008 | 0.2131 |
| 0.00009 | 0.2214 |



# APPENDIX B

This appendix contains tables of results for the TIPSTER panel. The tables have the same descriptions as those in Appendix A. The results in this appendix represent a second round of work on this task, as a previous evaluation of TIPSTER results had taken place in September. The TIPSTER contractors were able to use some (minimal) failure analyis to correct some of the easier problems found in that evaluation in time for these results. Whereas these results cannot be compared directly with the TREC-1 results, they represent what can be accomplished in only one month of additional tuning for the task.

| Summary Statistics | |
|---|---|
| Run number | INQRY0–full, manual |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 10000 |
| Relevant | 16400 |
| Rel_ret | 4503 |

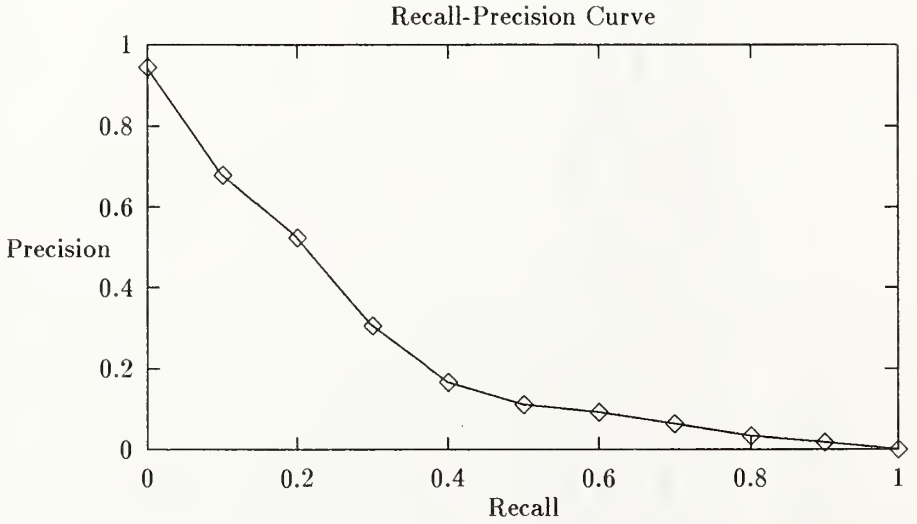
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.9448 |
| 0.10 | 0.6794 |
| 0.20 | 0.5250 |
| 0.30 | 0.3067 |
| 0.40 | 0.1668 |
| 0.50 | 0.1116 |
| 0.60 | 0.0926 |
| 0.70 | 0.0631 |
| 0.80 | 0.0337 |
| 0.90 | 0.0174 |
| 1.00 | 0.0000 |

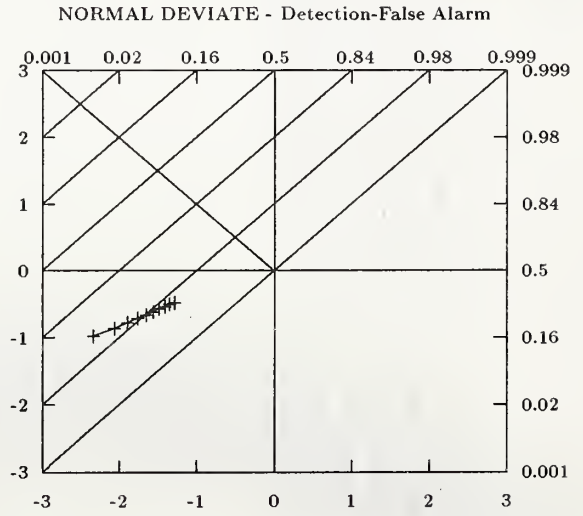
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.2674 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.2235 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0184 | 0.7640 |
| at 15 docs | 0.0549 | 0.7293 |
| at 30 docs | 0.0992 | 0.6840 |
| at 100 docs | 0.2451 | 0.5546 |
| at 200 docs | 0.3572 | 0.4503 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.1250 |
| 0.00001 | 0.1654 |
| 0.00002 | 0.1957 |
| 0.00003 | 0.2203 |
| 0.00004 | 0.2409 |
| 0.00005 | 0.2576 |
| 0.00006 | 0.2738 |
| 0.00007 | 0.2882 |
| 0.00008 | 0.2994 |
| 0.00009 | 0.3113 |



| Summary Statistics | |
|---|---|
| Run number | HNCmn1–disk1 only, manual |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 9982 |
| Relevant | 10477 |
| Rel_ret | 2711 |

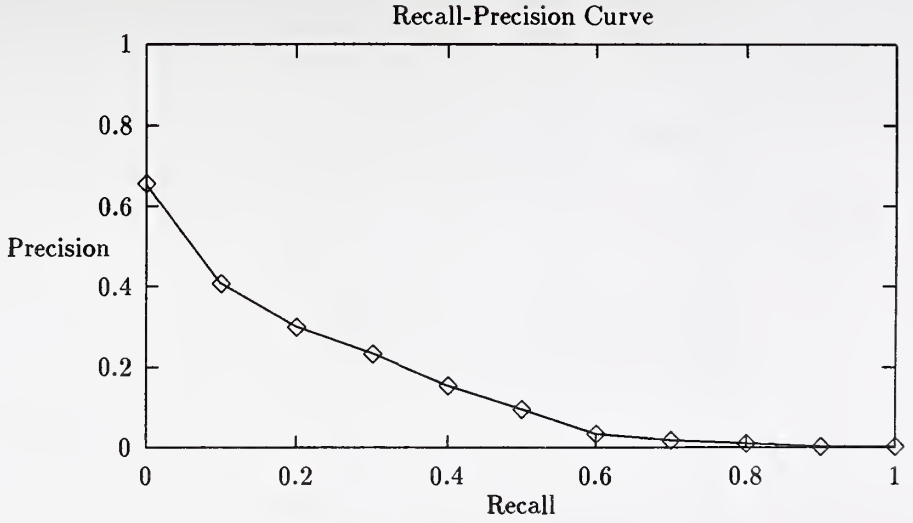
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.6569 |
| 0.10 | 0.4078 |
| 0.20 | 0.3005 |
| 0.30 | 0.2331 |
| 0.40 | 0.1540 |
| 0.50 | 0.0952 |
| 0.60 | 0.0338 |
| 0.70 | 0.0169 |
| 0.80 | 0.0101 |
| 0.90 | 0.0020 |
| 1.00 | 0.0020 |

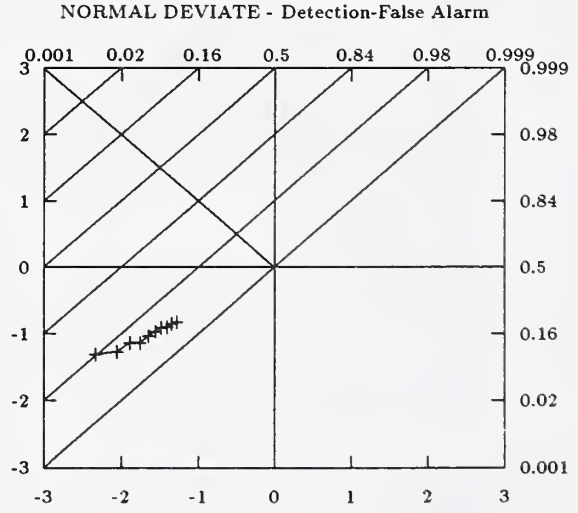
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.1739 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.1352 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0184 | 0.4400 |
| at 15 docs | 0.0469 | 0.4227 |
| at 30 docs | 0.0758 | 0.3847 |
| at 100 docs | 0.2098 | 0.3388 |
| at 200 docs | 0.3209 | 0.2711 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.0726 |
| 0.00001 | 0.0967 |
| 0.00002 | 0.1039 |
| 0.00003 | 0.1287 |
| 0.00004 | 0.1301 |
| 0.00005 | 0.1518 |
| 0.00006 | 0.1677 |
| 0.00007 | 0.1837 |
| 0.00008 | 0.1857 |
| 0.00009 | 0.1995 |



| Summary Statistics | |
|---|---|
| Run number | INQRYC–full, manual |
| Num_queries | 50 |
| **Total number of documents over all queries** | |
| Retrieved | 10000 |
| Relevant | 15459 |
| Rel_ret | 4527 |

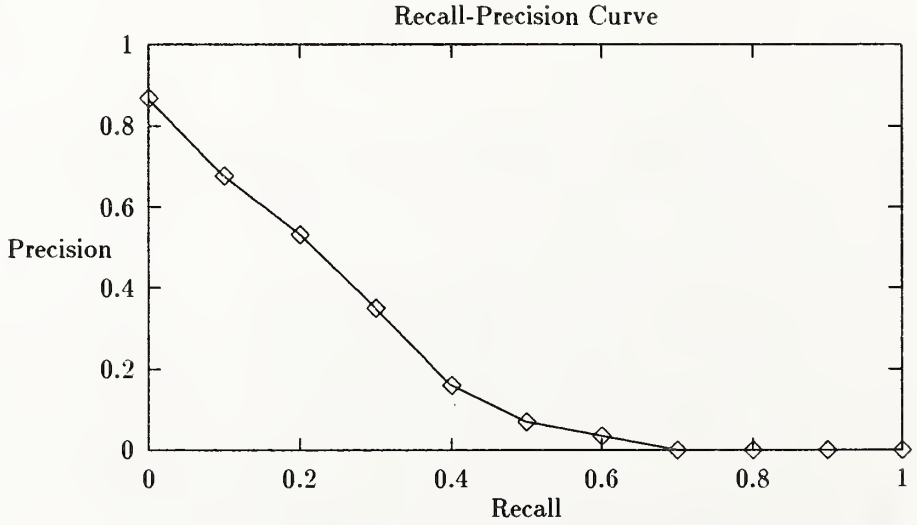
| Recall Level Averages | |
|---|---|
| Recall | Precision |
| 0.00 | 0.8688 |
| 0.10 | 0.6770 |
| 0.20 | 0.5321 |
| 0.30 | 0.3515 |
| 0.40 | 0.1606 |
| 0.50 | 0.0703 |
| 0.60 | 0.0342 |
| 0.70 | 0.0000 |
| 0.80 | 0.0000 |
| 0.90 | 0.0000 |
| 1.00 | 0.0000 |

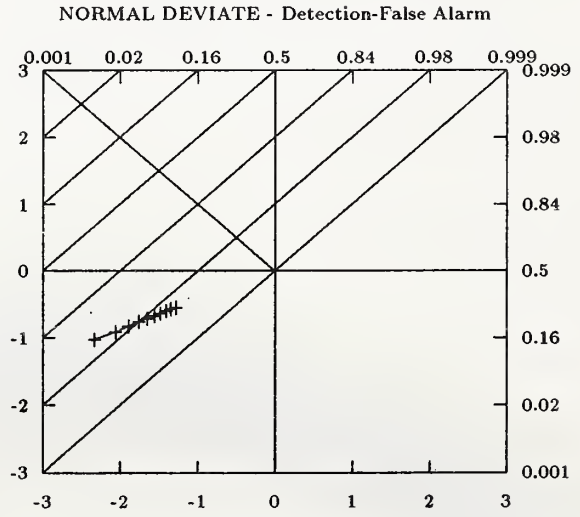
| Average precision for all points | |
|---|---|
| 11-pt Avg | 0.2449 |

| Average precision for 3 intermediate points (0.20, 0.50, 0.80) | |
|---|---|
| 3-pt Avg | 0.2008 |

| Document Level Averages | | |
|---|---|---|
| | Recall | Precision |
| at 5 docs | 0.0202 | 0.7080 |
| at 15 docs | 0.0542 | 0.6893 |
| at 30 docs | 0.0947 | 0.6493 |
| at 100 docs | 0.2268 | 0.5478 |
| at 200 docs | 0.3457 | 0.4527 |

| ROC Averages | |
|---|---|
| False Alarm | Detection |
| 0.00000 | 0.1161 |
| 0.00001 | 0.1544 |
| 0.00002 | 0.1833 |
| 0.00003 | 0.2082 |
| 0.00004 | 0.2262 |
| 0.00005 | 0.2414 |
| 0.00006 | 0.2537 |
| 0.00007 | 0.2670 |
| 0.00008 | 0.2797 |
| 0.00009 | 0.2899 |



# APPENDIX C

This appendix contains the supplemental forms filled out by each group about their system. These forms are meant to supplement the papers and contain a standarded and formatted description of system features and timing aspects.

## System Summary and Timing

### City University, London

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list **yes**
    a. how many words in list?
        **126 general stop words + 6 function words. Excluded from indexes and queries. Semi-stopword list of 256 words and phrases. These are not used in query expansion following relevance feedback unless they occur in the original query.**
2. is a controlled vocabulary used? **No. But see I C 1 b.**
3. stemming **yes**
    a. standard stemming algorithms
        **A moderately weak suffixing algorithm based on M. F. Porter, "An algorithm for suffix stripping." Program, 14(3), Jul 1980, 130-137. We also use a degree of British/American spelling conflation.**
    b. morphological analysis **no**
4. term weighting **No. Query terms are weighted, but not index terms.**
5. phrase discovery **no**
6. syntactic parsing **no**
7. word sense disambiguation **no**
8. heuristic associations **no**
9. spelling checking (with manual correction) **no**
10. spelling correction **no**
11. proper noun identification algorithm **no**
12. tokenizer (recognizes dates, phone numbers, common patterns) **no**
13. are the manually-indexed terms used? **no**

B. Statistics on data structures built from TREC text (please fill out each applicable section)

1. inverted index
    a. total amount of storage (megabytes) **810**
    b. total computer time to build (approximate number of hours) **43**
    c. is the process completely automatic? **yes**
    d. are term positions within documents stored? **No. Insufficient disk space to do this.**
    e. single terms only? **Single terms and pre-specified phrases (see I C 1 b below)**

C. Data built from sources other than the input text

1. internally-built auxiliary files **One manually-built file.**
    a. domain independent or domain specific (if two separate files, please fill out one set of questions for each file) **Loosely domain-dependent**
    b. type of file (thesaurus, knowledge base, lexicon, etc.)
        **Small quasi-thesaurus containing synonym classes, prefixes, go phrases,**

**stopwords, function words and semi-stopwords (see I A 1 a for semi-stopwords).**

c. total amount of storage (megabytes) **0.013**

d. total number of concepts represented **About 1500**

e. type of representation (frames, semantic nets, rules, etc.) **Simple lookup table**

f. total computer time to build (approximate number of hours)

**Manually built. Structured at runtime, time negligible.**

g. total manual time to build (approximate number of hours)

**Perhaps 8 person-hours. Several iterations, based on frequency counts from indexing runs, other similar files, TREC queries and documents.**

h. use of manual labor

(4) other (describe) **Manually built using text editor**

2. externally-built auxiliary file **no**

II. Query construction

(please fill out a section for each query construction method used)

A. Automatically built queries (ad hoc)

1. topic fields used

**Concepts. Other fields were tried but gave overall (though not uniformly) worse results.**

2. total computer time to build query (cpu seconds)

**0.02 seconds/query to parse topic and extract**

3. which of the following were used?

j. other (describe)

**Concept terms processed and weighted.**

**Term weight = constant * log(((r+c)/(R-r+1-c)) / ((n-r+c)/(N-n-R+r+1-c))) where N is the number of indexed documents, n the number of documents containing the term, R the number of known relevant documents, r the number of known relevant documents containing the term, c = 0.5. Weights rounded to nearest integer.**

B. Manually constructed queries (ad hoc)

1. topic fields used **Any: searchers' free choice.**

2. average time to build query (minutes) **About 40 minutes (often including trial searches)**

3. type of query builder

**Six searchers were used. None was a domain expert. Two might be described as experts on the search system.**

4. tools used to build query

c. other lexical tools (identify) **Trial lookups giving frequency. Trial searches.**

5. which of the following were used?

a. term weighting **As in II A 3 j above**

b. Boolean connectors (AND, OR, NOT)

**All available. AND and OR were used in a number of searches.**

d. addition of terms not included in topic

(1) source of terms

**Searchers' world knowledge and terms from relevant documents found in trial searches.**

C. Feedback (ad hoc)

1. initial query built by method 1 or method 2? **Method 2**

2. type of person doing feedback

**Searchers were Masters students in Information Science and two people working on the TREC project.**

3. average time to do complete feedback
    a. cpu time (total cpu seconds for all iterations) **About 20 seconds**
    b. clock time from initial construction of query to completion of final query (minutes) **About 20 minutes**
4. average number of iterations **One**
    a. average number of documents examined per iteration **About 20**
5. minimum number of iterations **One**
6. maximum number of iterations **One**
7. what determines the end of an iteration?
    **Searchers were recommended to stop after assessing 20 documents or when they had found 10 relevant documents. These guidelines were not always adhered to.**
8. feedback methods used
    b. automatic query expansion from relevant documents
        (2) only top X terms added (what is X)
        **Term pool was all query terms + all non-semi-stop terms from relevant documents. The former were given an R-value of R + 3 and an r-value of r + 2. Top 20 terms were used, selected in descending order of (term_weight * r) and weighted using the formula given previously. See section II A 3 j & I A 1 a for "R", "r", "semi-stop".**

D. Automatically built queries (routing)
1. topic fields used **Concepts**
2. total computer time to build query (cpu seconds)
    **Depended strongly on number of known relevant documents in training set and their length. Average perhaps 10 minutes.**
3. which of the following were used in building the query?
    a. terms selected from
        (1) topic
        (3) only documents with relevance judgments
    b. term weighting
        **As in II A 3 j above except R = R + 10 and r = r + 10 for concept terms.**

III. Searching

A. Total computer time to search (cpu seconds)
1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)
    **Typical figure for 12-term search producing output list of 350,000 document identifiers: 45 seconds (note that in an interactive production system we would use a weight threshold which would reduce this by perhaps 50%).**
2. ranking time (total cpu seconds to sort document list)
    **For list above: about 65 seconds (weight threshold would reduce by 50-90%).**

B. Which methods best describe your machine searching methods?
2. probabilistic model

C. What factors are included in your ranking?
2. inverse document frequency
    **Inverse document frequency and relevance information when available (see above for weighting scheme).**

IV. What machine did you conduct the TREC experiment on?
**Sun SPARC Server 4/330 with Sun IPC as fileserver**
How much RAM did it have? **16 megabytes**

What was the clock rate of the CPU? **Not specified. Sun claims 16 Mips.**

V. Some systems are research prototypes and others are commercial.
To help compare these systems:

1. How much "software engineering" went into the development of your system?
   **System is non-commercial. It has undergone continual modification since 1982 to meet the requirements of a number of different research projects, mainly on end-user bibliographic searching.**
2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?
   **Faster hardware would of course increase speed. Main bottleneck is disk & network I/O. Very large amounts of RAM (of the order of a gigabyte per process--or could be shared between processes searching the same database) would greatly reduce I/O dependence. On the software side, earlier versions of the system were often optimised for speed at some cost in added complexity and reduced flexibility. This optimisation has been removed from the version produced for TREC, partly because interactive searching by general users was not envisaged.**
   **It is impossible to give definite estimates on speed improvements. It would not be unreasonable to expect an order of magnitude improvement within current hardware and software constraints.**
3. What features is your system missing that it would benefit by if it had them?
   **Given enough disk we would have stored positional information in the indexes, and probably used it to modify document weights, perhaps by giving weight bonuses for term proximity. This would have increased inversion storage overheads to a little over 100% of bibliographic file size. (This is not really a "missing feature," because the system does have the capability.)**
   **We might have considered some form of weight adjustment for document length. This would involve a modification of the index structure which might just have been feasible within the disk constraints.**
   **Other possibilities worth investigating include phrase discovery and term dependency statistics.**

# System Summary and Timing

## University of Pittsburgh

### General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list
   a. how many words in list? **2,529 words on the list, including digital (0-9).**
3. stemming
   a. standard stemming algorithms
      **We use Porter stemming algorithm and it was implemented by C. Fox, using C programs.**
4. term weighting
5. phrase discovery
6. syntactic parsing
7. word sense disambiguation
8. heuristic associations
9. spelling checking (with manual correction)
10. spelling correction
11. proper noun identification algorithm
12. tokenizer (recognizes dates, phone numbers, common patterns)
13. are the manually-indexed terms used?
14. other techniques used to build data structures (brief description)

B. Statistics on data structures built from TREC text (please fill out each applicable section)

1. inverted index
   a. total amount of storage (megabytes)
      **For the storage space information, only the data on disk one is available. Following table provides the data in Megabytes unit.**

| | DOE | AP | ZIFF | WSJ |
|---|---|---|---|---|
| **Inverted files** | **162.3** | **199.8** | **143.7** | **223.4** |
| **Indexed files** | **3.0** | **2.2** | **2.4** | **2.1** |
| **Address files** | **4.3** | **1.7** | **1.7** | **2.6** |

**Note: * Data on FR is also not available (loaded into tapes);**
*** Address files are the indexed files which include document numbers and their offspring in the text files where the document stored.**

   b. total computer time to build (approximate number of hours)
      **Please refer to Table 1 in our paper.**
   c. is the process completely automatic? **yes**

d. are term positions within documents stored? **no**
e. single terms only? **yes**

C. Data built from sources other than the input text **--no**

II. Query construction

(please fill out a section for each query construction method used)

A. Automatically built queries (ad hoc)

1. topic fields used

**Training queries: Title and concepts are used. However, the nationality might be included if it's necessary to meet the narrative item.**

**Routing queries: The routing queries are the final converged queries from the training queries. There is no modification.**

**Ad hoc queries: Title, concepts are used, and some keywords from narrative items are added.**

2. total computer time to build query (cpu seconds)

**Computing time to build queries is not available.**

3. which of the following were used?

a. term weighting with weights based on terms in topics

**Term weighting for queries is assigned by the system. It is our research topic on term weights modification. Note the stemming algorithm used on document processing was also used on query terms.**

**Training queries: All term weights were assigned automatically by the system and also adjusted by the system using feedback information.**

**Routing queries: The term weights are those from the last generation of the training queries. No changes are applied.**

**Ad hoc queries: For one query individual the term weights were assigned manually by the researchers. The other query individuals' term weights were generated by the system. (Note: our system uses 10 query individuals searching documents simultaneously.) Also the term weights were adjusted by using the feedback information.**

C. Feedback (ad hoc)

1. initial query built by method 1 or method 2?

**Initial queries were built by method 1 (automatic).**

2. type of person doing feedback **Evaluation is done by our researchers.**
3. average time to do complete feedback **Please refer to Table 2 in our paper.**
4. average number of iterations **3 iterations on average.**
5. minimum number of iterations **0**
6. maximum number of iterations **9**
7. what determines the end of an iteration?

**No more relevant documents are retrieved or it is not valuable to do more feedback due to the time constraint.**

8. feedback methods used

**Query terms are automatically modified by the system using the genetic algorithm in our system.**

III. Searching

A. Total computer time to search (cpu seconds)

1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained) **Please refer to Table 2 in our paper.**
2. ranking time (total cpu seconds to sort document list) **not available**

B. Which methods best describe your machine searching methods?

1. vector space model **A distance function (Lp metric) is used as similarity measurement.**

C. What factors are included in your ranking?

15. other (specify)

**Document ranking bases on the distance. The shorter the distance, the high the rank. That is, the document with the shortest distance is put on the top of the list.**

IV. What machine did you conduct the TREC experiment on?

How much RAM did it have?

What was the clock rate of the CPU?

**Two types of systems are used.**

**Sun-670: 32 MB RAM and 40 MHz CPU clock rate;**

**Sun SPARC/IPC: 24 MB RAM and 25 MHz clock rate.**

V. Some systems are research prototypes and others are commercial.

To help compare these systems:

1. How much "software engineering" went into the development of your system?
2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?

   **If our system can be implemented on a parallel machine, the retrieval could be 10 times faster.**
3. What features is your system missing that it would benefit by if it had them?

   **There are a lot of parameters which can be adjusted to make our system more flexible and more adaptive. We need to build a good user interface on which several parameters can be controlled and manipulated by the users.**

## System Summary and Timing

Cornell University
Run 1: Single term automatic ad hoc run (global/local match)

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list
   a. how many words in list? **570**
2. is a controlled vocabulary used? **no**
3. stemming **yes**
   a. standard stemming algorithms
      which ones? **SMART**
4. term weighting
   **In docs, tf * idf, cosine normalization (ntc)**
   **In queries, tf * idf, cosine normalization (ntc)**
   **In sentences, tf * idf, no normalization (ntn)**
5. phrase discovery
6. syntactic parsing
7. word sense disambiguation
8. heuristic associations
9. spelling checking (with manual correction)
10. spelling correction
11. proper noun identification algorithm
12. tokenizer (recognizes dates, phone numbers, common patterns)
13. are the manually-indexed terms used?
14. other techniques used to build data structures (brief description)

B. Statistics on data structures built from TREC text (please fill out each applicable section)

1. inverted index
   a. total amount of storage (megabytes) **690**
   b. total computer time to build (approximate number of hours)
      **4.7 hours to create doc vectors from text**
      **0.7 hours to reweight doc vectors and produce inverted file**
   c. is the process completely automatic? **yes**
   d. are term positions within documents stored? **no**
   e. single terms only? **yes**

5. other data structures built from TREC text (what?)
   **Map from docid to text location (also gives title for each doc)**
   a. total amount of storage (megabytes) **68 Mbytes.**
   b. total computer time to build (approximate number of hours)
      **Time to create included in inverted file creation above.**
   c. is the process completely automatic? **Automatic**

d. brief description of methods used

other data structures built from TREC text (what?)
**Map from internal concept to token string**
a. total amount of storage (megabytes) **18 Mbytes**
b. total computer time to build (approximate number of hours)
**Time to create included in inverted file creation above.**
c. is the process completely automatic? **Automatic**
d. brief description of methods used

C. Data built from sources other than the input text
**None, other than stopword file.**

II. Query construction
(please fill out a section for each query construction method used)

A. Automatically built queries (ad hoc)
1. topic fields used **Topic, Nationality, Narrative, Concepts, Factors, Description**
2. total computer time to build query (cpu seconds) **1.5 seconds for all queries**
3. which of the following were used?
a. term weighting with weights based on terms in topics **(idf)**

III. Searching

A. Total computer time to search (cpu seconds)
**1465 seconds (includes retrieval + ranking + indexing 500 docs per query).**
1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)
2. ranking time (total cpu seconds to sort document list)

B. Which methods best describe your machine searching methods?
1. vector space model

C. What factors are included in your ranking?
1. term frequency
2. inverse document frequency
7. proximity of terms **within sentence needed for local sim.**
8. information theoretic weights
9. document length

IV. What machine did you conduct the TREC experiment on? **Sun SPARC 2**
How much RAM did it have? **64 MB**
What was the clock rate of the CPU? **40 MHz**

V. Some systems are research prototypes and others are commercial.
To help compare these systems:

1. How much "software engineering" went into the development of your system?
**About 3 person-years for the SMART system itself**
2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?
**Of course!**
**Retrieval local similarity needed to index 500 docs per query; this could all be done in advance if a single local approach had been decided on.**

**Reduce retrieval time by a factor of 5.**
**A 6 machine distributed version of SMART should be faster by a factor of 3 for both indexing and retrieval.**

3. What features is your system missing that it would benefit by if it had them?
**Distributed version has not fully been implemented yet.**

## System Summary and Timing

Cornell University
Run 2: Phrase automatic ad hoc (Cornell Global/Local)

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list
   a. how many words in list? **570**
2. is a controlled vocabulary used?
   **Not for single terms.**
   **A phrase list was automatically constructed from phrases occurring 25 times or more in the first doc set (D1). Only those phrases were used.**
3. stemming **yes**
   a. standard stemming algorithms
      which ones? **SMART**
4. term weighting
   **In docs, tf * idf, cosine normalization over length of single terms (ntc)**
   **In queries, tf * idf, cosine normalization over length of single terms (ntc)**
   **In sentences, tf * idf, no normalization (ntn)**
   **Phrases weighted using their natural tf*idf, cosine normalized by length of single terms, and divided by sqrt(2). [Phrase match worth 0.5 of single term match]**
5. phrase discovery
   a. what kind of phrase?
      **Adjacent non-stopwords, components stemmed, that occurred at least 25 times in the D1 document set.**
6. syntactic parsing
7. word sense disambiguation
8. heuristic associations
9. spelling checking (with manual correction)
10. spelling correction
11. proper noun identification algorithm
12. tokenizer (recognizes dates, phone numbers, common patterns)
13. are the manually-indexed terms used?
14. other techniques used to build data structures (brief description)

B. Statistics on data structures built from TREC text (please fill out each applicable section)

1. inverted index
   a. total amount of storage (megabytes) **840**
   b. total computer time to build (approximate number of hours)
      **9.7 hours to create doc vectors from text**
      **0.9 hours to reweight doc vectors and produce inverted file**
   c. is the process completely automatic? **yes**
   d. are term positions within documents stored? **no**

e. single terms only? **no**

5. other data structures built from TREC text (what?)
   **Map from docid to text location (also gives title for each doc)**
   a. total amount of storage (megabytes) **68 Mbytes.**
   b. total computer time to build (approximate number of hours)
      **Time to create included in inverted file creation above.**
   c. is the process completely automatic? **Automatic**

other data structures built from TREC text (what?)
**Map from internal concept to token string**
a. total amount of storage (megabytes) **25 Mbytes**
b. total computer time to build (approximate number of hours)
**Time to create included in inverted file creation above.**
c. is the process completely automatic? **Automatic**

other data structures built from TREC text (what?)
**Phrase dictionary (controlled vocabulary)**
**Phrases were adjacent non-stopwords, components stemmed, that occurred at least 25 times in the D1 document set.**
a. total amount of storage (megabytes) **14 Mbytes to store dictionary.**
b. total computer time to build (approximate number of hours)
**It took 5.8 CPU hours to index D1, finding 4,700,000 phrases and their collection stats. Of those phrases 158,000 occurred at least 25 times.**
c. is the process completely automatic?

C. Data built from sources other than the input text
**None, other than stopword file.**

II. Query construction
(please fill out a section for each query construction method used)

A. Automatically built queries (ad hoc)
1. topic fields used **Topic, Nationality, Narrative, Concepts, Factors, Description**
2. total computer time to build query (cpu seconds) **2.7 seconds for all 50 queries**
3. which of the following were used?
   a. term weighting with weights based on terms in topics **(idf)**
   b. phrase extraction from topics **yes, using controlled list of phrases**

III. Searching

A. Total computer time to search (cpu seconds)
**2405 seconds (includes retrieval + ranking + indexing 500 docs/query).**
1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)
2. ranking time (total cpu seconds to sort document list)

B. Which methods best describe your machine searching methods?
1. vector space model

C. What factors are included in your ranking?
1. term frequency
2. inverse document frequency
7. proximity of terms **for phrases and for local similarity between sentences**
9. document length

IV. What machine did you conduct the TREC experiment on? **Sun SPARC 2**
How much RAM did it have? **64 MB**
What was the clock rate of the CPU? **40 Mhz**

V. Some systems are research prototypes and others are commercial.
To help compare these systems:

1. How much "software engineering" went into the development of your system?
**About 3 person-years for the SMART system itself**
2. Given appropriate resources, could your system be made to run faster? By how much (estimate)? **Of course!**
3. What features is your system missing that it would benefit by if it had them?

## System Summary and Timing

Cornell University
Run 3: Automatic routing (Cornell Ide feedback)

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?
1. stopword list
a. how many words in list? **570**
2. is a controlled vocabulary used? **no**
3. stemming **yes**
a. standard stemming algorithms
which ones? **SMART**
b. morphological analysis
4. term weighting
**In docs + queries, tf * idf, cosine normalization (ntc) (in docs idf is based on collection frequency within doc set D1 only)**
5. phrase discovery
6. syntactic parsing
7. word sense disambiguation
8. heuristic associations
9. spelling checking (with manual correction)
10. spelling correction
11. proper noun identification algorithm
12. tokenizer (recognizes dates, phone numbers, common patterns)
13. are the manually-indexed terms used?
14. other techniques used to build data structures (brief description)

B. Statistics on data structures built from TREC text (please fill out each applicable section)
1. inverted index
a. total amount of storage (megabytes) **275**
b. total computer time to build (approximate number of hours)
**1.9 hours (not including time to index D1 to obtain collection frequency info)**
c. is the process completely automatic? **yes**
d. are term positions within documents stored? **no**
e. single terms only? **yes**
5. other data structures built from TREC text (what?)
**Map from docid to text location (also gives title for each doc)**
a. total amount of storage (megabytes) **24 Mbytes.**
b. total computer time to build (approximate number of hours)
**Time to create included in inverted file creation above.**
c. is the process completely automatic? **Automatic**

other data structures built from TREC text (what?)

**Map from internal concept to token string**

a. total amount of storage (megabytes) **13 Mbytes**

b. total computer time to build (approximate number of hours)

**Time to create included in inverted file creation for D1**

c. is the process completely automatic? **Automatic**

C. Data built from sources other than the input text

**None, other than stopword file.**

II. Query construction

(please fill out a section for each query construction method used)

D. Automatically built queries (routing)

1. topic fields used **Topic, Nationality, Narrative, Concepts, Factors, Description**
2. total computer time to build query (cpu seconds) **306**
3. which of the following were used in building the query?
   a. terms selected from
      (1) topic
      (3) only documents with relevance judgments
   b. term weighting
      (1) with weights based on terms in topics
      (2) with weights based on terms in all training documents
      (3) with weights based on terms from documents with relevance judgments
   i. expansion of queries using previously-constructed data structure (from part I)
      (1) which structure? **30 best terms from relevant docs**

III. Searching

A. Total computer time to search (cpu seconds) **293 seconds (includes retrieval + ranking).**

1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)
2. ranking time (total cpu seconds to sort document list)

B. Which methods best describe your machine searching methods?

1. vector space model

C. What factors are included in your ranking?

1. term frequency
2. inverse document frequency
9. document length

IV. What machine did you conduct the TREC experiment on? **Sun SPARC 2**

How much RAM did it have? **64 MB**

What was the clock rate of the CPU? **40 Mhz**

V. Some systems are research prototypes and others are commercial.

To help compare these systems:

1. How much "software engineering" went into the development of your system?
   **About 3 person-years for the SMART system itself**
2. Given appropriate resources, could your system be made to run faster? By how much (estimate)? **Of course!**

3. What features is your system missing that it would benefit by if it had them?

## System Summary and Timing

University of California, Berkeley

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list **Yes, augmented SMART stoplist**
    a. how many words in list? **About 600**
2. is a controlled vocabulary used? **no**
3. stemming
    a. standard stemming algorithms **yes**
        which ones? **SMART system (Version 10) stemmer**
    b. morphological analysis **none**
4. term weighting
    **yes. Weights determined from various frequency statistics by logistic regression**
5. phrase discovery **none**
6. syntactic parsing **none**
7. word sense disambiguation **none**
8. heuristic associations **none**
9. spelling checking (with manual correction) **none**
10. spelling correction **none**
11. proper noun identification algorithm **none**
12. tokenizer (recognizes dates, phone numbers, common patterns) **none**
13. are the manually-indexed terms used? **no**
14. other techniques used to build data structures (brief description)

B. Statistics on data structures built from TREC text (please fill out each applicable section)

1. inverted index
    a. total amount of storage (megabytes)
        **Ranges from 70 to 180 mb for each of the five collections**
    b. total computer time to build (approximate number of hours)
        **Ranges from 6 to 14 hours for each of the five collections**
    c. is the process completely automatic? **yes**
    d. are term positions within documents stored? **no**
    e. single terms only? **yes**

C. Data built from sources other than the input text **--no**

II. Query construction
(please fill out a section for each query construction method used)

A. Automatically built queries (ad hoc)

1. topic fields used **all**
2. total computer time to build query (cpu seconds) **around 3 seconds total per query**
3. which of the following were used?
    a. term weighting with weights based on terms in topics
    j. other (describe)

**Absolute and relative frequency of each stem in query were used to weight the stems, using a formula obtained by logistic regression from the WSJ relevance data.**

III. Searching

A. Total computer time to search (cpu seconds)
1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)
2. ranking time (total cpu seconds to sort document list)

B. Which methods best describe your machine searching methods?
2. probabilistic model

**Yes, probabilistic searching based on linked dependence assumption and two stages of logistic regression as described in Proceedings ACM/SIGIR Copenhagen June 1992.**

C. What factors are included in your ranking?
1. term frequency
2. inverse document frequency
3. other term weights (where do they come from?) **see 15. below**
5. position in document **stem occurrence frequencies in titles were doubled in some collections.**
9. document length
15. other (specify)

**variables used were:**

**absolute and relative frequency of stem in query**
**absolute and relative frequency of stem in document**
**inverse document frequency of stem in collection**
**global relative frequency of stem in all document texts**
**document length measured in stem-occurrences.**

IV. What machine did you conduct the TREC experiment on?

**Three different machines:**

**1. DECStation 5000/125 with 16 Megabytes RAM for most work.**
**2. DECStation 5000/125 with 64 Megabytes RAM for a little.**
**3. IBM Model 3090 for the logistic regression analysis.**

How much RAM did it have?
What was the clock rate of the CPU?

**25 MHz for the 16 Megahyte DECStation. This was used for the timed retrieval runs.**
**40 MHz for the 64 Megahyte DECStation.**

V. Some systems are research prototypes and others are commercial.
To help compare these systems:

1. How much "software engineering" went into the development of your system?

**None except for the novel two-stage probabilistic logic. The Berkeley system is an experimental prototype only, programmed as a minimal modification of the SMART**

**system.**

2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?

   **Yes, see discussion in SMART's documentation: SMART is "not strongly optimized for any one particular use." The Berkeley system has roughly the same efficiency characteristics as SMART.**

3. What features is your system missing that it would benefit by if it had them?

   **Would probably benefit from a conflator, a thesaurus, a disambiguator, phrase discovery, stem proximity detection, etc. The Berkeley system is a bare-bones design, intended only to explore the workability of staged logistic regression.**

## System Summary and Timing

Universitaet Dortmund
Single term automatic ad hoc run (Fuhr learning)

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list
   a. how many words in list? **570**
2. is a controlled vocabulary used? **no**
3. stemming **yes**
   a. standard stemming algorithms
      which ones? **SMART**
   b. morphological analysis
4. term weighting
   **In docs, linear combination of several factors**
   **In queries, tf * idf, cosine normalization (ntc)**
5. phrase discovery **no**
6. syntactic parsing **no**
7. word sense disambiguation **no**
8. heuristic associations **no**
9. spelling checking (with manual correction) **no**
10. spelling correction **no**
11. proper noun identification algorithm **no**
12. tokenizer (recognizes dates, phone numbers, common patterns) **no**
13. are the manually-indexed terms used? **no**
14. other techniques used to build data structures (brief description)
    **Coefficients for linear combinations used in weighting were determined automatically using Q1,D1,Judgments of Q1 on D1. This took 1.7 hours (not including 2.6 hours to index Q1,D1).**

B. Statistics on data structures built from TREC text (please fill out each applicable section)

1. inverted index
   a. total amount of storage (megabytes) **690**
   b. total computer time to build (approximate number of hours)
      **4.7 hours to create doc vectors from text**
      **1.7 hours to reweight doc vectors and produce inverted file**
   c. is the process completely automatic? **yes**
   d. are term positions within documents stored? **no**
   e. single terms only? **yes**
5. other data structures built from TREC text (what?)
   **Map from docid to text location (also gives title for each doc)**
   a. total amount of storage (megabytes) **68 Mbytes.**
   b. total computer time to build (approximate number of hours)

**Time to create included in inverted file creation above.**

c. is the process completely automatic? **yes**

other data structures built from TREC text (what?)

**Map from internal concept to token string**

a. total amount of storage (megabytes) **18 Mbytes**

b. total computer time to build (approximate number of hours)

**Time to create included in inverted file creation above.**

c. is the process completely automatic? **yes**

C. Data built from sources other than the input text

**None, other than stopword file.**

II. Query construction

(please fill out a section for each query construction method used)

A. Automatically built queries (ad hoc)

1. topic fields used **Topic, Nationality, Narrative, Concepts, Factors, Description**
2. total computer time to build query (cpu seconds) **1.5 seconds**
3. which of the following were used?

a. term weighting with weights based on terms in topics **(idf)**

III. Searching

A. Total computer time to search (cpu seconds) **383 seconds (includes retrieval + ranking).**

1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)
2. ranking time (total cpu seconds to sort document list)

B. Which methods best describe your machine searching methods?

1. vector space model
2. probabilistic model

C. What factors are included in your ranking?

1. term frequency
2. inverse document frequency
8. information theoretic weights
9. document length

IV. What machine did you conduct the TREC experiment on? **Sun SPARC 2**

How much RAM did it have? **64 MB**

What was the clock rate of the CPU? **40 Mhz**

V. Some systems are research prototypes and others are commercial.

To help compare these systems:

1. How much "software engineering" went into the development of your system?

**About 3 person-years for the SMART system itself**

**2 person-weeks for the Fuhr weighing code**

2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?

**Of course!**

**A 6 machine distributed version of SMART should be faster by a factor of 3 for both indexing and retrieval.**

3. What features is your system missing that it would benefit by if it had them?
**Distributed version has not fully been implemented yet.**

## System Summary and Timing

Universitaet Dortmund
Phrase automatic ad hoc (Fuhr learning)

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list
   a. how many words in list? **570**
2. is a controlled vocabulary used?
   **Not for single terms.**
   **A phrase list was automatically constructed from phrases occurring 25 times or more in the first doc set (D1). Only those phrases were used.**
3. stemming **yes**
   a. standard stemming algorithms
      which ones? **SMART**
   b. morphological analysis
4. term weighting
   **In docs, linear combination of several factors**
   **In queries, tf * idf, cosine normalization (ntc)**
5. phrase discovery
   a. what kind of phrase?
      **Adjacent non-stopwords, components stemmed, that occurred at least 25 times in the D1 document set.**
   b. using statistical methods
   c. using syntactic methods
6. syntactic parsing **no**
7. word sense disambiguation **no**
8. heuristic associations **no**
9. spelling checking (with manual correction) **no**
10. spelling correction **no**
11. proper noun identification algorithm **no**
12. tokenizer (recognizes dates, phone numbers, common patterns) **no**
13. are the manually-indexed terms used? **no**
14. other techniques used to build data structures (brief description)
    **Coefficients for linear combinations used in weighting were determined automatically using Q1,D1,,Judgments of Q1 on D1. This took 2.4 hours (not including 5.6 hours to index Q1,D1).**

B. Statistics on data structures built from TREC text (please fill out each applicable section)

1. inverted index
   a. total amount of storage (megabytes) **840**
   b. total computer time to build (approximate number of hours)
      **9.7 hours to create doc vectors from text**

**2.9 hours to reweight doc vectors and produce inverted file**

c. is the process completely automatic? **yes**

d. are term positions within documents stored? **no**

e. single terms only? **no**

5. other data structures built from TREC text (what?)

**Map from docid to text location (also gives title for each doc)**

a. total amount of storage (megabytes) **68 Mbytes.**

b. total computer time to build (approximate number of hours)

**Time to create included in inverted file creation above.**

c. is the process completely automatic? **yes**

other data structures built from TREC text (what?)

**Map from internal concept to token string**

a. total amount of storage (megabytes) **25 Mbytes**

b. total computer time to build (approximate number of hours)

**Time to create included in inverted file creation above.**

c. is the process completely automatic? **yes**

other data structures built from TREC text (what?)

**Phrase dictionary (controlled vocabulary)**

**Phrases were adjacent non-stopwords, components stemmed, that occurred at least 25 times in the D1 document set.**

a. total amount of storage (megabytes) **14 Mbytes to store dictionary.**

b. total computer time to build (approximate number of hours)

**It took 5.8 hours to index D1, finding 4,700,000 phrases and their collection stats. Of those phrases 158,000 occurred at least 25 times.**

c. is the process completely automatic?

C. Data built from sources other than the input text

**None, other than stopword file.**

II. Query construction

(please fill out a section for each query construction method used)

A. Automatically built queries (ad hoc)

1. topic fields used **Topic, Nationality, Narrative, Concepts, Factors, Description**
2. total computer time to build query (cpu seconds) **2.7 seconds**
3. which of the following were used?

a. term weighting with weights based on terms in topics **(idf)**

b. phrase extraction from topics **yes, using controlled list of phrases**

III. Searching

A. Total computer time to search (cpu seconds)

**374 seconds (includes retrieval + ranking).**

1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)
2. ranking time (total cpu seconds to sort document list)

B. Which methods best describe your machine searching methods?

1. vector space model
2. probabilistic model

C. What factors are included in your ranking?

1. term frequency
2. inverse document frequency
7. proximity of terms **(for phrases)**
8. information theoretic weights
9. document length

IV. What machine did you conduct the TREC experiment on? **Sun SPARC 2**
How much RAM did it have? **64 MB**
What was the clock rate of the CPU? **40 MHz**

V. Some systems are research prototypes and others are commercial.
To help compare these systems:

1. How much "software engineering" went into the development of your system?
   **About 3 person-years for the SMART system itself**
   **2 person-weeks for the Fuhr weighing code**
2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?
   **Of course!**
   **Can speed up phrase indexing by 40% by algorithm change (speed up has been done for single terms, but not for phrases)**
3. What features is your system missing that it would benefit by if it had them?

## System Summary and Timing

Universitaet Dortmund
Automatic routing (RPI feedback)

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list
   a. how many words in list? **570**
2. is a controlled vocabulary used? **no**
3. stemming **yes**
   a. standard stemming algorithms
      which ones? **SMART**
   b. morphological analysis
4. term weighting
   **In docs + queries, tf * idf, cosine normalization (ntc) (in docs idf is based on collection frequency within doc set D1 only)**
5. phrase discovery **no**
6. syntactic parsing **no**
7. word sense disambiguation **no**
8. heuristic associations **no**
9. spelling checking (with manual correction) **no**
10. spelling correction **no**
11. proper noun identification algorithm **no**
12. tokenizer (recognizes dates, phone numbers, common patterns) **no**
13. are the manually-indexed terms used? **no**
14. other techniques used to build data structures (brief description) **no**

B. Statistics on data structures built from TREC text (please fill out each applicable section)

1. inverted index
   a. total amount of storage (megabytes) **275**
   b. total computer time to build (approximate number of hours)
      **1.9 hours (not including time to index D1 to obtain collection frequency info)**
   c. is the process completely automatic? **yes**
   d. are term positions within documents stored? **no**
   e. single terms only? **yes**
5. other data structures built from TREC text (what?)
   **Map from docid to text location (also gives title for each doc)**
   a. total amount of storage (megabytes) **24 Mbytes.**
   b. total computer time to build (approximate number of hours)
      **Time to create included in inverted file creation above.**
   c. is the process completely automatic? **yes**

other data structures built from TREC text (what?)

**Map from internal concept to token string**

a. total amount of storage (megabytes) **13 Mbytes**

b. total computer time to build (approximate number of hours)

**Time to create included in inverted file creation of D1.**

c. is the process completely automatic? **yes**

C. Data built from sources other than the input text

**None, other than stopword file.**

II. Query construction

(please fill out a section for each query construction method used)

D. Automatically built queries (routing)

1. topic fields used **all**
2. total computer time to build query (cpu seconds)

**1300 seconds, not including time to index D1 (3.0 hours)**

3. which of the following were used in building the query?

a. terms selected from

(1) topic

b. term weighting

(1) with weights based on terms in topics

(2) with weights based on terms in all training documents

(3) with weights based on terms from documents with relevance judgments

III. Searching

A. Total computer time to search (cpu seconds) **312 seconds (includes retrieval + ranking).**

1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)
2. ranking time (total cpu seconds to sort document list)

B. Which methods best describe your machine searching methods?

1. vector space model
2. probabilistic model

C. What factors are included in your ranking?

1. term frequency
2. inverse document frequency
8. information theoretic weights
9. document length

IV. What machine did you conduct the TREC experiment on? **Sun SPARC 2**

How much RAM did it have? **64 MB**

What was the clock rate of the CPU? **40 MHz**

V. Some systems are research prototypes and others are commercial.

To help compare these systems:

1. How much "software engineering" went into the development of your system?

**About 3 person-years for the SMART system itself**

2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?

**Of course!**

**Due to algorithm flaw, CPU time for constructing routing query is about a factor of**

**5 too much (Algorithm found best terms to expand by even though we had requested expansion by 0 terms).**

3. What features is your system missing that it would benefit by if it had them?

## System Summary and Timing

### University of Illinois at Chicago

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

**Each document is represented as a set of word pairs. Pairs were formed from all adjacent words, plus all words separated by one and two intermediate words. Documents were the unit of organization for the data structure. If a pair occurred only once in a document it was dropped from the data structure for that document only.**

**A sample record is as follows:**

**MULTIMEDIA ENCYCLOPEDIA 2 WSJ880818-0014**

**The number of times the pair occurred in the document appears in the third field, just before the document id.**

A. Which of the following were used to build your data structures?

1. stopword list

**The stopword list from SMART version 10 was used. Some additional stop words from TREC markup codes were used.**

a. how many words in list? **The total size of the stoplist was 631 words.**

2. is a controlled vocabulary used? **none**
3. stemming **none**
   a. standard stemming algorithms
      which ones?
      **Some small stemming experiments were later performed using the code from SMART version 10 and three training queries. For query 002 stemming had no effect, while for query 026 it resulted in a 43% increase in recall, and for query 049 a 73% improvement in recall.**
   b. morphological analysis **none**
4. term weighting

**None. Weighting was planned but could not be implemented given limitations that arose.**

5. phrase discovery
   a. what kind of phrase?
      **Word pairs occurring within three word positions of one another.**
   b. using statistical methods **All such pairs were identified.**
   c. using syntactic methods
6. syntactic parsing **none**
7. word sense disambiguation **none**
8. heuristic associations
   a. short definition of these associations **Only the basic pairing associations used.**
9. spelling checking (with manual correction) **none**
10. spelling correction **none**

11. proper noun identification algorithm **none**
12. tokenizer (recognizes dates, phone numbers, common patterns) **none**
13. are the manually-indexed terms used? **none**
14. other techniques used to build data structures (brief description) **none**

B. Statistics on data structures built from TREC text (please fill out each applicable section)

1. inverted index **Based only on pairs, not individual terms.**
    a. total amount of storage (megabytes) **819 Megabytes**
    b. total computer time to build (approximate number of hours) **100 hours**
    c. is the process completely automatic? **yes**
    d. are term positions within documents stored? **no**
    e. single terms only? **none**
2. n-grams, suffix arrays, signature files **See B1.**

C. Data built from sources other than the input text **--no**

II. Query construction

(please fill out a section for each query construction method used)

A. Automatically built queries (ad hoc)

1. topic fields used **Title, Description, Narrative, and Concepts (only first two.)**
2. total computer time to build query (cpu seconds) **0.26 second.**
3. which of the following were used? **none**

D. Automatically built queries (routing)

1. topic fields used **Title, Description, Narrative, Concepts (first two).**
2. total computer time to build query (cpu seconds) **55 seconds**
3. which of the following were used in building the query?
    c. phrase extraction
        (2) from all training documents
            **Word pairs occurring in the relevant training documents for the query but not in the irrelevant documents were used.**

III. Searching

A. Total computer time to search (cpu seconds)

1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)
    **This was not optimized for the current experiments. Run time was approximately 20 minutes per search. Proper optimization will reduce this time.**
2. ranking time (total cpu seconds to sort document list) **.22 seconds**

B. Which methods best describe your machine searching methods?

4. n-gram matching

C. What factors are included in your ranking?

11. n-gram frequency

IV. What machine did you conduct the TREC experiment on? **IBM 3090/300J**

How much RAM did it have? **16 Meg for a virtual machine.**

What was the clock rate of the CPU? **14.5 nanoseconds, or 69 MHz.**

V. Some systems are research prototypes and others are commercial.

To help compare these systems:

1. How much "software engineering" went into the development of your system?
   **40 hours of new development, beyond using word pairing tools that were developed earlier over a period of years.**
2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?
   **Yes, search time could be reduced, but a reliable estimate of how much cannot be made at this time.**
3. What features is your system missing that it would benefit by if it had them?
   **Phrase weighting**
   **Term weighting and auxiliary single term search**
   **Stemming**
   **Removal of pair order effects**
   **Shortest path network search**

# System Summary and Timing

## Bellcore

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list **yes (though some experiments without stoplist)**
   a. how many words in list? **n=439; standard SMART list, I think**
2. is a controlled vocabulary used? **no**
3. stemming **none (except truncation at 20 characters/wd)**
4. term weighting **yes, log(tf)*(1-entropy)**
5. phrase discovery **no**
6. syntactic parsing **no**
7. word sense disambiguation **no**
8. heuristic associations **no**
9. spelling checking (with manual correction) **no**
10. spelling correction
    **no (not directly, but the LSI analyses does some of this for free)**
11. proper noun identification algorithm **no**
12. tokenizer (recognizes dates, phone numbers, common patterns)
13. are the manually-indexed terms used? **no**
14. other techniques used to build data structures (brief description)
    **LSI/SVD analysis of term-by-document matrix. Takes raw term-by-doc matrix; transforms entries using log entropy term weightings; calculates best "reduced-dimensional" approximation to transformed matrix using SVD. Number of dimensions 250-350. Does all query-doc matching in this reduced-dimension vector space.**

B. Statistics on data structures built from TREC text (please fill out each applicable section)

5. other data structures built from TREC text (what?)
   **LSI/SVD uses reduced-dimensional vectors (see below for description of how they are derived). The number of dims was between 235 and 250. There is one such vector for each term and for each document. Queries are also represented as vectors and compared to every document.**
   a. total amount of storage (megabytes)
      **All reduced dimensional vectors are stored in a binary database. Database consists of a vector for every doc and every term occurring in more than one doc. The vectors currently consist of single precision real values. For TREC, we built one database for each collection.**
      **Approx. 50000 docs are sampled. Terms that occur in more than one of these documents are used in the SVD analysis. The remaining docs are added to the database.**

      **DOE1 - docs: 226087, terms: 42221, ndim: 250 -> 262 meg db**

WSJ1 - docs: 99111, terms: 70019, ndim: 250 -> 169 meg db
AP1 - docs: 84930, terms: 78167, ndim: 250 -> 163 meg db
ZIFF1 - docs: 75180, terms: 60565, ndim: 250 -> 135 meg db
FR1 - docs: 26207, terms: 54713, ndim: 250 -> 80 meg db
WSJ2 - docs: 74520, terms: 76080, ndim: 235 -> 141 meg db
AP2 - docs: 79923, terms: 82997, ndim: 235 -> 153 meg db
ZIFF2 - docs: 56920, terms: 72197, ndim: 235 -> 121 meg db
FR2 - docs: 20108, terms: 48728, ndim: 235 -> 64 meg db

Used 250 dims for routing and 235 dims for ad hoc queries.
In general, database size will be: (ndocs+nterms)*ndim*4
The totals here are 1288 meg (750000 docs and 585000 terms).
If a single database had been used, the total would have been smaller because of term overlap--currently, many of the terms are represented in more than one database; there are only 200000 unique terms.

b. total computer time to build (approximate number of hours)

Four main stages:
1. indexing (extracting keys; calculating wts; etc.)
2. SVD (number of dimensions extracted ranged from 235-310)
NOTE 1: only 235-250 dims were actually used for retrieval. I don't have timing data for extracting only this smaller number of dimensions, but I'd estimate that the numbers for AP1, ZIFF1 and FR1 could be reduced by about 20%.
NOTE 2: initial indexing and SVD are typically done on a subset of ~50000 docs and nterms
3. various i/o translations (much of this will go away soon)
4. adding new docs to dbase (if sub-sampled for SVD).
SVD done on ~50000 docs; the remaining docs are indexed and added to the database after the SVD.
all times in MINUTES (SVD run on DEC5000; rest on SPARC2)
DOE1 - index: 49 SVD: 1219 io: 194 add: 591 SUM: 2053 mins
WSJ1 - index: 241 SVD: 1474 io: 174 add: 404 SUM: 2293 mins
AP1 - index: 271 SVD: 1644 io: 214 add: 455 SUM: 2584 mins
ZIFF1 - index: 241 SVD: 1359 io: 156 add: 352 SUM: 2108 mins
FR1 - index: 241 SVD: 939 io: 133 add: 0 SUM: 1313 mins
WSJ2 - index: 427 SVD: 1382 io: 220 add: 461 SUM: 2490 mins
AP2 - index: 338 SVD: 1210 io: 218 add: 273 SUM: 2039 mins
ZIFF2 - index: 260 SVD: 1452 io: 208 add: 0 SUM: 1920 mins
FR2 - index: 187 SVD: 486 io: 105 add: 0 SUM: 778 mins

c. is the process completely automatic? YES

d. brief description of methods used

LSI/SVD analysis of document collection
1. creates raw term-by-doc matrix; transforms entries using log entropy term weightings
2. calculates best "reduced-dimensional" approximation to transformed matrix using SVD. Number of dimensions in the SVD calculations ranged from 235 to 310. BUT, only 235 or 250 were used for the comparisons. Fewer dims could have been calculated, so some reported SVD times are higher than necessary. I'd estimate about 20% reductions in SVD times for AP1, ZIFF1, and FR1.
3. perform various database translations. Current SVD program outputs vectors in a different format and order than we need for the database. It

**will eventually output vectors in the appropriate database format, and this entire step can be omitted.**
**4. SVD calculations usually run on ~50,000 docs x nterms matrices. The remaining docs (if any) were indexed and added to the database here.**

C. Data built from sources other than the input text --no

II. Query construction

(please fill out a section for each query construction method used)

A. Automatically built queries (ad hoc) yes

**submitted two sets of ad hoc queries; queries were the same in both cases; only difference was how information from different sub-collections was combined**

1. topic fields used

**all (except NO manually indexed terms used)**

2. total computer time to build query (cpu seconds)

**Queries are vector sums of constituent term vectors**
**Separate query vector created for matching against each of 9 databases (DOE, WSJ1, AP1, FR1, ZIFF1, WSJ2, AP2, FR2, ZIFF2)**
**Time = .4 sec/query/database -> 3.6 secs/query**
**NOTE: These times simulate handling each query separately (so there is no i/o buffering). There are big improvements if you initially read in all the term vectors and create all the ad hoc queries at once.**

3. which of the following were used?

a. term weighting with weights based on terms in topics

**term weighting, but weights based on term usage in document collections**

h. expansion of queries using previously-constructed data structure (from part I)

**not really**

D. Automatically built queries (routing) yes

**submitted two sets of routing queries. Both were automatically created from**
**1) the text of the topics and**
**2) the relevant documents**

1. topic fields used

**all (except NO manually indexed terms) for both 1) and 2)**

2. total computer time to build query (cpu seconds)

**Queries are vector sums of constituent term vectors [case 1)] or document vectors [case 2)].**
**Separate query vector created for matching against each of 4 separate databases (WSJ1, AP1, FR1, ZIFF1)**
**Time = .4 sec/query/database in case 1) -> 1.6 secs/query**
**Time = .1 sec/query/database in case 2) -> 0.4 secs/query**
**NOTE: These times simulate handling each query separately (so there is no i/o buffering).**

3. which of the following were used in building the query?

a. terms selected from

(1) topic case 1)
(3) only documents with relevance judgments case 2)

b. term weighting

(2) with weights based on terms in all training documents

III. Searching

A. Total computer time to search (cpu seconds)

1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)

**Time = ~50000 query-doc comparisons/minute when all vectors are pre-loaded. Currently, we compare ALL docs to each query.**

**For ad hoc queries, the time to compare a query to the 750,000 docs is ~12 minutes**

**For routing queries, the time to compare a query (new doc) to the profiles (50 profiles in each of 4 databases) is about .3 sec**

2. ranking time (total cpu seconds to sort document list)

**none; it's included in the times given in 1. Currently both comparisons and ranking are done in the same routine**

B. Which methods best describe your machine searching methods?

1. vector space model

C. What factors are included in your ranking?

**Hum, not sure I get this. Similarity between a query and a document is the cosine between the query vector and the document vector. This cosine determines the rank.**

**Term weights are used to determine the location of the query vector. The query is located at the weighted vector sum of its constituent terms.**

1. term frequency **log(tf)*(1-entropy) term weight; so there's a tf part**

3. other term weights (where do they come from?)

**log entropy; weight come from training docs (disk1) for routing queries, and from both the training and test docs for ad hoc queries**

4. semantic closeness (as in semantic net distance)

**sort of; if you think of term vector locations as reflecting semantic associations. But these locations are auto derived from the SVD analysis**

8. information theoretic weights **log (tf) * (1-entropy)**

IV. What machine did you conduct the TREC experiment on?

How much RAM did it have?

What was the clock rate of the CPU?

**SVDs run on DEC5000 w/ ~400 meg; clock is ??? MHz**

**all else run on SPARC 2 w/ 384 meg; clock is 25 MHz (I think)**

V. Some systems are research prototypes and others are commercial.

To help compare these systems:

1. How much "software engineering" went into the development of your system?

**Real hard. The system was built as a research prototype to look at many different issues. I'd say about 1-2 person-years, but this is much more than would have been required if specs had been fixed at the beginning.**

2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?

**The existing tools were used pretty much as is for TREC, even though they were developed to work with much smaller databases. Also, there are far more parameters and options than we typically use. Almost no effort went into re-engineering for large databases or to more efficiently handle what we now use as default parameters.**

**Time in query construction and retrieval are spent:**

**1) seeking for vectors in a single large database of term and doc vectors. The database could easily be split.**

**2) many calculations (scalings of various sorts) are done on the fly. This could be eliminated if one knew that users wanted to retrieve only**

documents, for example. Currently both terms and docs can be retrieved with the same programs and scaling isn't done until we see that the user wants retrieved.

3) all calculations are done in floating point. Could be done with integers.

4) each ad hoc query was compared to EVERY document. This can be speeded up by some document clustering algorithms that we have looked at. This can also be speeded up tremendously by using more than one machine or by using a parallel machine. All vectors are independent, so it's trivial to split query processing.

I'd guess that improvements of a factor of 2-5 could be obtained just by tweaking items 1), 2) and 3).

Parallel query matching is the way to go. For example, we got speed-ups of 50-100 times using a MasPar for query storage and processing with no attempt to optimize.

In terms of pre-processing and SVD analyses:

1) about 10% of the time is spent in unnecessary I/O translation (because we've patched together pre-existing tools). Much of this will eventually go away.

2) more than 50% of the time is spent in the SVD. These algorithms get better and faster all the time (the algorithm we now use is about 100 times faster than what we used initially). There are speed-memory tradeoffs in different SVD algorithms, so time can probably be decreased by a factor of 2 or 3 by using more memory. Parallel algorithms will help some, but probably only by a factor or 2 or 3.

These are one-time costs for relatively stable domains. We've found that new items can be added to the existing solutions without redoing the scaling for a while.

Others ???

3. What features is your system missing that it would benefit by if it had them?

Precision would probably be increased by many of the standard things--phrases, proper noun identification, tokenizer (for dates, phone numbers, addresses, etc.), and some better handling of negation and union.

Some form of literal string matching might be useful to use in combination with LSI for some types of queries.

Others ???

## System Summary and Timing

Queens College, CUNY

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list **yes**
   a. how many words in list? **595**
2. is a controlled vocabulary used? **no**
3. stemming
   a. standard stemming algorithms **yes**
      which ones? **Porter's Algorithm**
   b. morphological analysis **no**
4. term weighting **yes**
5. phrase discovery **no**
6. syntactic parsing **no**
7. word sense disambiguation **no**
8. heuristic associations **no**
9. spelling checking (with manual correction) **no**
10. spelling correction **no**
11. proper noun identification algorithm **no**
12. tokenizer (recognizes dates, phone numbers, common patterns) **no**
13. are the manually-indexed terms used? **no**
14. other techniques used to build data structures (brief description)
    **A table of 396 manually created 2-word phrases. When these are identified in adjacent positions in documents or queries, they are used as additional index terms.**

B. Statistics on data structures built from TREC text (please fill out each applicable section)

1. inverted index
   a. total amount of storage (megabytes) **378**
   b. total computer time to build (approximate number of hours)
      **95+11+2=108 for 500MB. clock time**
   c. is the process completely automatic? **Yes, if sufficient disk. Not in this experiment.**
      if not, approximately how many hours of manual labor? **0.5**
   d. are term positions within documents stored?
      **No, but sentence yes. Can modify to capture word positions.**
   e. single terms only? **Yes, except for I.A.14.**
4. special routing structures (what?) **See I.B.5**
   **Network node, edge files. Routing using network node and edge files is straightforward.**
   a. total amount of storage (megabytes)
      **Node file: 4x7.5 Edge file: 4x4**
      **Network segmented into 4, because of insufficient ram.**
   b. total computer time to build (approximate number of hours)

**40+5+1+4x0.2=46.8, starting from text file.**

c. is the process completely automatic? **yes if sufficient ram and disk space.**

d. brief description of methods used

**1. Process (old) collection A.**

**2. Process queries against collection A.**

**3. Process new collection B as if they were queries--to make use of collection A statistics.**

**4. Combine queries, (old) dictionary and collection B into network for retrieval.**

5. other data structures built from TREC text (what?)

**1. Subdocument file**
**2. Coded file**
**3. Docid checking file**
**4. Termid checking file**
**5. Docnum file**
**6. Termnum (dictionary) file**
**7. Direct file**
**8. Index to direct file**
**9. Node file**
**10. Edge file**

a. total amount of storage (megabytes)

**1. 481** **2. 324**
**3. 7** **4. 4**
**5. 11** **6. 6**
**7. 372** **8. 19**
**9. 4x14** **10. 4x9**

**System was developed for experimental research, with flexibility to generate other data. Some of the files are not necessary for retrieval.**

b. total computer time to build (approximate number of hours)

**1. 1.5**
**2,3,4,5,6. 95**
**7,8. 11**
**9,10. 4x0.25=1**

c. is the process completely automatic?

**Yes if sufficient RAM and disk space. For this experiment, no.**
if not, approximately how many hours of manual labor? **2**

d. brief description of methods used

**raw text --> subdocument file**
**subdocument --> coded file, docid file, termid file, docnum file, termnum (dictionary) file.**
**Zipf-law program truncates dictionary via user assigned limits.**
**Coded, termnum --> direct file with index**
**direct --> inverted file**
**direct, inverted --> node, edge files.**

C. Data built from sources other than the input text

1. internally-built auxiliary files

a. domain independent or domain specific (if two separate files, please fill out one set of questions for each file) **phrase file**

b. type of file (thesaurus, knowledge base, lexicon, etc.) **word pair**

c. total amount of storage (megabytes) **0.005**

d. total number of concepts represented **396**

f. total computer time to build (approximate number of hours)

**0 (this is a file created via editor).**

g. total manual time to build (approximate number of hours) **16**

h. use of manual labor

(4) other (describe) **Search for WSJ terminology in library and from topics.**

2. externally-built auxiliary file **no**

II. Query construction

(please fill out a section for each query construction method used)

A. Automatically built queries (ad hoc)

1. topic fields used **<TITLE>, <DESC>, <NARR>, <CON>**
2. total computer time to build query (cpu seconds) **5 (average for each query).**
3. which of the following were used?
   a. term weighting with weights based on terms in topics **yes + others**
   b. expansion of queries using previously-constructed data structure **(from part I) yes**
      (1) which structure? **word-pair phrase file**

B. Manually constructed queries (ad hoc)

1. topic fields used **<TITLE>, <DESC>, <NARR>, <CON>**
2. average time to build query (minutes) **300 minutes for 25 queries**
3. type of query builder
   b. computer system expert
4. tools used to build query
   a. word frequency list **sometimes**
   b. knowledge base browser (knowledge base described in part I) **no**
   c. other lexical tools (identify) **no**
5. which of the following were used?
   a. term weighting
   b. Boolean connectors (AND, OR, NOT)
   d. addition of terms not included in topic
      (1) source of terms **word-pair phrase file**

C. Feedback (ad hoc)

1. initial query built by method 1 or method 2? **method 1**
2. type of person doing feedback
   b. system expert
3. average time to do complete feedback
   a. cpu time (total cpu seconds for all iterations)
      **12 per query per iteration--no expansion**
      **50 " " " " --with expansion**

   b. clock time from initial construction of query to completion of final query (minutes)
      **60 per query to do relevance judgment**
4. average number of iterations **1**
   a. average number of documents examined per iteration **10**
5. minimum number of iterations **1**
6. maximum number of iterations **1**
7. what determines the end of an iteration? **deadline + lack of manpower**
8. feedback methods used
   a. automatic term reweighting from relevant documents
   b. automatic query expansion from relevant documents
      (2) only top X terms added (what is X)
      **Top 20 most 'activated' terms that have document frequency < 2000 were used. Because many were already in query, about 12 on the average were new and added per query.**

c. other automatic methods
brief description **feedback is based on sub-documents**

D. Automatically built queries (routing)
1. topic fields used **<TITLE>, <DESC>, <NARR>, <CON>**
2. total computer time to build query (cpu seconds) **5 (average for each query)**
3. which of the following were used in building the query?
a. terms selected from
(1) topic
b. term weighting
(1) with weights based on terms in topics
(2) with weights based on terms in all training documents
(3) with weights based on terms from documents with relevance judgments
i. expansion of queries using previously-constructed data structure (from part I)
(1) which structure? **word-pair phrase file**

E. Manually constructed queries (routing)
1. topic fields used **<TITLE>, <DESC>, <NARR>, <CON>**
2. average time to build query (minutes) **300 minutes for 25 queries**
3. type of query builder
b. system expert
4. data used for building query
a. from training topic
5. tools used to build query
a. word frequency list **sometimes**
6. which of the following were used?
a. term weighting
b. Boolean connectors (AND, OR, NOT)

III. Searching

A. Total computer time to search (cpu seconds)
1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)
**8-20 per query without soft-Boolean (combine 2 methods).**
**20-60 " " with " " (combine 3 methods).**
2. ranking time (total cpu seconds to sort document list) **4-12 per query**

B. Which methods best describe your machine searching methods?
2. probabilistic model
8. neural networks

C. What factors are included in your ranking?
1. term frequency
3. other term weights (where do they come from?)
**inverse collection term frequency total word occurrences**
9. document length

IV. What machine did you conduct the TREC experiment on? **SPARC-2GS**
How much RAM did it have? **48 MB**
What was the clock rate of the CPU? **40 MHz**

V. Some systems are research prototypes and others are commercial.
To help compare these systems:

1. How much "software engineering" went into the development of your system?
   **Not much, time was spent to truncate record sizes to save space and fit certain structures in memory; replace some linked lists with arrays.**
2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?
   **Yes. 50-100%. Lots of code was translated from PASCAL to C and used as is.**
3. What features is your system missing that it would benefit by if it had them?
   **Dedicated SPARCstation.**
   **More RAM.**
   **More disk space.**

## System Summary and Timing

### New York University

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list **yes**
    a. how many words in list? **380**
2. is a controlled vocabulary used? **no**
3. stemming **yes**
    a. standard stemming algorithms **no**
    b. morphological analysis **yes**
4. term weighting **yes**
5. phrase discovery **yes**
    a. what kind of phrase? **NP's VP's, others**
    b. using statistical methods **partially**
    c. using syntactic methods **yes**
6. syntactic parsing **yes**
7. word sense disambiguation **no**
8. heuristic associations **yes**
    a. short definition of these associations **synonymy, specializations**
9. spelling checking (with manual correction) **no**
10. spelling correction **no**
11. proper noun identification algorithm **partial**
12. tokenizer (recognizes dates, phone numbers, common patterns) **no**
13. are the manually-indexed terms used? **no**
14. other techniques used to build data structures (brief description)

B. Statistics on data structures built from TREC text (please fill out each applicable section)

1. inverted index
    a. total amount of storage (megabytes) **290 MB (0.5 Gbyte txt)**
    b. total computer time to build (approximate number of hours) **250**
    c. is the process completely automatic? **yes**
    d. are term positions within documents stored? **yes**
    e. single terms only? **no**
3. knowledge bases **yes**
    a. total amount of storage (megabytes) **0.5**
    b. total number of concepts represented **3262**
    c. type of representation (frames, semantic nets, rules, etc.) **associations**
    d. total computer time to build (approximate number of hours) **175**
    e. total manual time to build (approximate number of hours) **0**
    f. use of manual labor **none**
    g. auxiliary files needed for machine use
        (1) machine-readable dictionary (which one?) **OALD**

C. Data built from sources other than the input text **--no**

II. Query construction

(please fill out a section for each query construction method used)

A. Automatically built queries (ad hoc) **yes**

1. topic fields used **<num> <title> <desc> and <narr>**
2. total computer time to build query (cpu seconds) **3.0**
3. which of the following were used?
   - b. phrase extraction from topics
   - c. syntactic parsing of topics
   - e. proper noun identification algorithm **partial**
   - g. heuristic associations to add terms
   - h. expansion of queries using previously-constructed data structure (from part I)
     - (1) which structure? **term clusters**
   - j. other (describe) **syntactic phrases**

D. Automatically built queries (routing)

1. topic fields used **same as in ad hoc**
2. total computer time to build query (cpu seconds) **3.2**
3. which of the following were used in building the query?
   - a. terms selected from
     - (2) all training documents
   - b. term weighting
     - (2) with weights based on terms in all training documents
   - c. phrase extraction
     - (1) from topics
     - (2) from all training documents
   - d. syntactic parsing
     - (1) of topics
     - (2) of all training documents
   - f. proper noun identification algorithm
     - (1) from topics **partial**
     - (2) from all training documents **partial**
   - h. heuristic associations to add terms
     - (2) from all training documents
   - i. expansion of queries using previously-constructed data structure (from part I)
     - (1) which structure? **clusters from training data**

III. Searching

A. Total computer time to search (cpu seconds)

1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)
   **TOTAL TIME (CPU + I/O) search and ranking is about 60 minutes per query**
2. ranking time (total cpu seconds to sort document list)

B. Which methods best describe your machine searching methods?

1. vector space model

C. What factors are included in your ranking?

1. term frequency
2. inverse document frequency

IV. What machine did you conduct the TREC experiment on?

How much RAM did it have? **56 Mbytes**

What was the clock rate of the CPU? **28 MIPS**

V. Some systems are research prototypes and others are commercial.

To help compare these systems:

1. How much "software engineering" went into the development of your system? **A lot**
2. Given appropriate resources, could your system be made to run faster? By how much (estimate)? **base IR system is very inefficient now**
3. What features is your system missing that it would benefit by if it had them?

   **There is still a lot of room for improvement of NLP programs; more time and experiments are required**

## System Summary and Timing

### University of Central Florida

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list **yes**
    a. how many words in list?
        **166 stop words, 122 abbreviations, 47 hyphenated words, 24 entries for abbreviations and alternate notions for months, 35 entries for legitimate words not to be prefixed, and 6 entries for legitimate prefixes.**
2. is a controlled vocabulary used? **no**
3. stemming **yes**
    a. standard stemming algorithms
        which ones? **J.B. Lovins' Stemming Algorithm (modified).**
    b. morphological analysis **none**
4. term weighting **yes**
5. phrase discovery **no**
6. syntactic parsing **no**
7. word sense disambiguation
    **Yes. The semantic lexicon we used is based on word senses found in Roget's Thesaurus.**
8. heuristic associations **no**
9. spelling checking (with manual correction) **no**
10. spelling correction **no**
11. proper noun identification algorithm **no**
12. tokenizer (recognizes dates, phone numbers, common patterns) **yes**
    a. which patterns are tokenized?
        **The QA System recognizes dates. But we felt it was not useful for the NIST experiment so we removed this feature to improve text processing speed.**
13. are the manually-indexed terms used? **no**
14. other techniques used to build data structures (brief description)
    **The QA System uses B-tree storage structures for inverted index file access and semantic lexicon access. But for the NIST experiments, we used the QA System text scanning ability and coupled it with hash table access (replacing the B-tree access) and the use of 32-bit codes for text strings.**

B. Statistics on data structures built from TREC text (please fill out each applicable section)

1. inverted index **yes**
    a. total amount of storage (megabytes)
        **For Vol. 1 the index storage was 385 megabytes.**
    b. total computer time to build (approximate number of hours)
        **73 hours using nine IBM 50 MHz 486 PCs running in parallel.**
    c. is the process completely automatic? **yes**

d. are term positions within documents stored? no
e. single terms only? yes

C. Data built from sources other than the input text yes
1. internally-built auxiliary files **yes, a semantic lexicon**
a. domain independent or domain specific (if two separate files, please fill out one set of questions for each file) **Domain independent**
b. type of file (thesaurus, knowledge base, lexicon, etc.)
**Semantic lexicon built by examination of Roget's Thesaurus.**
c. total amount of storage (megabytes) **0.34 megabytes.**
d. total number of concepts represented
**There are 36 semantic categories and there are approximately 24,000 words in two lexicons with the categories they trigger. The probability of each triggered category is also stored.**
e. type of representation (frames, semantic nets, rules, etc.) **It could be viewed as rules.**
f. total computer time to build (approximate number of hours)
(1) if already built, how much time to modify for TREC?
**Since the 1911 edition of Roget's Thesaurus became public domain recently, we spent approximately 16 hours creating the software to process the 1911 Thesaurus. Approximately 6 hours of processing time was required to automatically extract 20,000 lexicon entries. However, we did not have time to explore the use of these entries.**
g. total manual time to build (approximate number of hours)
(1) if already built, how much time to modify for TREC?
**Prior to TREC, there were 3,000 entries in the lexicon established by manual processing of approximately 6,000 words in 300 hours. For TREC, we made 1,000 new entries (in 85 hours) by examination of 1,700 frequently occurring words found in the training topics and the training text. So, the lexicon we used had 4,000 entries in it.**
h. use of manual labor
(4) other (describe) **Refer to (f) and (g).**
2. externally-built auxiliary file no

II. Query construction
(please fill out a section for each query construction method used)

A. Automatically built queries (ad hoc) yes
1. topic fields used **All fields**
2. total computer time to build query (cpu seconds) **1 second**
3. which of the following were used?
f. tokenizer (recognizes dates, phone numbers, common patterns)
**Dates recognizable but not used.**
h. expansion of queries using previously-constructed data structure (from part I)
(1) which structure? **Semantic lexicon described in I.C.1.**
j. other (describe) **Term weighting based on terms in training text.**

D. Automatically built queries (routing) yes
1. topic fields used **All fields**
2. total computer time to build query (cpu seconds) **1 second.**
3. which of the following were used in building the query?
a. terms selected from
(1) topic
b. term weighting
(2) with weights based on terms in all training documents

g. tokenizer (recognizes dates, phone numbers, common patterns)

**Dates are recognized by the QA System but were not used for the TREC experiments.**

i. expansion of queries using previously-constructed data structure (from part I)

(1) which structure? **Semantic lexicon described in I.C.1.**

III. Searching

A. Total computer time to search (cpu seconds) **3-10 minutes per query to retrieve and rank.**

1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)
2. ranking time (total cpu seconds to sort document list)

B. Which methods best describe your machine searching methods?

1. vector space model

C. What factors are included in your ranking?

1. term frequency
2. inverse document frequency
9. document length

IV. What machine did you conduct the TREC experiment on?

**We used nine IBM PS/2 Model 95 computers. These were 50 MHz 486 computers with 8 megabytes of RAM. Two of them had 16 megabytes of RAM. A 33 MHz 486 PC was used to distribute text to the nine IBM PCs for indexing and query processing.**

How much RAM did it have?

What was the clock rate of the CPU?

V. Some systems are research prototypes and others are commercial.

To help compare these systems:

1. How much "software engineering" went into the development of your system?

**Our QA System (built for NASA and restricted to an IBM compatible PC platform running under DOS and using no other license agreement commercial software such as a DOS extender) is a prototype and has been under development for one and a half years. Approximately 2,000 hours of programming have been used to develop the current software. The system is implemented in C and uses B-tree structures for the inverted file structure. We felt our system was not fast enough to appear reasonable for TREC, so we designed a separate system without a pleasant user interface which used a hashing scheme to establish codes for strings to cut down on storage space; we also eliminated the use of B-trees in this separate system. We custom built a system for TREC during July and August; approximately 400 hours of programming and debugging went into this effort. The custom system generated the results which we sent in. However, we are now trying to produce some semantic results using the original QA System.**

2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?

**Assuming we stay with DOS then we could easily run 8 to 16 times faster using the following:**

**Hardware Improvements:**

**1. New 66 MHz PCs now on the market.**

**2. Multiple hard drives.**

**3. 16 or 32 megabytes of RAM instead of 8 megabytes to be used for a larger disk cache and for our hashing algorithms.**

**Software Improvements:**

1. **Proper use of RAM for hashing.**
2. **Use of a DOS extender or switch to an OS/2 or UNIX environment.**

3. What features is your system missing that it would benefit by if it had them?

**The following software improvements would benefit the retrieval performance:**

1. **Relevance Feedback**
2. **Larger Semantic Lexicon**
3. **Breakdown of Lexicon into noun, verb, adjective, adverb, preposition, conjunction, interjection, use coupled with a part of speech tagger.**

## System Summary and Timing

Advanced Decision Systems

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list **yes**
   a. how many words in list? **421**
2. is a controlled vocabulary used? **no**
3. stemming **no**
4. term weighting **no**
5. phrase discovery **no**
6. syntactic parsing **no**
7. word sense disambiguation **no**
8. heuristic associations **no**
9. spelling checking (with manual correction) **no**
10. spelling correction **no**
11. proper noun identification algorithm **no**
12. tokenizer (recognizes dates, phone numbers, common patterns) **no**
13. are the manually-indexed terms used? **no**
14. other techniques used to build data structures (brief description)
    **yes--binary classification trees built automatically from the original documents and topic statements**

B. Statistics on data structures built from TREC text (please fill out each applicable section)

5. other data structures built from TREC text (what?)
   **yes--classification vectors; actually integer arrays**
   a. total amount of storage (megabytes)
      **Only a few Kbytes for the training sets used for the official scores--vectors generated on the fly for routing the test data.**
   b. total computer time to build (approximate number of hours)
      **Feature extraction takes less than 10 seconds per document.**
   c. is the process completely automatic? **yes**
   d. brief description of methods used
      **Give a specification of a set of features, for example, a list of word tokens; the document is searched for the number of occurrences of each feature.**

C. Data built from sources other than the input text **--no**

II. Query construction
(please fill out a section for each query construction method used)

D. Automatically built queries (routing)

1. topic fields used **<desc>, <narr>, <con>, <def>**
2. total computer time to build query (cpu seconds)
   **takes less than 30 seconds to build the classification tree including feature extraction--this does depend on the size of the training set though**
3. which of the following were used in building the query?
   a. terms selected from
      (1) topic **yes**
      (2) all training documents **no**
      (3) only documents with relevance judgments
         **yes--including some additional judgments generated by us**
   k. other (brief description)
      **feature counts--in this case these are just word counts**

III. Searching

A. Total computer time to search (cpu seconds)
1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)
   **approximately 20 hours (sic) of elapsed time on the WSJ test set--no accurate measures of CPU time available to us**
2. ranking time (total cpu seconds to sort document list)
   **approximately 5 minutes of elapsed time--no accurate measures of CPU time available to us**

B. Which methods best describe your machine searching methods?
10. other (describe)
   **binary classification algorithm based on counts of feature occurrence in the TES document**

C. What factors are included in your ranking?
15. other (specify)
   **statistical estimate of the misclassification rate (probability) of the classifier**

IV. What machine did you conduct the TREC experiment on?
**Sun SPARCstation IPC**
How much RAM did it have?
**24Mb**
What was the clock rate of the CPU?
**20MHz**

V. Some systems are research prototypes and others are commercial.
To help compare these systems:

1. How much "software engineering" went into the development of your system?
   **approximately 4 person-weeks for the TREC infrastructure--the CART algorithm implementation used was "off the shelf"**
2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?
   **Absolutely! The feature extraction algorithms were not optimized for speed, and no database or indexes were built to do the testing. With faster algorithms and a set of inverted indexes, we estimate a document could be classified in less than 1 second.**
3. What features is your system missing that it would benefit by if it had them?
   **We would like to experiment with "off the shelf" tools to assist in feature**

specification and extraction, for example: a part of speech tagger, a tokenizer, a proper name recognizer. We also did not explore the use of concept-based techniques (e.g., RUBRIC/TOPIC) to provide low-level concepts as features.

## System Summary and Timing

### CITRI, Royal Melbourne Institute of Technology

We are providing 2 reports on the system. This is because we have tried experiments on two very different systems, and tested quite different hypotheses.

Project: retrieval from a compressed database using the cosine measure and approximate representations of document lengths

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list **no**
2. is a controlled vocabulary used? **no**
3. stemming **yes, for construction of index**
    a. standard stemming algorithms
        which ones? **Lovins' 1968 algorithm**
4. term weighting **no**
5. phrase discovery **no**
6. syntactic parsing **no**
7. word sense disambiguation **no**
8. heuristic associations **no**
9. spelling checking (with manual correction) **no**
10. spelling correction **no**
11. proper noun identification algorithm **no**
12. tokenizer (recognizes dates, phone numbers, common patterns) **no**
13. are the manually-indexed terms used? **no**
14. other techniques used to build data structures (brief description)
    **no, but see discussion of compression below**

B. Statistics on data structures built from TREC text (please fill out each applicable section)

1. inverted index
    a. total amount of storage (megabytes)
        **50.7 Mb (37.9 Mb for pointers, 12.8 Mb for frequencies)**
    b. total computer time to build (approximate number of hours)
        **4.20 cpu hours, once a vocabulary has been built**
    c. is the process completely automatic? **yes**
    d. are term positions within documents stored?
        **no, but term frequency within document is stored**
    e. single terms only? yes
5. other data structures built from TREC text (what?)
    **model for subsequent compression of text**
    a. total amount of storage (megabytes) **2.4 Mb**
    b. total computer time to build (approximate number of hours) **2.54 hours**
    c. is the process completely automatic? **yes**

d. brief description of methods used
**count word and non-word frequencies using splay tree**

other data structures built from TREC text (what?)
**a single file of the text itself, compressed**
a. total amount of storage (megabytes) **253.2 Mb**
b. total computer time to build (approximate number of hours) **3.10 cpu hours**
c. is the process completely automatic? **yes**
d. brief description of methods used
**zero-order word-based model using Huffman coding**

other data structures built from TREC text (what?)
**a file of document addresses and document lengths (for cosine)**
a. total amount of storage (megabytes) **1.8 Mb**
b. total computer time to build (approximate number of hours) **negligible**

other data structures built from TREC text (what?) **vocabulary for inverted index**
a. total amount of storage (megabytes) **3.6 Mb**
b. total computer time to build (approximate number of hours) **2.41 cpu hours**
c. is the process completely automatic? **yes**
d. brief description of methods used **count stemmed word frequencies using splay tree**

other data structures built from TREC text (what?) **a file of inverted index entry addresses**
a. total amount of storage (megabytes) **1.2 Mb**
b. total computer time to build (approximate number of hours) **negligible**

other data structures built from TREC text (what?)
**a file of approximate document lengths**
a. total amount of storage (megabytes) **0.2 Mb**
b. total computer time to build (approximate number of hours) **negligible**

C. Data built from sources other than the input text **--no**

II. Query construction
(please fill out a section for each query construction method used)

A. Automatically built queries (ad hoc)
1. topic fields used **all**
2. total computer time to build query (cpu seconds) **less than one second**
3. which of the following were used?
a. term weighting with weights based on terms in topics **yes, as in cosine measure**
j. other (describe)
**used stop words to eliminate common words from query**
**eliminated SGML tags and all punctuation**

III. Searching

A. Total computer time to search (cpu seconds)
**1 & 2 were not timed separately; 35 seconds per query to identify the top 200 ranked items further 4.6 seconds of cpu decompress the top 200 items, 18.6 seconds in total including retrieval time**
1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)
2. ranking time (total cpu seconds to sort document list)

B. Which methods best describe your machine searching methods?

1. vector space model **cosine measure**

C. What factors are included in your ranking?

1. term frequency
2. inverse document frequency
9. document length **approximate document lengths were used to reduce memory requirements**

IV. What machine did you conduct the TREC experiment on? **Sun SPARC 2**

How much RAM did it have? **128 Mb**

What was the clock rate of the CPU? **25 MIP**

V. Some systems are research prototypes and others are commercial.

To help compare these systems:

1. How much "software engineering" went into the development of your system?
   **very little**
2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?
   **processing to rank items can be 30-50% faster; retrieval and decompression of text are currently limited by characteristics of the disk and the UNIX operating system**
3. What features is your system missing that it would benefit by if it had them?
   **current transformation of topics into queries is simple-minded**
   **the database is static**

## System Summary and Timing

### CITRI, Royal Melbourne Institute of Technology

We are providing 2 reports on the system. This is because we have tried experiments on two very different systems, and tested quite different hypotheses.

Project: retrieval from a compressed database using the cosine measure and approximate representations of document lengths

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list
    a. how many words in list? **420**
2. is a controlled vocabulary used? **no**
3. stemming
    a. standard stemming algorithms
        which ones? **Lovins' 1968 algorithm**
    b. morphological analysis **no**
4. term weighting **tf.idf**
5. phrase discovery
    a. what kind of phrase? **Adjacent pairs**
    b. using statistical methods **yes**
    c. using syntactic methods **no**
6. syntactic parsing **no**
7. word sense disambiguation **no**
8. heuristic associations **no**
9. spelling checking (with manual correction) **queries only**
10. spelling correction **queries only**
11. proper noun identification algorithm **no**
12. tokenizer (recognizes dates, phone numbers, common patterns) **no**
13. are the manually-indexed terms used? **they were not discarded**
14. other techniques used to build data structures (brief description)

B. Statistics on data structures built from TREC text (please fill out each applicable section)

2. n-grams, suffix arrays, signature files
    a. total amount of storage (megabytes)
        **Data (compressed) 220m**
        **Index 313m**
    b. total computer time to build (approximate number of hours) **23 hrs**
    c. brief description of methods used **multi-organisational signature FILE**
    d. is the process completely automatic? **yes**

C. Data built from sources other than the input text **--no**

II. Query construction

(please fill out a section for each query construction method used)

**A large number of techniques were tried.**

A. Automatically built queries (ad hoc)

1. topic fields used

**Boolean queries were constructed from a variety of the topic fields. The queries were then ranked possibly using different fields.**

2. total computer time to build query (cpu seconds) ~10
3. which of the following were used?

a. term weighting with weights based on terms in topics

b. phrase extraction from topics

i. automatic addition of Boolean connectors or proximity operators

III. Searching

A. Total computer time to search (cpu seconds)

1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)
2. ranking time (total cpu seconds to sort document list)

**These operations occurred together. It took 6 hrs to obtain a ranked list of 1,000 documents for each of the 50 queries.**

B. Which methods best describe your machine searching methods?

1. vector space model

C. What factors are included in your ranking?

1. term frequency
2. inverse document frequency
7. proximity of terms
9. document length
15. other (specify)

**The location of the term in the query. A variety of models were tried.**

IV. What machine did you conduct the TREC experiment on? **Sun SPARC 2**

How much RAM did it have? **128 Mb**

What was the clock rate of the CPU? **25 MIP**

V. Some systems are research prototypes and others are commercial.

To help compare these systems:

1. How much "software engineering" went into the development of your system?

**The system is a robust research tool. Limited effort has been put into speed.**

2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?

**Considerably faster, but we estimate it would twice as fast if we changed the architecture (we use UNIX pipes to communicate). It is unclear what other speed-ups can occur.**

3. What features is your system missing that it would benefit by if it had them?

**All sorts of things would be nice! A good form of transaction management would be the most useful to transform the system into a commercial product.**

## System Summary and Timing

### Australian Computing and Communications Institute

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

**The software does not invert the text. It inverts the queries (or filters) and passes the text through the combined index formed from the queries.**

II. Query construction

(please fill out a section for each query construction method used)

D. Automatically built queries (routing)

1. topic fields used **Concept field used**
2. total computer time to build query (cpu seconds) **< 5 seconds**
3. which of the following were used in building the query?
   a. terms selected from
      (1) topic
   b. term weighting
      (3) with weights based on terms from documents with relevance judgments
      **Terms weighted with weights based on terms from documents with relevance judgments, and dynamically modified through the training set and the test set.**
   c. phrase extraction
      (1) from topics
   j. automatic addition of Boolean connectors or proximity operators
      (1) using information from the topics

E. Manually constructed queries (routing)

1. topic fields used **All topic fields used**
2. average time to build query (minutes) **30 minutes**
3. type of query builder
   b. system expert
4. data used for building query
   a. from training topic
6. which of the following were used?
   b. Boolean connectors (AND, OR, NOT)
   c. proximity operators

III. Searching

A. Total computer time to search (cpu seconds)

**One message through 200 filters per second. This includes searching and ranking.**

1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)

2. ranking time (total cpu seconds to sort document list)

B. Which methods best describe your machine searching methods?

6. fuzzy logic (include your definition)

10. other (describe) **Software uses a fuzzy AND and Proximity measure to rank documents.**

C. What factors are included in your ranking?

1. term frequency

5. position in document

7. proximity of terms

IV. What machine did you conduct the TREC experiment on?

How much RAM did it have?

What was the clock rate of the CPU?

**The experiments were run on an HP 486/33 with 8 Mbytes under SCO UNIX. The CD ROM drive was accessed via NFS.**

V. Some systems are research prototypes and others are commercial.

To help compare these systems:

1. How much "software engineering" went into the development of your system?
2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?
3. What features is your system missing that it would benefit by if it had them?

**AMR is commercial strength software developed by Computer Power Group. Its commercialisation software engineering phase took some three person-years.**

# System Summary and Timing

## Carnegie Mellon University

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list

**No. But the NLP/morphological-analysis components of the system do limit the possible lexical categories of some English words to eliminate useless ambiguities. For example, "but" is given lexical category "cnj" (conjunction) and not alternative, possible categories, such as "sn" (singular-noun); "can" is limited to category "auxm" (modal-auxiliary-verb) and not "sn"; etc. Such selective restrictions have some of the effects of "stop-word" lists, since spurious (or irrelevant) categories will not enter into later indexing stages.**

**Furthermore, the NLP/parsing components of the system return simplex noun phrases (NPs) as candidate terms in which some components of the NP are eliminated, such as quantifiers (e.g., "many", "one", etc.), determiners (e.g., "the", "a", etc.), and conjunctions (e.g., "and", "or", etc.). In addition, in normal CLARIT NP processing, the parser does not return prepositions, non-NP adverbs, and extra-NP elements. This practice, therefore, also has the effect of eliminating items that normally appear on "stop-word" lists. It clearly goes beyond that practice in eliminating all extra-NP words as well.**

a. how many words in list?

**Approximately 100 lexical items have been given restrictive syntactic treatment, in addition to the words with unambiguously empty categories.**

2. is a controlled vocabulary used? No

3. stemming No

a. standard stemming algorithms

which ones?

b. morphological analysis

**Yes. The Morph component of the system provides for comprehensive inflectional-morphological analysis. In practice, the morph-normal form of nouns and adjectives is used in the NP-based terms of the system. Participles are not morphologically reduced (though it is possible to do so). Derivational-morphological analysis is not used. A lexicon of approximately 97,000 'root-form' items (English words) is the principal resource used by Morph in addition to its morphological rule set.**

4. term weighting

**Yes/No. The CLARIT process uses NLP to identify candidate terms in route to indexing, development of associated resources (e.g., thesauri), and analysis of queries or topics. These are taken as the 'information units' of interest and are analyzed statistically and heuristically. 'Weights' are associated with terms at various stages of processing. In indexing TREC documents, for example, an IDF/TF score was associated with terms for each document. In the case of multi-word terms (the**

norm), the terms are assigned IDF/TF scores, and each word in the term is broken out and assigned an independent IDF/TF score.

5. phrase discovery Yes
   a. what kind of phrase?
      Simplex noun phrases (= all modifiers and the head of the NP but no determiners, quantifiers, or post-head-position modifying phrases or clauses).
   b. using statistical methods
      No. NPs retained for thesaurus creation are scored using statistically-based measures of expected 'rarity' (based on component words), distribution, frequency, and coverage. But NPs are not identified in texts based on statistical parsing, for example.
   c. using syntactic methods
      Yes. NPs are discovered using a parser that implements a 'heuristic' grammar. In particular, following word-for-word morphological analysis (resulting in a set of syntactic-category tags for each word encountered in a text), the parser identifies the subsequences that form NPs. Identification of NPs is based on rules that perform NP-boundary-condition tests.
6. syntactic parsing Yes (see above). A single-pass parser follows morphological analysis.
7. word sense disambiguation
   No. No attempt is made to control for word senses in morphological or syntactic analysis. As noted above, disambiguation of grammatical categories is facilitated by restricting possible categories for selective items. In addition, absolute preferences are established for grammatical categories appearing in noun phrases.
8. heuristic associations
   a. short definition of these associations
      Yes. The principal relation the system currently uses is that of 'similarity' of terms. 'Similarity' is determined by different procedures in different contexts. For example, partial or 'fuzzy' matching of terms is facilitated by noting whether terms share words or attested subphrases. For example, in vector-space modeling of documents, the contained words of all terms (in the document vector as well as the query vector) are broken out, giving, in effect, the possibility of matching parts of terms (though, technically, the individual words are realized as independent dimensions of the space). In addition, in nominating terms for inclusion in thesauri and in matching terms to thesauri, CLARIT processing takes account of contained words and attested subphrases.
9. spelling checking (with manual correction) No
10. spelling correction No
11. proper noun identification algorithm
    Yes/No. The system provides for identification of 'candidate proper nouns' based on morphological analysis. (Essentially, since the morphological analysis is virtually exhaustive for English, words that cannot be mapped to specific lexical items are given the provisional label "cpn"--'candidate proper noun'--and parsing proceeds accordingly.) There is a facility in CLARIT for highly-reliable proper name (including acronym) identification, but it was not used in this round of TREC processing.
12. tokenizer (recognizes dates, phone numbers, common patterns)
    a. which patterns are tokenized?
       Certain common abbreviations are included in the lexicon and, under morphological processing, are rendered into normalized forms. The system can utilize--and even partially discover--supplemental lexicons of domain-specific abbreviations and other phrasal-lexical patterns, but this facility was not used for TREC processing.

13. are the manually-indexed terms used?

**Yes/No. The manually-indexed terms were treated as additional text and processed (for NPs) along with the other sections of the topic statement. They may or many not have survived review; they were not given special treatment except as potential sources of NPs for the topic.**

14. other techniques used to build data structures (brief description)

**The CLARIT system has facilities for the discovery of 'first-order' thesauri (= a list of important and characteristic terms) over collections of documents. The technique requires that documents in the collection be from the same 'domain' or 'topic' (broadly conceived) and is reliable only if the document set is large enough (e.g., minimally 2-3 megabytes). TREC topics--even when supplemented by sets of relevant documents--fall far short of the minimal size required, so general CLARIT thesaurus discovery could not be used in preparing topics or to support the indexing of texts. However, one effect of the CLARIT thesaurus-discover procedure is to rank terms in a collection based on their frequency, distribution, and 'rarity' scores. In preparing sets of terms to assist in partitioning the TREC corpus (to identify a subset of documents with the best candidates under any topic), we produced pseudo-thesauri for each topic by using CLARIT thesaurus-discovery modules. In particular, the process produced a list of terms from the available topic-relevant documents (or from a small sample of relevant documents that we may have found) and automatically chose the top (approximately 20%) ranked terms to supplement the original query (as derived from the topic statement) to produce a "routing/partitioning thesaurus" for the topic. (The use of this resource is described below.)**

**Furthermore, in developing extended queries for our final processing step (= a vector-space retrieval), we supplemented the original set of terms for the topic with *all* of the terms from the small set of top-ranking documents (as determined by routing/partitioning score) for each topic.**

B. Statistics on data structures built from TREC text (please fill out each applicable section)

4. special routing structures (what?)

**Yes. Each topic text was automatically analyzed by CLARIT to extract NPs. Terms nominated by parsing were reviewed by members of the CLARIT team for appropriateness (and retained or eliminated) and given a weight of "1", "2", or "3" to quantify relevance. Available topic-relevant documents were processed for supplemental terms (each given a fractional weight, e.g., "0.3"). The combined list--terms from the topic text and terms from the topic-relevant documents--formed a "routing/partitioning thesaurus" for the topic.**

**Each TREC document was 'scored' against the routing/partitioning thesaurus for each topic. In particular, every NP in each document was matched against the NPs (terms) in the routing thesaurus; partial matches were allowed; a formula yielded a composite score for the document based on the number of exact and partial hits as a function of document length and term 'value'.**

**In the first round (first 50 topics) of processing, this approach was used to identify the highest-scoring 2000 documents for each topic.**

a. total amount of storage (megabytes)

**0.6 megabytes for the merged 50 routing structures, i.e., the 50 "routing/partitioning thesauri" for the 50 topics.**

b. total computer time to build (approximate number of hours)

**5 minutes of real time--exclusive of the preparatory time to parse, build a simple index, find some relevant documents, review them, and combine them into an input file.**

c. is the process completely automatic?

**The manual review and weighting of terms from the topic statement took**

**approximately 5 minutes per topic. All additional steps were performed automatically.**

d. brief description of methods used (See above.)

5. other data structures built from TREC text (what?)

**Each TREC document had to be formatted for CLARIT processing, by making the unique text ID accessible to CLARIT as a special field and by delimiting the beginning and end of each text in a file. Intermediate (but unretained) files generated in CLARIT processing include a file of the words in each text, in their original order, annotated with morphological categories. Other files contain the output of the parser, as a list of NPs in the order in which they occurred in each text. The parsed representation of the text was retained and used at all subsequent steps of processing.**

a. total amount of storage (megabytes)

**Processing steps are piped through the system; intermediate files are not retained. The parsed representation of all the texts takes up approximately 98% of the space occupied by the original text.**

b. total computer time to build (approximate number of hours)

**The total time to transform the original 2-gigabytes of text into parsed text takes about 10 real hours, with processing distributed over 5 machines.**

c. is the process completely automatic? **Yes**

d. brief description of methods used

**A 'lex' program was used to reformat the TREC text to CLARIT format. The English morphological analyzer is written in C, and utilizes the lexicon of 97,000 items (mentioned above and further described below).**

**The noun phrase parser, also written in C, uses the grammatical categories supplied by the morphological analysis and an ATN-style rule set to extract noun phrases.**

C. Data built from sources other than the input text

1. internally-built auxiliary files

a. domain independent or domain specific (if two separate files, please fill out one set of questions for each file) **Domain independent**

b. type of file (thesaurus, knowledge base, lexicon, etc.)

c. total amount of storage (megabytes)

**CLARIT Lexicon (2 megabytes)**

**English-word statistics derived from the Grolier's Encyclopedia (2 megabytes)**

d. total number of concepts represented

**97,000 words (CLARIT Lexicon)**

**139,000 words (Grolier's list)**

e. type of representation (frames, semantic nets, rules, etc.)

**Lexicon: A sorted word list, giving for each word its possible grammatical categories and category-dependent normalization.**

**Grolier's: A list of words with distribution and frequency counts**

f. total computer time to build (approximate number of hours)

(1) if already built, how much time to modify for TREC?

**Already built--Not modified for TREC**

g. total manual time to build (approximate number of hours)

(1) if already built, how much time to modify for TREC?

**Already built--Not modified for TREC**

h. use of manual labor

(1) mostly manually built using special interface

(2) mostly machine built with manual correction

(3) initial core manually built to "bootstrap" for completely machine-built

completion

(4) other (describe)

**Initially derived from on-line sources but substantially modified and maintained manually**

2. externally-built auxiliary file
   a. type of file (Treebank, WordNet, etc.) **None**
   b. total amount of storage (megabytes)
   c. total number of concepts represented
   d. type of representation (frames, semantic nets, rules, etc.)

II. Query construction

(please fill out a section for each query construction method used)

B. Manually constructed queries (ad hoc)

**Note, as described below, there were only two steps in the CLARIT process that required non-automatic processing: (1) initial review and weighting of the index terms automatically-nominated and derived for the topic and (2) review of 1st-pass retrieved documents to identify 5-10 relevant ones for "feedback".**

1. topic fields used **<title>, <desc>, <narr>, <con>, <fac>, <def>**
2. average time to build query (minutes)

   **5 minutes--average time to review & weight automatically-nominated terms**
3. type of query builder **Graduate students**
4. tools used to build query
   c. other lexical tools (identify)

      **CLARIT noun-phrase parsing (extraction) nominated query terms from the textual descriptions of topics.**
5. which of the following were used?
   a. term weighting

      **Yes. Graduate students weighted terms with weights of "3", "2", or "1", according to whether the extracted term was central or peripheral to the topic. (Some extracted noun phrases were discarded as irrelevant or ill-formed; the vast majority were retained.)**

   c. proximity operators

      **No. Though proximity plays an implicit role when noun phrases are used as terms.**

   d. addition of terms not included in topic

      (1) source of terms **Not in the first round of routing**

   e. other (describe)

      **The ad hoc queries for the second fifty topics were formed in three stages. The first stage was the construction of a topic-derived routing/partitioning thesaurus.**

      **The routing/partitioning thesaurus was generated by CLARIT from the method described above, using only text fields of the topics. The automatically derived noun phrases were hand-weighted by graduate students with weights of "3", "2", or "1", according to whether the extracted term was central or peripheral to the topic. Some extraneous terms were deleted.**

      **The routing/partitioning thesaurus was passed over the parsed representation of original 1.2 gigabyte training set, inducing a ranking of all 500,000 documents using a scoring method taking account of exact and partial matches and document length. The top 50 documents were retained, for the next stage. These documents were manually judged by graduate**

**students, starting from the highest scored downward until 5-10 relevant documents were found. In effect, this represented a "relevance-feedback" step in the retrieval process.**

**In the next stage, the 5-10 "relevant" documents were used to produce a CLARIT-derived pseudo-thesaurus for the topic. (As described above, this consists of a list of prominent terms in the collection of documents, based on frequency, distribution, and "rarity" scores.) To this thesaurus were added the terms retained from the hand-weighting of the original topics. This thesaurus formed the second routing/partitioning thesaurus. The entire 2-gigabyte TREC collection was rescored against this second routing/partitioning thesaurus and the highest ranking 2000 documents were selected for the final-query stage.**

**The third, or final-query, stage involved, first, calculating an IDF/TF score for each term and all term-contained words in the 2000-document set for the topic. The query for that topic was created by taking the IDF/TF weightings of the terms from the originally chosen 5-10 relevant documents and automatically forming a query by combining all these terms along with the topic-derived terms into a long query vector. A vector-space representation of the 2000 documents was generated; the query vector was used to identify the final set of 200 ranked documents for each topic based on cosine similarity measures.**

D. Automatically built queries (routing)

1. topic fields used **<title>, <desc>, <narr>, <con>, <fac>, <def>**
2. total computer time to build query (cpu seconds) **0.03 cpu seconds**
3. which of the following were used in building the query?

a. terms selected from

(1) topic

(3) only documents with relevance judgments

b. term weighting

(1) with weights based on terms in topics

**Yes. Topic terms were initially hand weighted.**

c. phrase extraction

(1) from topics

(3) from documents with relevance judgments

d. syntactic parsing

(1) of topics

(2) of all training documents

(3) of documents with relevance judgments

g. tokenizer (recognizes dates, phone numbers, common patterns)

(1) which patterns are tokenized?

**Only simple acronyms such as "I.B.M." were automatically recognized as a unit.**

k. other (brief description)

**The routing queries were formed in two stages.**

**The first stage was the construction of a routing/partitioning thesaurus.**

**The routing/partitioning thesaurus was generated by CLARIT from the supplied list of relevant documents per topic. The text of the topic fields was parsed and added to the pseudo-thesaurus derived from the relevant documents. (Each pseudo-thesaurus consists of automatically chosen noun phrases scoring above a certain threshold, when scored for rarity, distribution, and frequency in the relevant document set.) Partial noun phrases, derived from**

**thesaurus entries, and attested in the documents, were also added to the thesaurus with a partial score.**
**The routing/partitioning thesaurus was passed over the parsed representation of 1.2-gigabyte training set, inducing a ranking of all 500,000 documents. The top 2000 documents were retained for the next stage.**
**The next stage of construction of each topic's routing/partitioning query began by calculating the IDF/TF score of all the terms and their contained words in the 2000 retained documents for that topic. The IDF/TF-weighted terms from the 5 relevant documents that were ranked highest in the previous stage were added to the original hand-weighted query terms, forming the final query.**
**For the second 900-megabyte data set, the routing/partitioning thesaurus developed in the first stage of processing (as described above) was used to select the 2000 highest-ranked documents.**
**The final query produced in the second stage (above) was used as a vector-space query (with partial matching) over the 2000 documents to produce a final set of 200 ranked documents for each topic.**

III. Searching

A. Total computer time to search (cpu seconds)

1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)

**The final set of 2000 documents for each topic was collected by the use of the routing/partitioning thesaurus (described above). This process was done simultaneously for all queries and took about 6 hours for the complete corpus.**

2. ranking time (total cpu seconds to sort document list)

**Once the vector-space matrix for the final set of 2000 documents was constructed, the actual comparison of the query vector to all other vectors in the matrix took on the order of 10-20 seconds.**

B. Which methods best describe your machine searching methods?

1. vector space model

**Yes. Using whole and partial matching on IDF/TF-weighted terms.**

C. What factors are included in your ranking?

1. term frequency
2. inverse document frequency
3. other term weights (where do they come from?)

**Topic terms were given additional factors of "1", "2", or "3".**

7. proximity of terms

**Parts of noun phrases are close. Our partial matching of noun phrases implicitly includes proximity.**

9. document length

IV. What machine did you conduct the TREC experiment on?

How much RAM did it have?

What was the clock rate of the CPU?

**Total available machines, used variously:**

**1 DECstation 5820 (64-Meg RAM)**
**2 DECstation 5000 (32-Meg RAM)**

1 DECstation 5000 (24-Meg RAM)
3 DECstation 3100 (24-Meg RAM)

V. Some systems are research prototypes and others are commercial.
To help compare these systems:

1. How much "software engineering" went into the development of your system?
   **The CLARIT system is a research prototype and has been under development for 4 years. The original system was implemented in Lisp; the current system has been re-engineered into C in the past 12 months.**
   **The specific configuration of the system used in the TREC experiments was produced in less than a week.**
   **As a research prototype, the system has minimal true "software engineering".**
2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?
   **Size constraints and the lack of global methods of attack caused us to duplicate work (both human and computer). Global methods that are smarter about resource consumption could make an order of magnitude difference. Almost all CLARIT processing is modular and separable; results of processes are additive/composable. Splitting the process across machines--or running in parallel--would greatly speed up the system.**
3. What features is your system missing that it would benefit by if it had them?
   **User interface.**
   **Some database mechanism for document storage.**
   **Potential "next features" include the following:**
   - **automatic spelling correction**
   - **integrated proper noun recognition**
   - **programmable token recognition**
   - **programmable / automated category assignment (guessing)**
   - **programmable / automated document structure analysis**
   - **automated synonym / related word discovery and use**
   - **database support for domains and thesauri, contexts, etc.**
   - **an integrated interface for both database construction and query elaboration**

## System Summary and Timing

ConQuest Software, Inc.

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list **yes**
    a. how many words in list? **70**
2. is a controlled vocabulary used? **no**
3. stemming
    a. standard stemming algorithms **no**
    b. morphological analysis **yes**
4. term weighting **yes**
5. phrase discovery **yes**
    a. what kind of phrase? **Paraphrase of Query**
    b. using statistical methods **Statistical proximity match**
    c. using syntactic methods **Limited**
6. syntactic parsing **Limited--POS assignment**
7. word sense disambiguation **In query by user, & in explosion of terms**
8. heuristic associations **yes**
    a. short definition of these associations **Terms associated via semantic net**
9. spelling checking (with manual correction) **In query only**
10. spelling correction **no**
11. proper noun identification algorithm **If identified by lexicon**
12. tokenizer (recognizes dates, phone numbers, common patterns)
    a. which patterns are tokenized? **Many**
13. are the manually-indexed terms used? **no**
14. other techniques used to build data structures (brief description)

    **Index organized hierarchically so that best documents (based on a coarse grained ranking algorithm) are returned to user while search continues on very large databases. Linked lists are used to connect and identify idioms. Semantic network term explosion is controlled by "weighted" links where weights are selected as either numerical or fuzzy sets based upon the link source and relationship.**

B. Statistics on data structures built from TREC text (please fill out each applicable section)

1. inverted index
    a. total amount of storage (megabytes) **1.2 Gb for 2.3 Gb text, 52%**
    b. total computer time to build (approximate number of hours) **150**
    c. is the process completely automatic? **yes**
        if not, approximately how many hours of manual labor? **Setup--4 hours**
    d. are term positions within documents stored? **yes**
    e. single terms only? **no**
3. knowledge bases
    a. total amount of storage (megabytes) **12 Mbytes**

b. total number of concepts represented **250,000+ concepts, 1.5M links**
c. type of representation (frames, semantic nets, rules, etc.) **Weighted semantic network**
d. total computer time to build (approximate number of hours) **0, already had it**
e. total manual time to build (approximate number of hours) **0**
f. use of manual labor
(2) mostly machine built with manual correction
**yes--but prior to TREC, not DB specific**
g. auxiliary files needed for machine use
(1) machine-readable dictionary (which one?) **Merriam Webster (abridged)**
(2) other (identify) **Word Net, plus several thesaurus files**

C. Data built from sources other than the input text **See 3(g) above**
1. internally-built auxiliary files **Semantic Network**
a. domain independent or domain specific (if two separate files, please fill out one set of questions for each file)
b. type of file (thesaurus, knowledge base, lexicon, etc.) **All in one**
c. total amount of storage (megabytes) **12**
d. total number of concepts represented **250,000+**
e. type of representation (frames, semantic nets, rules, etc.) **Semantic net**
f. total computer time to build (approximate number of hours) **Already had**
(1) if already built, how much time to modify for TREC? **None**
g. total manual time to build (approximate number of hours) **Already had**
(1) if already built, how much time to modify for TREC? **None**
h. use of manual labor
(2) mostly machine built with manual correction

II. Query construction
(please fill out a section for each query construction method used)

A. Automatically built queries (ad hoc)
1. topic fields used **Used entire topic with some simple filtering**
2. total computer time to build query (cpu seconds) **unknown, est. < 0.1 sec. ea.**
3. which of the following were used?
a. term weighting with weights based on terms in topics
b. phrase extraction from topics
c. syntactic parsing of topics
d. word sense disambiguation
e. proper noun identification algorithm **(look up)**
f. tokenizer (recognizes dates, phone numbers, common patterns)
(1) which patterns are tokenized? **many**
h. expansion of queries using previously-constructed data structure (from part I)
(1) which structure? **Tapered window**

B. Manually constructed queries (ad hoc)
1. topic fields used **User judgment**
2. average time to build query (minutes) **1-5 minutes**
3. type of query builder
b. computer system expert
4. tools used to build query
a. word frequency list **yes**
b. knowledge base browser (knowledge base described in part I) **yes**
(1) which structure from part I
c. other lexical tools (identify) **Lexicon**
5. which of the following were used?

a. term weighting **yes**
b. Boolean connectors (AND, OR, NOT) **Available. Not used.**
c. proximity operators **Automatic**
d. addition of terms not included in topic **yes--based on user judgment**
(1) source of terms
e. other (describe)

C. Feedback (ad hoc) **Available. Not submitted in TREC.**

D. Automatically built queries (routing) **Available. Not submitted in TREC.**

E. Manually constructed queries (routing) **Available. Not submitted in TREC.**

III. Searching

A. Total computer time to search (cpu seconds) **2-10 seconds, dep. on query**
1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained) **see above**
2. ranking time (total cpu seconds to sort document list) **included in number above**

B. Which methods best describe your machine searching methods?
1. vector space model **Some techniques used**
2. probabilistic model **Some probability used in ranking**
5. Boolean matching **Available. Not used in TREC.**
6. fuzzy logic (include your definition) **Fuzzy semantic net links used in term explosion.**
8. neural networks **No--see 6**
9. conceptual graph matching **Yes--query concept created by explosion**

C. What factors are included in your ranking?
1. term frequency
2. inverse document frequency **Available. Not used in TREC.**
3. other term weights (where do they come from?) **Manual**
4. semantic closeness (as in semantic net distance) **yes**
5. position in document **Available. Not used in TREC.**
6. syntactic clues (state how) **Available. Not used in TREC.**
7. proximity of terms
9. document length
10. completeness (what % of the query terms are present)
15. other (specify) **User chooses--programmable**

IV. What machine did you conduct the TREC experiment on? **Sun SPARC II**
How much RAM did it have? **64 Mbytes**
What was the clock rate of the CPU? **50 MHz**

V. Some systems are research prototypes and others are commercial.
To help compare these systems:

1. How much "software engineering" went into the development of your system?
**The underlying "engine" used for TREC is also used in a commercial product (ConQuest)--hence, lots of S/W engineering is behind it.**
2. Given appropriate resources, could your system be made to run faster? By how much (estimate)? **Yes--at least a factor of 2**
3. What features is your system missing that it would benefit by if it had them?
**Subject domain add-on dictionary.**

## System Summary and Timing

### GE Research and Development Center

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

**We did no pre-indexing of the data**

B. Statistics on data structures built from TREC text **no data provided**

C. Data built from sources other than the input text **--no**

II. Query construction

(please fill out a section for each query construction method used)

B. Manually constructed queries (ad hoc)

1. topic fields used **Mostly description, narrative, and concepts**
2. average time to build query (minutes) **About 20 minutes for initial query**
3. type of query builder
   b. computer system expert
4. tools used to build query
   b. knowledge base browser (knowledge base described in part I)
      (1) which structure from part I **inverted samples of corpus**
5. which of the following were used?
   b. Boolean connectors (AND, OR, NOT)
   c. proximity operators
   d. addition of terms not included in topic
      (1) source of terms
      **system lexicon, statistical analysis of samples matched by initial queries**

C. Feedback (ad hoc) **We did not do feedback, but we did query refinement**

E. Manually constructed queries (routing) **Ad hoc and routing were done using the same method**

1. topic fields used
2. average time to build query (minutes) **About 20 minutes for initial query**
3. type of query builder
   b. system expert
4. data used for building query
   b. from all training documents **statistical analysis of samples retrieved**
   c. from documents with relevance judgments
      **used for training, testing, and word frequency analysis**
5. tools used to build query
   d. machine analysis of training documents
      (1) describe
      **Word weighting analysis to determine what terms to add to queries**

6. which of the following were used?
   b. Boolean connectors (AND, OR, NOT)
   c. proximity operators
   e. other (brief description)
      **system lexicon, statistical analysis of samples matched by initial queries**

III. Searching

A. Total computer time to search (cpu seconds)
   1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)
      **About 20 hours = 72000 CPU seconds. As the documents are not pre-indexed, this includes all operations on all documents**
   2. ranking time (total cpu seconds to sort document list) **About 300 CPU seconds**

B. Which methods best describe your machine searching methods?
   5. Boolean matching
   7. free text scanning

C. What factors are included in your ranking?
   5. position in document
   15. other (specify) **Number of hits on topic description**

IV. What machine did you conduct the TREC experiment on? **SUN SPARCstation-2**
   How much RAM did it have? **48 Meg**
   What was the clock rate of the CPU? **standard**

V. Some systems are research prototypes and others are commercial.
   To help compare these systems:
   **Our system used a pattern matcher and lexicon that have been commercially developed, but the basic Boolean document processing engine was developed for TREC in a few days**
   1. How much "software engineering" went into the development of your system?
      **2 days for the basic engine**
   2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?
      **Processing time per document could easily he improved by a factor of 2. Processing time for ad hoc retrieval could be improved hy a factor of about 100,000 by using an inverted indexing strategy, at a cost of additional storage and indexing time for the corpus.**
   3. What features is your system missing that it would benefit by if it had them?
      **Automatic query generation, aids for compiling queries from higher-level descriptions**

## System Summary and Timing

### TRW

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

**None--we used a free text scanning approach. CD-ROM data was decompressed and loaded onto magnetic disk in raw form.**

B. Statistics on data structures built from TREC text (please fill out each applicable section) **--none**

C. Data built from sources other than the input text **--none**

II. Query construction

(please fill out a section for each query construction method used)

A. Automatically built queries (ad hoc)

**We performed some initial trials with building queries based on word frequency taken from documents from the initial relevance judgments supplied by NIST. Unfortunately, this appeared to lead us down a blind alley, perhaps because the initial judgments were not all that good. We are planning to try this again with the new judgments.**

B. Manually constructed queries (ad hoc) **We primarily used this method for the TREC queries.**

1. topic fields used
2. average time to build query (minutes)

**The initial query would take a couple of minutes to manually form by cutting and pasting from the topic descriptions with a text editor. "Feedback" was the human looking at the retrieved documents, comparing with the sample good documents supplied by NIST, making independent judgments on document relevance, and refining the query in an iterative manner.**

3. type of query builder
   b. computer system expert
4. tools used to build query **no special tools**
   a. word frequency list
   b. knowledge base browser (knowledge base described in part I)
      (1) which structure from part I
   c. other lexical tools (identify)
5. which of the following were used?
   b. Boolean connectors (AND, OR, NOT)
   c. proximity operators
   d. addition of terms not included in topic
      (1) source of terms

**Additional terms were supplied by human based on outside knowledge or from reading the text.**

C. Feedback (ad hoc)

1. initial query built by method 1 or method 2?

**Initial queries were built by human from subset of topic keywords.**

2. type of person doing feedback

b. system expert **computer system analyst**

3. average time to do complete feedback **We did this manually.**

a. cpu time (total cpu seconds for all iterations)

**A human refining the queries for an hour might use 10 minutes of FDF time.**

b. clock time from initial construction of query to completion of final query (minutes)

**Feedback/query refinement was done manually. Some topics were fairly easy, with reasonable results being achieved in less than an hour. Others, took several hours.**

4. average number of iterations

a. average number of documents examined per iteration **Typically 20-30.**

5. minimum number of iterations **Maybe 10.**

6. maximum number of iterations **Maybe 100.**

7. what determines the end of an iteration?

**Each iteration is (1) the human updates the queries, (2) the machine executes, (3) the human reviews the retrieved documents.**

**We stopped working on a topic when it seemed that the results were converging to practical limit for our approach, i.e., adding additional synonym keywords, or changing the query structure, wasn't going to produce more reasonable results.**

8. feedback methods used

d. manual methods

(1) using individual judgment with no set algorithm

**After working through the first dozen or so topics, we started to fall into a semi-routine. We are still thinking about the nature of this "routine" and what types of tools could help automate it.**

E. Manually constructed queries (routing)

**Same answers as for ad hoc. If fact, given our query language approach, final ad hoc queries and routing queries are the same.**

III. Searching

A. Total computer time to search (cpu seconds)

1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)

**Time to process a single set of topic queries against 1.2GB is 2-3 minutes.**

**Time to load the tipster corpus (read from CD-ROM, decompress, and load onto FDF's disk) was less than 8 hours.**

2. ranking time (total cpu seconds to sort document list) **1-2 seconds**

B. Which methods best describe your machine searching methods?

7. free text scanning

**To perform the actual searches, we used the fast data finder (FDF) text search hardware. The FDF implements a wide variety of pattern matching functions including word/string/phrase matching, fuzzy matches, Boolean logic, proximity operators, term counting, term completeness, and numeric ranging.**

C. What factors are included in your ranking?

5. position in document

7. proximity of terms

9. document length
10. completeness (what % of the query terms are present)
12. word specificity (i.e., animal vs. dog vs. poodle)

**To provide a coarse-grain ranking we ran several queries per topic, to provide increasing levels of recall. The five methods above were used in addition to Boolean logic, numeric ranging, and word order.**

IV. What machine did you conduct the TREC experiment on? **Sun-3/160 with FDF2000 and C-51 disk array**
How much RAM did it have? **8 MB**
What was the clock rate of the CPU?

**A Sun-3/160 is a couple of Mips. Note that the Sun is just the host, the FDF does the actual pattern matching. The FDF2000 model used for TREC clocks at around 12 MHz.**

V. Some systems are research prototypes and others are commercial.
To help compare these systems:

1. How much "software engineering" went into the development of your system?
   **No special programming was done for the TREC conference. The FDF system itself was the result of extensive prior development.**
2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?
   **How fast would you like it to go? The system used to execute the TREC queries was 20% of a full-up system. We're currently working on software that will automatically coordinate multiple FDF systems to working in parallel. We're also considering faster FDF chips and data transfer methods.**
3. What features is your system missing that it would benefit by if it had them?
   **The next generation of FDF systems have an in-hardware term weighting capability that can be used, in combination with the existing features, to return a numeric score for a document. This would allow for finer grain in ranking. New model prototypes were not available for this effort.**

## System Summary and Timing

### VPI & SU

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list **yes**
    a. how many words in list? **418**
2. is a controlled vocabulary used? **no**
3. stemming **no**
4. term weighting
    **Vector and p-norm runs were done with no term weights. Vector runs were also performed with aug_norm * idf weighting.**
5. phrase discovery **no**
6. syntactic parsing **no**
7. word sense disambiguation **no**
8. heuristic associations **no**
9. spelling checking (with manual correction) **no**
10. spelling correction **no**
11. proper noun identification algorithm **As provided in SMART**
12. tokenizer (recognizes dates, phone numbers, common patterns) **As provided in SMART**
13. are the manually-indexed terms used? **not used as suggested in guidelines**
14. other techniques used to build data structures (brief description)
    **1983 version of SMART, enhanced with VPI&SU routines**

B. Statistics on data structures built from TREC text (please fill out each applicable section)
**Except if you want us to answer under 4 here re the knowledge base used to help build our Boolean queries, please advise.**

5. other data structures built from TREC text (what?) **Document vector file and term dictionary**
    a. total amount of storage (megabytes)
        **Approx. 15MB for the dictionary and 121 MB for the Document vector file for the entire Wall Street Journal collection.**
    b. total computer time to build (approximate number of hours)
        **Approx. time to build above 10 hours on ccrd1 (DECstation 5000 Model 25, i.e., a MIPS R3000 chip running at 25MHz)**
    c. is the process completely automatic? **yes**
    d. brief description of methods used
        **The document text is tokenized, stop words are thrown out, and non-noise words are kept in the term dictionary along with its occurrence frequency. Each term in the dictionary has a unique identification number. The vector file contains for each document its unique ID, and a vector of term ID and weights for the term. The weighting scheme is flexible and can be changed to one of several schemes after the indexing is complete. (If necessary we can fill in details here. Please advise.)**

C. Data built from sources other than the input text --no

II. Query construction

(please fill out a section for each query construction method used)

A. Automatically built queries (ad hoc)

1. topic fields used **Description, Narrative, and Concepts.**
2. total computer time to build query (cpu seconds) **Vector queries--50 seconds for 50 topics**
3. which of the following were used?
   a. term weighting with weights based on terms in topics
      **Term weighting was used for vector queries.**
   e. proper noun identification algorithm **As provided in SMART**
   f. tokenizer (recognizes dates, phone numbers, common patterns)
      **As provided in SMART**

B. Manually constructed queries (ad hoc)

1. topic fields used **Description, Narrative, and Concepts.**
2. average time to build query (minutes) **3 mins/query**
3. type of query builder
   b. computer system expert
4. tools used to build query
   b. knowledge base browser (knowledge base described in part I)
      (1) which structure from part I
      **for some of our work we build a knowledge base to help suggest broader/narrower terms--added information can be provided if appropriate**
   c. other lexical tools (identify) **vi (editor)**
5. which of the following were used?
   b. Boolean connectors (AND, OR, NOT)
   d. addition of terms not included in topic
      (1) source of terms **domain knowledge of experts**

III. Searching

A. Total computer time to search (cpu seconds)

**Approx. 4 minutes for each topic.**
**We did a full sequential pass through documents for this since we did not have enough disk space for the inverted file.**

1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)
2. ranking time (total cpu seconds to sort document list)

B. Which methods best describe your machine searching methods?

**Methods: We used three main methods, and a scheme for combining the results from those runs**

1. vector space model
5. Boolean matching
6. fuzzy logic (include your definition) **p-norm matching**

C. What factors are included in your ranking?

**We used several weighting methods in combination with the methods, to get a total of 8 runs that were the basis for our submission. We used binary weights, as well as:**

1. term frequency
2. inverse document frequency

3. other term weights (where do they come from?)
**aug_norm, computed by SMART using the above factors**

IV. What machine did you conduct the TREC experiment on? **DECstation 5000 Model 25**
How much RAM did it have? **40M bytes**
What was the clock rate of the CPU? **MIPS R3000 at 25MHz**
**At the end of our work for the submission, we finally had 3Ghytes of disk storage to work with.**

V. Some systems are research prototypes and others are commercial.
To help compare these systems:

1. How much "software engineering" went into the development of your system?
**We began with the 1983 version of SMART, and have enhanced it. We tried to use the new version of SMART on an RS/6000 hut could not get reliable results and so went back to our older version. We underwent extensive software development since May but due to lack to disk space could not use most of what we developed for the submission.**
2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?
**With more disk space we could have used the inverted file option and that would have made things much faster. That would have allowed real time interactive searching. Also, with more disk space, we could have used an RS/6000, assuming SMART could be ported and made fully operational.**
3. What features is your system missing that it would benefit by if it had them?
**Because of the disk space problem we were not able to do many of the efforts we wanted to do. Work will continue this fall if disks are received in time. Among the features:**
   - **phrase identification and matching**
   - **building "decision trees" after training with a sufficient set of relevance judgments**
   - **implementing the CEO model of P. Thompson and trying it out in a variety of ways to combine results from a variety of runs and indexing schemes (that could include stemming and/or morphological analysis).**

## System Summary and Timing

### GTE Laboratories

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list
    a. how many words in list? **280 words**
2. is a controlled vocabulary used? **no**
3. stemming
    a. standard stemming algorithms
        which ones? **Paice conflation**
    b. morphological analysis **no**
4. term weighting **yes**
5. phrase discovery **no**
6. syntactic parsing **no**
7. word sense disambiguation **no**
8. heuristic associations **no**
9. spelling checking (with manual correction) **no**
10. spelling correction **no**
11. proper noun identification algorithm **no**
12. tokenizer (recognizes dates, phone numbers, common patterns) **no**
13. are the manually-indexed terms used? **no**
14. other techniques used to build data structures (brief description)

B. Statistics on data structures built from TREC text (please fill out each applicable section)

1. inverted index
    a. total amount of storage (megabytes) **3360 (for the 2400 MB of text)**
    b. total computer time to build (approximate number of hours) **672**
    c. is the process completely automatic? **yes**
    d. are term positions within documents stored? **yes**
    e. single terms only? **yes**
5. other data structures built from TREC text (what?) **statistics files**
    a. total amount of storage (megabytes) **400**
    b. total computer time to build (approximate number of hours) **24**
    c. is the process completely automatic? **yes**
    d. brief description of methods used
        **Index is scanned for frequency, location, popularity and record size statistics. The results are used in normalizing the weighting attributes.**

C. Data built from sources other than the input text **--no**

II. Query construction
(please fill out a section for each query construction method used)

A. Automatically built queries (ad hoc)

1. topic fields used **Topic, Description, Narrative**
2. total computer time to build query (cpu seconds) **2 seconds**
3. which of the following were used?
    a. term weighting with weights based on terms in topics
    c. syntactic parsing of topics
    i. automatic addition of Boolean connectors or proximity operators

D. Automatically built queries (routing)

1. topic fields used **Topic, Description, Narrative**
2. total computer time to build query (cpu seconds) **2 seconds**
3. which of the following were used in building the query?
    a. terms selected from
        (1) topic
    b. term weighting
        (1) with weights based on terms in topics
    d. syntactic parsing
        (1) of topics
    j. automatic addition of Boolean connectors or proximity operators
        (1) using information from the topics

III. Searching

A. Total computer time to search (cpu seconds) **/* all topics */ 108000**

1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained) **~72000**
2. ranking time (total cpu seconds to sort document list) **~36000**

B. Which methods best describe your machine searching methods?

10. other (describe) **multi-level attribute weighting**

C. What factors are included in your ranking?

1. term frequency
2. inverse document frequency
3. other term weights (where do they come from?) **explicit term weighting by user**
5. position in document
7. proximity of terms
9. document length
10. completeness (what % of the query terms are present)
15. other (specify) **record (document) id**

IV. What machine did you conduct the TREC experiment on? **IBM RS/6000 320**

How much RAM did it have? **32 MB**

What was the clock rate of the CPU? **25 MHz**

V. Some systems are research prototypes and others are commercial.

To help compare these systems:

1. How much "software engineering" went into the development of your system?
    **This is a prototype.**
2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?
    **yes, given faster hardware and more RAM, we can probably double the**

**performance.**

3. What features is your system missing that it would benefit by if it had them?
   **Variable sized buckets to implement linked lists.**
   **Improved ranking attribute range calculation.**
   **Spelling correction.**

## System Summary and Timing

Siemens Corporate Research, Inc.

General Comments

The timings should be the time to replicate runs from scratch, not including trial runs, etc. The times should also be reasonably accurate. This sometimes will be difficult, such as getting total time for document indexing of huge text sections, or manually building a knowledge base. Please do your best.

**Summary of method: Completely automatic vector matching where both document and query vectors have been expanded using synonyms extracted from WordNet.**

I. Construction of indices, knowledge bases, and other data structures (please describe all data structures that your system needs for searching)

A. Which of the following were used to build your data structures?

1. stopword list
   a. how many words in list?
      **571 word stopword list used (standard SMART stopword list)**
2. is a controlled vocabulary used? **no**
3. stemming
   a. standard stemming algorithms
      which ones?
   b. morphological analysis
      **Extremely simple suffix stripper to look words up in WordNet. (Checks for one of 22 suffixes and possibly modifies end of stem if a matching suffix is found. This was in code I inherited--I don't know the source of the suffix list, but the list is a subset of that used by SMART, so it probably comes from some "standard" algorithm.) All words also pass through the "triestem" stemmer of SMART. This stemmer was originally based on Lovin's CACM article, but has evolved over the years.**
4. term weighting
   **A tf*idf weight is used for both query and document terms, where the weight is further normalized so that an inner product computation produces the cosine ("tfc" weights using the terminology of "Term Weighting Approaches in Automatic Text Retrieval" by Salton and Buckley). A term is counted as appearing in a document (for idf purposes) if it was in the original text or if it was added as a synonym. The tf*idf portion of an added term's weight is multiplied by .8 to produce its final weight.**
5. phrase discovery
   a. what kind of phrase?
   b. using statistical methods
   c. using syntactic methods
      **WordNet contains collocations as members of synonym sets, so some phrases may be added as synonyms. However, such a collocation is assigned a unique concept number and will only match that exact collocation (so I don't consider it to be "phrasing"). No other phrasing used.**
6. syntactic parsing **no**
7. word sense disambiguation
   **No specific sense disambiguation procedure used. If a term occurs in more than one WordNet synonym set (which, by definition, means that it is polysemous), the synonyms from all of its senses may potentially be added to the vector. The**

**algorithm requires that at least two original text words agree on a synonym before it is added to the vector.**

**The effect of this is to do a poor man's version of sense disambiguation for the synonyms.**

8. heuristic associations
    a. short definition of these associations
        **WordNet synonymy relation only association used.**
9. spelling checking (with manual correction) **no**
10. spelling correction **no**
11. proper noun identification algorithm **no**
12. tokenizer (recognizes dates, phone numbers, common patterns) **no**
13. are the manually-indexed terms used? **no**
14. other techniques used to build data structures (brief description) **no**

B. Statistics on data structures built from TREC text (please fill out each applicable section)
1. inverted index
    a. total amount of storage (megabytes) **947 megabytes of disk storage**
    b. total computer time to build (approximate number of hours)
        **5 hours to build index given document vectors; document vectors took 37 hours to build from text. Thus, approximately 42 hours to go from text to inverted index.**
    c. is the process completely automatic? **yes**
    d. are term positions within documents stored?
        **No term position information maintained.**
    e. single terms only?
        **Single terms only (although, as stated above, a single term from WordNet may be a collocation such as 'electrical_discharge').**
2. n-grams, suffix arrays, signature files
    **N-grams and signature files not used. SMART stemmer algorithm incorporates a (static) trie of suffixes.**
3. knowledge bases
    **No knowledge base used other than WordNet (described under I.C.2).**

C. Data built from sources other than the input text
1. internally-built auxiliary files **none**
2. externally-built auxiliary file
    a. type of file (Treebank, WordNet, etc.) **WordNet (noun portion only)**
    b. total amount of storage (megabytes) **5 megabytes**
    c. total number of concepts represented **35,155 synonym sets (67,293 word senses)**
    d. type of representation (frames, semantic nets, rules, etc.)
        **We used only the synonymy relation that WordNet contains. However, WordNet contains many other lexical relationships making it similar to a semantic net.**

II. Query construction

(please fill out a section for each query construction method used)

**[We submitted one set of results; those results were for automatically built ad hoc queries.]**

A. Automatically built queries (ad hoc)
1. topic fields used
    **Concepts (<con>), Description (<desc>), Factors (<fac>), Narrative (<narr>), Nationality (<nat>), Title (<title>)**
2. total computer time to build query (cpu seconds)
    **1 second, on average (50 seconds to index 50 queries)**

3. which of the following were used?

a. term weighting with weights based on terms in topics **Yes, as described above**

d. word sense disambiguation

**Only as described above (two original query terms must agree on a synonym to be added).**

h. expansion of queries using previously-constructed data structure (from part I)

(1) which structure? **WordNet.**

III. Searching

A. Total computer time to search (cpu seconds)

1. retrieval time (total cpu seconds between when a query enters the system until a list of document numbers are obtained)

**15 cpu seconds, on average (756.4 cpu seconds to process 50 queries)**

2. ranking time (total cpu seconds to sort document list)

**not applicable: list of top 200 similarities maintained while searching**

B. Which methods best describe your machine searching methods?

1. vector space model

C. What factors are included in your ranking?

1. term frequency
2. inverse document frequency
4. semantic closeness (as in semantic net distance) **(synonyms)**
9. document length
13. word sense frequency

**(nouns with only one sense in WordNet get all their synonyms added)**

IV. What machine did you conduct the TREC experiment on? **Sun IPX**

How much RAM did it have? **64 megabytes**

What was the clock rate of the CPU? **40MHz**

V. Some systems are research prototypes and others are commercial.

To help compare these systems:

1. How much "software engineering" went into the development of your system?

**Our system is a version of SMART with modified indexing code. SMART has been well-engineered (but its main goal is flexibility, not raw speed). Little time was spent optimizing our modifications.**

2. Given appropriate resources, could your system be made to run faster? By how much (estimate)?

**SMART could probably be made to run somewhat faster if it were made less flexible, that is, if we coded a version that performed only the sorts of runs we made here. I doubt the difference would be dramatic. Preprocessing steps performed on WordNet could improve the efficiency of the expansion code.**

3. What features is your system missing that it would benefit by if it had them?

**Incorporating part-of-speech tagging so that we could know if the term is a noun before looking it up in WordNet should be beneficial (we didn't do this for TREC because the tagger we have is fairly slow). In the same vein, a true sense disambiguator--a way of picking the correct WordNet synonym set--would clearly help, but I don't know of a way of doing that automatically yet (it is part of our research).**



*U.S. G.P.O.:1993-341-931:82636

## ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SYSTEMS TECHNOLOGY

Superintendent of Documents
Government Printing Office
Washington, DC 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Institute of Standards and Technology Special Publication 500-.

Name _____

Company _____

Address _____

City _____ State _____ Zip Code _____

**(Notification key N-503)**

# NIST Technical Publications

## Periodical

**Journal of Research of the National Institute of Standards and Technology**—Reports NIST research and development in those disciplines of the physical and engineering sciences in which the Institute is active. These include physics, chemistry, engineering, mathematics, and computer sciences. Papers cover a broad range of subjects, with major emphasis on measurement methodology and the basic technology underlying standardization. Also included from time to time are survey articles on topics closely related to the Institute's technical and scientific programs. Issued six times a year.

## Nonperiodicals

**Monographs**—Major contributions to the technical literature on various subjects related to the Institute's scientific and technical activities.

**Handbooks**—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

**Special Publications**—Include proceedings of conferences sponsored by NIST, NIST annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

**Applied Mathematics Series**—Mathematical tables, manuals, and studies of special interest to physicists, engineers, chemists, biologists, mathematicians, computer programmers, and others engaged in scientific and technical work.

**National Standard Reference Data Series**—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a worldwide program coordinated by NIST under the authority of the National Standard Data Act (Public Law 90-396). NOTE: The Journal of Physical and Chemical Reference Data (JPCRD) is published bimonthly for NIST by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements are available from ACS, 1155 Sixteenth St., NW, Washington, DC 20056.

**Building Science Series**—Disseminates technical information developed at the Institute on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

**Technical Notes**—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NIST under the sponsorship of other government agencies.

**Voluntary Product Standards**—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The standards establish nationally recognized requirements for products, and provide all concerned interests with a basis for common understanding of the characteristics of the products. NIST administers this program in support of the efforts of private-sector standardizing organizations.

**Consumer Information Series**—Practical information, based on NIST research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

*Order the* **above** *NIST publications from: Superintendent of Documents, Government Printing Office, Washington, DC 20402.*

*Order the* **following** *NIST publications—FIPS and NISTIRs—from the National Technical Information Service, Springfield, VA 22161.*

**Federal Information Processing Standards Publications (FIPS PUB)**—Publications in this series collectively constitute the Federal Information Processing Standards Register. The Register serves as the official source of information in the Federal Government regarding standards issued by NIST pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

**NIST Interagency Reports (NISTIR)**—A special series of interim or final reports on work performed by NIST for outside sponsors (both government and non-government). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Service, Springfield, VA 22161, in paper copy or microfiche form.

**U.S. Department of Commerce**
National Institute of Standards and Technology
Gaithersburg, MD 20899