RJ 10063 (90560) December 4, 1996
Computer Science

Research Report

SUBJECT-BASED SEARCHING USING AUTOMATICALLY EXTRACTED METADATA

Thomas Kirsche
Rob Barrett

IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

IBM Research Division
Yorktown Heights, New York • San Jose, California • Zurich, Switzerland

RJ 10063 (90560) December 4, 1996
Computer Science

SUBJECT-BASED SEARCHING USING AUTOMATICALLY EXTRACTED METADATA

Thomas Kirsche [1]
Rob Barrett

IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, California 95120-6099

ABSTRACT: Search for text documents is usually based on keywords. However, the usefulness of a document is also based on other characteristics such as writing style, language, and subject. The subject is a characteristic of paramount importance, because the use of synonyms might produce unwanted results with keyword-based searching alone. Vice versa, if the subject of a query and a candidate document is determined correctly, subject-based searching could help to identify the most useful documents.

This paper describes the architecture of a working prototype that is capable of determining automatically the subjects of text documents and queries. Based on the judgement, queries are matched with document collections that have a high probability of providing useful data. As the experimental results indicate, the AIM subject prototype could narrow search space to 2% of its original size and still obtain roughly half of the relevant documents.

---

[1] On leave from U Erlangen-Nuremberg, Germany

Part I: The scientific viewpoint

1 Introduction and rationale behind AIM

Fast and reliable retrieval for structured data that is maintained in a database management system has become a commodity over the recent years. Various indexing techniques provide fast access to the data records, and the known structure of the data lets the database system reliably decide whether the data will match a search criterion or not. Searching for text documents like news articles is a different business, because this data is unstructured and in general not maintained at a central data base. The AIM project is an effort to make available more sophisticated search mechanisms for text documents that reside in different, loosely coupled data repositories. AIM's general approach is to automatically extract meta data from documents and queries and use it for a better query processing. AIM is an acronym for Automatic Informative Metadata.

As far a loosely coupled information systems are concerned, the World Wide Web (WWW) has become the protagonist with respect to variety, quantity, distribution, and last not least public attention. Sources estimated some 5,000,000 on-line text documents ([CMU95a]) and some 40,000 servers ([Amer95]) in mid 1995. WWW servers could be viewed as independent repositories, although their current searching capabilities are limited. However, more and more servers offer a search function that covers all documents on that particular server. Content-based searching for documents in the whole WWW, i.e. searching more than one server, is not supported by the WWW. Besides "knowing" a document's uniform resource locator (URL) in advance, the WWW's native access path to documents is following hypertext links from one document to another document. One of AIM's objectives is to support search in distributed information systems like the WWW. As we will explain, AIM is narrows down search space to sources that have a high probability of carrying the sought information. Given a certain query, this capability could be used to select WWW servers offering the "right" documents and to exclude servers that are not likely to carry documents that match the query.

As far as searching of text documents is concerned, information retrieval academia and vendors of text retrieval products have contributed a lot over the last years. In general, however, all approaches focus on a keyword search that is maybe beefed up with a thesaurus for synonyms and an ontology for searching. Thus, the only used characteristic of a document (besides its keywords) is the language it is written in. Other characteristics of the text document, such as subject, writing style, and profession (classes of vocabulary) are hardly used. AIM analyses this meta data and uses it to compute a document's usefulness with respect to a query. The point is that all analysis is done automatically. For example, the writing style is computed as a function of the words belonging to complete sentences, percentage of sentences that end in a question mark, and the Kincaid reading level which is itself a function of the syllables per word and the number of words per sentence. To decide usefulness, relevance feedback from previous queries is used. Analysis of other meta data, as well as a general overview of AIM is described in [BaSe95].

The subject of a text is one characteristic that is considered of pivotal value in reducing the search space. For example, a query "Give me all documents addressing 'mice'" could be related to the subject of rodent biology or associated with computer pointing devices. If the retrieval system succeeds in determining the user intended subject, document collections on either biology or computers could be disregarded. Obviously, search space could be drastically decreased without

cutting out relevant documents. Of the other characteristics, only "language" has such a high degree of selectivity on the search space. Writing style and profession fall behind in selectivity.

Because of the paramount importance of the "subject" characteristic, a prototype was built to automatically determine the subject of documents on queries. Using this prototype, documents are put off-line into subject bins. A query is processed on-line for its subject and then redirected to their corresponding subject bin(s). Only documents in these bins are searched, for example, using conventional information retrieval techniques. The very same technique could be used to facilitate search in the WWW. Think of assigning WWW servers to subject bins. A WWW server carries a number of documents which cover a number of subject. Thus, a WWW server would probably show up in a couple of bins. A query looking for WWW documents could now be forwarded only to those servers that cover the subject of the query. AIM is the enabling technology to do the query forwarding.

In this report, the design and implementation of the AIM subject prototype is described. Part I is entitled the scientific viewpoint and discusses the architecture of this prototype (section 2). In section 3 the results of our experiments with this prototype are summarized. A related idea on how to provide support for searching the WWW is explained in section 4. Some loosely related work is the focus of section 5. Part II is entitled the implementation viewpoint. It is written as a reference manual for working with AIM and starts with an overview of AIM's components (section 6). Section 7 is about setting up the AIM system and section 8 discusses the code base and compilation questions.

2 AIM subject prototype

Although the AIM subject prototype is a complete text retrieval system, it is focused on the determination of subjects in queries and documents. Using the subject tags, queries are forwarded to collections of documents under the same subject tag. Thus, the probability of retrieving useful documents stays at the same level but the number of processed documents decreases dramatically, opposed to information systems employing a homogeneous document collection. In database terminology, AIM could be viewed as a subject-based index for documents. Section 2.1 explains AIM's basic subject determination technique. Next comes a description of AIM's overall architecture (section 2.2). Eventually, its data and control flow is detailed in section 2.3.

2.1 Subject determination

In the introduction, a sample query after "mice" was given that could relate to both to rodents and to computer pointing devices. Of course, it is not possible to judge from this simple word "mice" what subject would be more appropriate. This word is a used synonymously and thus ambiguously for two different purposes. AIM cannot do much about this. Assume now that a query consists of two words, like "mice teeth" or "mice cable". Obviously, the subject of the first query is probably more related to rodents whereas the second query is probably more related to computer hardware. The rationale behind this judgement comes from the observation that the combination of the words "mice" and "teeth" indicates more likely a biological subject than a computer-related topic.

Just as a human being would act, AIM also determines the subject of a text by looking at the words and their relative frequency within the collection. Figure 1 depicts the whole process. We call the process of assigning subjects to documents "tagging".

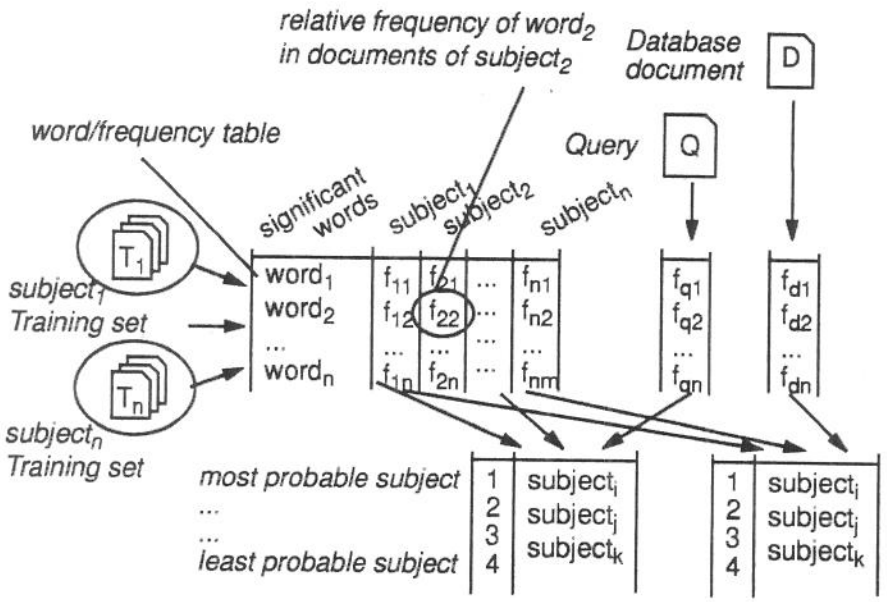

Figure 1. Subject determination (tagging)

Tagging starts with a collection of documents that is known as the training set. The training set is arranged into groups of documents that are considered to be of the same subject. In a first step, AIM builds a list of all significant words $word_1$ to $word_n$ that appear in the training set. Significant words do not include common words like pronous, prepositions, and other stopwords. The resulting vector of words is the union of the significant words found in all training documents. The second step determines the relative frequency of words for each subject separately. A vector of frequencies f_{i1} to f_{in} is computed for every $subject_i$ on the basis of the documents assumed to be of $subject_i$. Both steps together will produce a table (relation/matrix) whose columns reflect the usage of significant words within the subject. The word/frequency table is the pivotal reference for determining the subject of future documents.

The tagging of any text, whether it is a query or a document of the database, requires to create a word/frequency vector. Only words from the previously computed word/frequency table are used, which will hopefully have a fair overlap to the words of the document. In order to determine the subject of the text, the text-specific word/frequency vector is compared to word/frequency vectors of all subjects. Based on similarity of the vectors, a ranked list of subjects is produced. Subjects whose vectors are very similar to the text vector are on top of this subject list, subjects with a lower similarity are at the end. The first entry in the subject list is regarded the most probable subject of the document or query, respectively. Although a subject list could have up to n entries (if all subjects were ranked), it is reasonable to throw away subject below a certain threshold value. Depending on the type of a text, this threshold value is later called number of query subjects or document subjects (section 3.4).

As far as vector similarity is concerned, a great variety of distance metrics is available: cosine, asymmetric, Dice, Jaccard, Euclidian, and so on. The AIM subject prototype currently uses a cosine coefficient which seems to produce reasonable results. With this metric the distance between two vectors v_1 and v_2 is computed as the cosine of their angle α, where $\cos \alpha = (v_1 * v_2)/(|v_1| * |v_2|)$. The distance metrics are described in ([SaMc83]).

A crucial point for tagging is the selection of subjects and the training data for each subject. In the AIM subject prototype, the usenet news group hierarchy provides both the ontology and the training documents. The news groups could be viewed as a hierarchical ontology of subjects. Top subjects are "comp" (computers) or "rec" (recreation), for example. Second level subjects include "comp.databases" or "rec.aviation", and so on. This subject ontology is ready to use and has proven to cover a good part of computer-readable texts in the past. Most convenient in using news groups as subjects is that this ontology comes for free with training documents. These are the news articles posted to the respective newsgroups. For training, we select only the documents that are posted to only one news group, which results in a more specific characterization of the subjects/news group. The AIM subject prototype currently uses 1431 subjects that are only derived from news groups. Together with some 60,000 words, the size of the word/frequency table is around 38 MB.

2.2 Architecture

The assignment of subjects to textual documents or queries as described in the previous section 2.1 is regarded the core task of the AIM prototype. In order to use this technology for a working text retrieval engine, additional tasks have to be tackled. This section introduces the other components of the search engine and outlines the architecture of the prototype. A major design guideline was to build a distributed system that could eventually be promoted to a system useful in loosely coupled, WWW-like environments. A client/server type architecture comes handy to achieve that goal. In the sequel, we will introduce the following components:

1) indexer server
2) router server
3) tagger server
4) locator client
5) query client

On the server side, a first component has to deal with the actual document storage. This *indexer* is expected to store documents and to later retrieve them by some document id or as the result of the query. Of course, the scope of a query is limited to the documents of that indexer. A database system enabled for text data types or a general purpose text retrieval engine is such a document server. A standard HTTP server is not, because current servers lack sufficient support for searching. i.e. they are not able to retrieve documents fulfilling some given query criteria. Advanced HTTP servers that provide native support for text search or simulate searching via helper applications (cgi scripts), however, are candidates for document storage in the AIM subject prototype.

A reasonable assumption is that AIM cannot control what documents are stored in which document storage component. Like in the WWW, documents are added arbitrarily to the databases. Because of the number of document storages and their passive behavior, AIM could not

even keep track of the additions and deletions. As in the WWW AIM must not assume a document registry. We can assume, however, that the document databases cover only a relatively small number of subjects. Most of the existing document collections are dedicated to certain topis, like "computer science technical reports", "customer complaints", and so on. Although documents may get added to or removed from a database with a moderate or high rate, the subjects of a database will only change occasionally. To monitor theses subject changes, infrequent visits to a database are sufficient. Nevertheless, a component is needed that maintains a list of document databases and what subjects they cover. We call this component *router* because of its ability to route subjects to indexers. The list is called a routing table. Assuming that a database can be identified on the average with 30 characters (cf to IP host names), and one subject number fits into 2 bytes, a 100 byte entry would cover 35 subjects for each database. Referring again to the estimate of 40,000 WWW servers ([Amer95]), a router must store a 4 MB list in this set-up, which is clearly feasible even with low-end machines. The number of subjects per database is a parameter of the router.

The final server component is called *tagger*, because it tags documents or queries with subject tags. Tagging is based on the creation of a word/frequency table and the distance between text vectors and subject vectors. The tagging technique was explained in detail in section 2.1. Figure 2 gives an illustration of servers and clients in the AIM subject prototype.

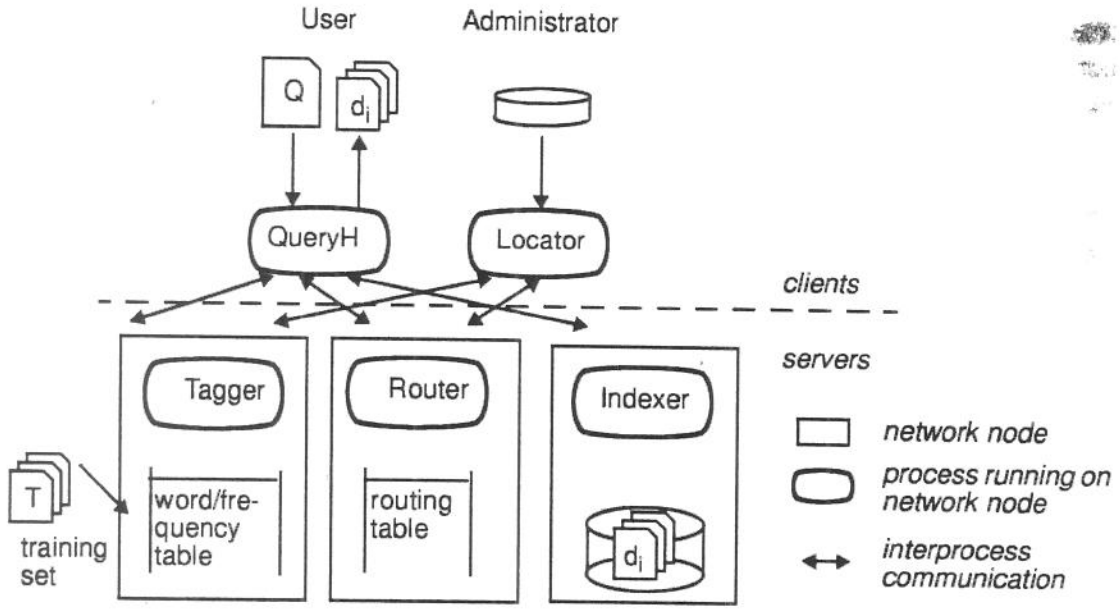

Figure 2. Architecture of the AIM subject prototype

On the client side, AIM must first have an administrator interface to monitor database. This component allows to locate new document collections in AIM and is therefore called *locator* client. As the database symbol above the locator indicates, the locator takes the identifier of a document collection as an input. All or some representative documents of this collection are processed in order to determine the subjects maintained at this collection. The findings are stored in the routing table. For that purpose, the locator needs to communicate with tagger and router. It does not communicate with the indexer, because the data is already present in the document collection.

Eventually, a component is needed that handles queries and to controls query processing as a whole. We call this the *query handler* client. First of all, it allows the user to input a query. Second, query processing involves communication with the tagger, router, and indexer servers. The query handler sends out messages to these servers and receives their answers. Because the servers do not communicate directly, the query handler acts as a mediator. Even the user may get involved again during query processing. For example, she might give relevance feedback on the selected subjects or the number of selected databases, before the query is actually executed. If the query succeeds in finding documents, the results are displayed as document ids at the query handler. Displaying the actual documents is not part the AIM subject prototype.

The client/server architecture of the prototype allows for replication of both servers and (of course) clients. An arbitrary number of taggers may be used as long as the word/frequency table remains the same for all of them. The same is true for the router, as long as the replicas of the routing table are kept consistent. With more than one router or tagger, clients may distribute the work load on more servers. Replicating indexers is beyond the scope of the AIM subject prototype. The number of indexers is very high which makes the distribution of queries on different indexers very likely to happen, anyway.

2.3 Control flow/data flow in the working prototype

The previous section gave a rough idea on the tasks involved in the AIM subject prototype. In this section, the course of processing is explained in more detail. We also explain the difference between idealistic architecture of section 2.2 and reality. Again, a figure is used as an illustration of the whole process (figure 3). Data and control flow are directed from left to right.

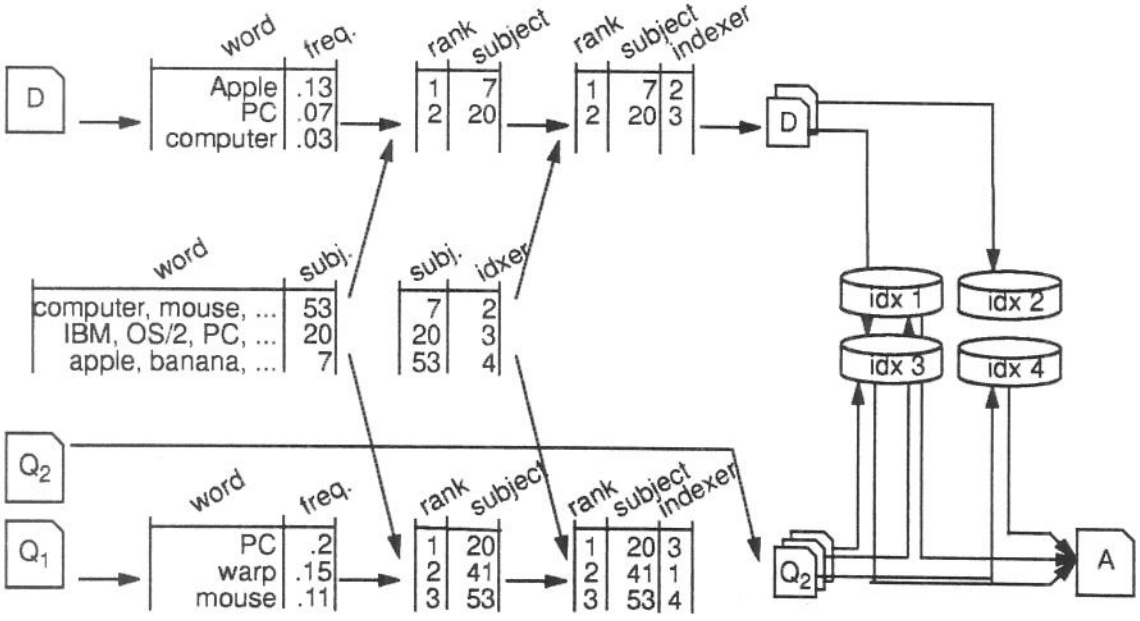

Figure 3. Basic data flow in AIM

Let's start with the query processing part in the working prototype. It is pretty close to the original architecture, except that the prototype takes two queries Q_1 and Q_2 (which may be identical, of course). By intention the first query is used to select document collections (indexers), and

the second query is run against the databases. To select indexers, first a word/frequency vector is computed for Q_1. In the example of figure 3, 20% of the query's words were "PC", 15% "warp", 11% "mouse" (the remaining 54% are not shown). This vector is compared to the word/frequency table (section 2.1) and the most similar vectors are compiled into a ranked list of subjects. The most probable subject for Q_1 is #20 which is about "IBM, OS/2, PC". Next comes the selection of indexers. Using the subject numbers as a key for the routing table, the router will look up the identifiers of indexers carrying the subject. Referring again to the example, indexer #3 covers subject #20. The same is done for the other entries on the ranked list. The length of the list is specified in the query subject parameter (section 3.4) and is set to 3 in the example. Once the indexers are selected, the set of indexers is presented to the user. If she agrees, Q_2 is loaded and run against the selected indexers #1, #3, and #4. Indexer #2 was not found of carrying useful subjects with respect to the query and is therefore spared. Finally, the duplicate documents are eliminated and one list of document names is presented as the answer to the query.

Different from the architectural picture (section 2.2), documents instead of database names are located in the initial phase of running the working prototype. In the prototype, we don't start with ready-to-use document collections, but load the documents into a given number of indexers. The selection of the indexers is quite similar as in query processing. It is driven by the locator client. Again, a word/frequency vector is created for a document D, and a ranked subject list is built according to vector similarity. The document ranking may operate with a different cut-off value than query subject ranking. For example, the number of document subjects is set to 2 which means that only the first and the second ranked subjects (#7 and #20) are used in further processing. With the help of the routing table, the documents are forwarded to indexers #2 and #3. That's why there is communication between locator and indexers, opposed to the idealistic architecture. At the indexer, the document is added to the existing collection. In the example, document D is forwarded to indexers #2 and #3, because these two indexers carry the two top ranked subjects #7 and #20, respectively.

Regarding the differences between architecture and working prototype, one more comment is appropriate. In the implementation, the actual indexers rely on a underlying document database system that does the insertion and retrieval of text documents. AIM's true indexers are degraded to wrappers around the document database. Their only job is to provide an interface to database's query language. Although any text retrieval engine might be used with AIM, IBM's SearchManager product in the alternative of choice in the current prototype.

3 Experimental results

This section summarizes the results of the experiments carried out on the AIM prototype. Section 3.1 briefly explains what has be measured. In part, the TREC document and query set was used for the experiments. Section 3.2 comments on TREC. Section 3.3 provides results for AIM's underlying text retrieval engine, IBM's SearchManager. The actual set up for the experiment is described in section 3.4, and the necessary adaptations to the metrics are explained in section 3.5. Finally, section 3.6 shows the results for the prototype.

3.1 Simple metrics

Information retrieval has come up with a number of metrics on how to measure the "quality" of retrieval algorithms and techniques. The most common are recall and precision, as defined by Salton ([SaMc83]):

$$\text{recall (q)} = \frac{\text{\# of relevant documents obtained for query q}}{\text{total \# of relevant documents for query q}} \qquad \text{precision (q)} = \frac{\text{\# of relevant documents obtained for query q}}{\text{total \# of retrieved documents for query q}}$$

Both metrics return values between 0 and 1. If recall is 1, all relevant document have been found. A precision value of 1 indicates that only the relevant documents have been obtained. To compute recall and precision, the total number of relevant documents for a given query q must be known in advance. This number is the denominator of the recall value and used as well to determine documents that are obtained and relevant (nominator of recall and precision). The denominator of precision is simply the cardinality of the result document set.

Both recall and precision are defined with respect to a given query, not for a given algorithm or product. To measure a technique, the query dependent values have to condensed into recall and precision of the information retrieval system as a whole. Lacking a more sophisticated process, we will take the arithmetic average of the query dependent values as the recall or precision of the whole system, respectively. Let Q be the set of queries q in the experiment:

$$\text{recall (system)} = \frac{\sum_{q \in Q} \text{recall}(q)}{|Q|} \qquad \text{precision (system)} = \frac{\sum_{q \in Q} \text{precision}(q)}{|Q|}$$

To make it crystal clear, the actual recall and precision values are dependent on the selection of the queries and the underlying document set. Variations in queries and document set may produce different results.

3.2 TREC data set

To test the AIM prototype, both documents and queries along with information on what documents are considered relevant was needed. The TREC data set is a commonly available source with the above characteristics. The objectives of TREC are best described by TREC itself:

> "The Text REtrieval Conference (TREC) is sponsored by the National Institute of Standards and Technology (NIST) and the Advanced Research Projects Agency (ARPA). The group meets annually to discuss and present their work.
>
> The goal of the conference is to bring research groups together to discuss their work on a new large test collection. The participants use a wide variety of retrieval techniques, including methods using automatic thesauri, sophisticated term weighting, natural language techniques, relevance feedback, and advanced pattern matching. Results are run through a common evaluation package in order for the groups to compare the effectiveness of different techniques and to discuss how differences between the systems affected performance." ([NIST95])

The full set of TREC data consists of older newspaper articles from Associated Press, Wall Street Journal, and so on and has approximately 3 GB spread over 1,000,000 documents. A couple of hundreds of documents are usually compiled into one file. Using a hypertext markup language similar to HTML, the documents are separated in such parts as document identification, headers, datelines, and the actual text. This is an excerpt of one document named AP890115-0006:

```
<DOC>
<DOCNO> AP890115-0006 </DOCNO>
<FILEID>AP-NR-01-15-89 1004EST</FILEID>
<FIRST>u i BC-Czechoslovakia 01-15 0322</FIRST>
<SECOND>BC-Czechoslovakia,0333</SECOND>
<HEAD>Czechoslovak Police Push Back Hundreds at Rally</HEAD>
<BYLINE>By TEDDIE WEYR</BYLINE>
<BYLINE>Associated Press Writer</BYLINE>
<DATELINE>PRAGUE, Czechoslovakia (AP) </DATELINE>
<TEXT> Police armed with batons and water cannons attacked hundreds of
people who staged a rally Sunday to commemorate the 20th anniversary of an
anti-Soviet protest in which Jan Palach committed suicide. At least one
person was badly injured and several others beaten by hundreds of police
...
other independent groups who called it said they would go ahead anyway.
Similar banned rallies in August and October drew thousands of people, who
also were dispersed by riot police.
</TEXT>
</DOC>
```

The TREC data set also includes 100 queries. Again, markups are used to split the queries into the parts like domain, title, description, narrative, concepts, factors, and definitions. This is TREC query #67 which will also be used as example in the subsequent sections.

```
<top>
<head> Tipster Topic Description
<num> Number: 067
<dom> Domain: International Relations
<title> Topic: Politically Motivated Civil Disturbances
<desc> Description:
Document will report a current civil disturbance in any country, involving
citizens of that country protesting a political position of their own
country's government.

<narr> Narrative:
A relevant document will report the location of the disturbance, the iden-
tity of the group causing the disturbance, the nature of the disturbance,
the identity of the group suppressing the disturbance and the political
goals of the protesters. It should NOT be about economically-motivated
civil disturbances and NOT be about a civil disturbance directed against a
second country.

<con> Concept(s):
1.  protest, unrest, demonstration, march, riot, clash, uprising, rally,
    boycott, sit-in
2.  students, agitators, dissidents
3.  police, riot police, troops, army, National Guard, government forces
4.  NOT economically-motivated

<fac> Factor(s):
<time>  Time:  Current
</fac>
<def> Definition(s):
</top>
```

Most notable is that the TREC queries are natural language queries that come in a "descriptive", a "narrative", and a "conceptual" style. The styles differ in length, expressiveness, conciseness, and wording. The above sample query gives a good understanding of the three query styles. Another characteristic of the TREC query set is that 11 out of 100 queries contain "negative" terms ("NOT economically-motivated"), i.e. they ask for terms that should not appear in relevant documents.

The last part of the TREC query set is a list of queries together with documents. A query/document combination tells that the document is considered relevant for the query. Typically, very few (less than 1%) documents are considered relevant. For the experiments with the AIM subject prototype, only a 700 MB subset of the whole TREC data set was installed.

3.3 TREC and SearchManager results

Before any evaluation of the AIM system takes place, a few paragraphs will demonstrate SearchManager's capabilities for the plain TREC document and query set. AIM employs SearchManager as a text retrieval engine in the indexers. As it will turn out SearchManager is not too well suited for the TREC set, i.e. recall and precision values are low even for fairly small document sets.

To compute SearchManager's recall and precision, the following experiment was conducted. A relatively small set of documents was loaded into only one SearchManager index, and the TREC queries were run against that index. No AIM subject tagging or AIM routing was involved.

The document set in the experiment consisted of 294 news articles totalling in 884,402 bytes (~3kB/document). They were loaded into SearchManager as they were, i.e. all special symbols, hypertext markups, dates, numbers, and so on remained in the documents. SearchManager will skip stop words and other unknown symbols, however.

A number of 6 queries were run against this SearchManager index. According to the TREC specification, each of these queries is supposed to lead to the retrieval of one or two relevant documents. Some documents are relevant for more than one query. too. As explained in section 3.2, TREC queries are natural language queries, and come basically in a "descriptive", "narrative" and a "conceptual" style. The concepts part is the most concise and thus the most valuable source for queries. This made it the alternative of choice in the experiment. To be used by SearchManager, however, the natural language query has to be translated into a sequence of terms and boolean operators. It is not clear off-hand what translation is most suitable for the query, so different translation styles were tested. Using again TREC query #67 for demonstration they are:

- OR translation
  All terms are considered to occur independently, thus all terms are OR'd. For example:

```
protest OR unrest OR demonstration OR march OR riot OR clash OR uprising
OR rally OR boycott OR sit-in
OR students OR agitators OR dissidents
OR police OR riot police OR troops OR army OR National Guard OR government
forces
OR NOT economically-motivated
```

- AND-OR translation
  All terms within a numeric bullet are treated as OR'd terms, the numeric bullets are considered to be AND'd. Because AND has a higher operator precedence, the entries of the numeric bullet lists are enclosed in parentheses. For example:

```
( protest OR unrest OR demonstration OR march OR riot OR clash OR uprising
OR rally OR boycott OR sit-in )
AND ( students OR agitators OR dissidents )
AND ( police OR riot police OR troops OR army OR National Guard OR govern-
ment forces )
OR NOT economically-motivated
```

With all translation styles, terms consisting of multiple words (e.g. "National Guard", "government forces") were kept adjacent to another. For comparison only, a third, somewhat optimal, translation was performed:

- manual AND translation
  All terms appearing in the query but not in the document are removed. The remaining terms are AND'd. For example,

```
march AND troops
```

are the only terms that are found in both in the concepts part of TREC query # 67 and the document considered to be relevant. As a characteristic of this experiment, on an average only 31.86% of the query terms appear in the relevant documents, too. In absolute numbers, the average query has 22.75 words of which only 7.25 words appear in the relevant document. Table 1 shows the retrieval results for the translation techniques described above.

query #	should find	OR translation			AND-OR translation			manual AND translation		
		found on rank # of #	precision	recall	found on rank # of #	precision	recall	found on rank # of #	precision	recall
52	1 doc	error	-	0	0/0	-	0	1/1	1	1
55	1 doc	17/28	.036	1	0/0	-	0	1/1	1	1
62	2 docs	error	-	0	0/0	-	0	3/4; 4/4	.5	1
67	2 docs	error	-	0	error	-	0	6/7; 7/7	.286	1
73	1 doc	33/51	.020	1	0/0	-	0	1/1	1	1
78	1 doc	error	-	0	error	-	0	1/1	1	1

Table 1. Location of relevant documents using different translation techniques

For example, query # 62 should obtain exactly 2 documents of the 294 documents in the experiment. Using the OR translation of the query, SearchManager raised an error and stopped processing. Precision cannot be computed, because the denominator "number of retrieved documents" is zero. Recall is 0, because zero documents were found out of two relevant documents. With the AND-OR translation an empty list of documents was returned and thus no relevant documents were found. Again, precision is non-computable and recall is 0, for the same reasons as above. Using the manual AND translation a list of 4 documents was retrieved, and the entries 3 and 4 of that ranked list were actually the relevant documents. Thus, precision is 2/4 = .5, and recall is 2/2 = 1.

Judging from the high error rates and very low precision and recall values of OR and AND-OR translation in table 1, SearchManager does not well for the TREC query and document set. The relatively good values for the manual AND translation must not be taken into consideration, because in this case the queries were adapted to the documents. They show, however, that SearchManager can do a good job, if presented the "right" queries.

SearchManager serves in AIM as a mere text retrieval engine underlying the indexers. Any other search engine could replace SearchManager. Therefore, it seems reasonable to value the AIM system without taking SearchManager (or any other search engine) into account. To do so, we assume from thereon that indexers can find all relevant documents and only the relevant documents, once a query arrives at the "right" indexers. This routing business is the very core of AIM, and therefore precision and recall of AIM is measured in terms of routing queries to the right indexers. Again, it is assumed that once queries arrive at the right indexer, the indexers eventually retrieve exactly the relevant documents. Vice versa, if AIM directs queries to a wrong indexer, no relevant documents will be retrieved.

3.4 AIM/TREC test set up

After having explained why SearchManager is better left out of consideration, this section describes how the experiments were carried out. The subsequent section 3.5 will focus on what is being measured.

Basically, TREC queries are run against the TREC data set. AIM tries to decrease the search space in a way that all relevant documents are kept, and only non-relevant documents are removed from the search space. The decision whether documents should belong to search space of a given query is based on the subject of the documents and the query. Ideally, only one subject is assigned to a query, and the query looks only at documents of this subject. Assuming that the documents are uniformly distributed over n subjects, AIM could narrow down search space to 1/n in the optimal case and still obtain all relevant documents.

Processing a query in the AIM/TREC set up works as follows. We also explain AIM's operational parameters *number of query subjects* and *document subjects* that specify how far AIM may get away from the ideal one-subject case of the previous paragraph. Again TREC query #67 is run against the same set of 294 documents as in section 3.3. AIM's tagging process maps that query to subjects in descending order from the most probable to the least probable subject. The length of this list is the *query subject number*, i.e. how many subjects should be taken into further consideration. In the example this parameter is set to 4 and thus the list is cut off after the fourth entry.

```
67:1:17 /* clari\news\civil_rights */
67:2:213 /* clari\world\americas\mexico */
67:3:4 /* clari\news\features */
67:4:238 /* clari\world\europe\russia */
```

The first column tells the query number, the second the rank, and the third column the subject number. The name of a subject, a news group name, is stated as a comment. Given the above list, the subject bins are now checked for the documents that match the subject best. The list for subject #17, #213, #4, and #238 looks like this, descending from left to right, top to bottom.

```
AP890630-0097:17:4      AP890630-0161:17:4      AP890630-0063:17:4
AP890630-0186:17:4      AP890630-0197:17:5      AP890630-0045:17:5
```

AP890630-0147:17:5	AP890630-0134:17:5	AP890630-0171:17:6
AP890630-0089:17:6	AP890630-0105:17:6	AP890630-0172:17:6
AP890630-0151:17:7	AP890630-0084:17:7	AP890630-0215:17:8
AP890630-0077:17:8	AP890630-0148:17:10	AP890630-0194:213:3
AP890630-0085:213:5	AP890630-0122:213:5	AP890630-0204:213:6
AP890630-0209:213:6	AP890630-0148:213:6	AP890630-0199:213:7
AP890630-0183:213:9	AP890630-0057:213:9	AP890630-0054:213:9
AP890630-0042:213:10	AP890630-0084:213:10	AP890630-0065:238:1
AP890630-0001:238:3	AP890630-0071:238:3	AP890630-0028:238:3
AP890630-0045:238:3	AP890630-0166:238:3	AP890630-0141:238:5
AP890630-0227:238:5	AP890630-0093:238:6	AP890630-0024:238:6
AP890630-0226:238:6	AP890630-0144:238:6	AP890630-0175:238:7
AP890630-0008:238:7	AP890630-0228:238:7	AP890630-0064:238:9

For every entry, the first column gives the document name, the second the subject number, and the third column the rank on which the document was mapped to the subject. For example, the first entry for document AP890630-0097 was considered to match three subject better than subject #17, and #17 was the fourth best match. The cut-off value for the third column is the *document subject number*, which is set to 10 in the above case. Hence the list contains for each subject only documents with a rank lower or equal than 10. This is different from having 10 documents in the list. Because subject #4 had no documents ranked 10 or less, it does not appear in the above list.

Another interpretation of the query subject number would be the fan-out of a query, i.e. how many indexers get the query. Similarly, the higher the document subject number is, the more indexers will store the document. In general, higher numbers of both parameters will improve recall and worsen precision, together with an increases processing time and storage overhead.

Without using a text retrieval engine, AIM has narrowed down the search space from 294 documents to 42 unique documents. Also, the documents considered to be relevant are in this list (printed in bold face).

3.5 AIM/TREC metrics

In this paragraph the actual formulas of recall and precision in AIM are given. Of course, they are based on Salton's definition (section 3.1), but AIM's two operational parameters number of query subjects q and document subjects d (section 3.4) make an adaptation necessary:

$$\text{recall}_{q,d}(Q) = \frac{\text{\# of relevant documents obtained for query Q wrt q, d}}{\text{total \# of relevant documents for query Q wrt q, d}} = \frac{\sum_{i=1}^{q}\sum_{j=1}^{d} nro_Q(i,j)}{nr_q}$$

$$\text{precision}_{q,d}(Q) = \frac{\text{\# of relevant documents obtained for query Q wrt q, d}}{\text{total \# of retrieved documents for query Q wrt q,d}} = \frac{\sum_{i=1}^{q}\sum_{j=1}^{d} nro_Q(i,j)}{\sum_{i=1}^{q}\sum_{j=1}^{d} no_Q(i,j)}$$

where

Q	is a specific query
q	is the number of query subjects, i.e. the number of subjects considered for the query
d	is the number of document subjects, i.e. the cut-off value for document ranks
$nro_Q(i,j)$	is the number of relevant, j-ranked documents within the i-th query subject
$no_Q(i,j)$	is the number of obtained, j-ranked documents within the i-th query subject
nr_Q	is the total number of relevant documents for Q

Each test query Q is run against a specific set-up of the AIM system. Thus, the number of document subjects d and the number of query subjects q is a parameter of recall and precision. The idea behind the nominators in the recall and precision formulas is to summarize over all obtained and relevant documents that are found following the most probable subject for the query, the second most probable subject and so on until the q most probable subject. For one subject, the summands are themselves a sum over documents that are ranked first, second, and so on down to the d-th rank. Similarly, precision's denominator is a sum over a sum over the number of obtained documents for a specific subject and on a specific document rank. If a document appears more than one time, it is only counted once at its first occurrence.

Referring to the example of section 3.4, we can now compute recall and precision using the newly introduced formulas. In the example, query #67 was run, the number of query subjects q was 4 and the number of document subjects d was 10. Thus, we are actually computing $recall_{4,10}(67)$ and $precision_{4,10}(67)$.

$$\text{recall}_{4,10}(67) = \frac{\sum_{i=1}^{4}\sum_{j=1}^{10} nro_{67}(i,j)}{nr_4} = \frac{1+0+0+1}{2} = 1$$

$$\text{precision}_{4,10}(67) = \frac{\sum_{i=1}^{4}\sum_{j=1}^{10} nro_{67}(i,j)}{\sum_{i=1}^{4}\sum_{j=1}^{10} no_{67}(i,j)} = \frac{1+0+0+1}{17+12+0+16} \approx 0.044$$

The nominator of $recall_{4,10}(67)$ is comprised of 4 summands for each subject #17, #213, #4, and #238. These are the 4 most probable subjects for query #67. Each summand is a sum of 10 other values which state the number of relevant documents for a given subject and document rank. For subject #17 the first value is 1, because in the document list of section 3.4 there are 0 relevant documents on rank 1, 0 on 2, 0 on 3, 0 on 4, 1 on rank 5 (this is AP890630-0045), 0 on 6, and so on. The next relevant document is found under subject #218, but this is again AP890630-0045 and is not counted therefore. Eventually, relevant document AP890630-0008 is found under subject #238 on rank 7. The denominator value of 2 is given by TREC and indicates that there are only two relevant documents for query #67 in document set. Note that only the relevant documents count for recall, not the number of obtained documents. The number of obtained documents is considered in the denominator of $precision_{4,10}(67)$. The search obtained 17 documents with ranks less or equal than 10 for subject #17, 12 for subject #213, 0 for subject #4, and 16 for subject #238. The nominator is computed as for the recall value.

Section 3.1 explained briefly that only the recall and precision values of queries are defined and there is no such commonly-agreed thing as the recall or precision of a whole information retrieval system. The proposed way to get such characteristics for a whole system was to take an average over all query-dependent values. In the above case the $\text{recall}_{4,10}(Q)$ and $\text{precision}_{4,10}(Q)$ values of all queries Q have to be averaged.

3.6 AIM/TREC results

This section presents the detailed results of three experiments. The experiments differ in the number of documents and the number of queries run on the document set. In the upper portion of table 2, an overview on the characteristics of the experiments is given.

experiment name	E294	E2108	E7543
number of documents	294	2108	7543
total size of all documents [bytes]	884,402	6,554,363	23,576,854
TREC file name pattern	AP89063*	AP89052*	AP8910*
number of queries	6	26	37
$recall_{1,1}$/$precision_{1,1}$	.2000/.1000	.1594/.0706	.1846/.0496
$recall_{4,4}$/$precision_{4,4}$	.4545/.0914	.6099/.0600	.4943/.0339
$recall_{10,10}$/$precision_{10,10}$	.5417/.0497	.7962/.0373	.7391/.0195
$recall_{19,19}$/$precision_{19,19}$	.6250/.0278	.8556/.0189	.8194/.0114

Table 2. Experimental result for three document sets

The experiments are named E294, E2108, and E7543 after the number of documents involved in the tests. E7543 uses a 25 times larger document set as E294 both in terms of the number of documents and the total size of the documents. In all experiments, the document database consisted of 1989's news articles from Associated Press (AP). Table 2 states a wildcard file name pattern that matches the documents of the experiment in TREC. TREC provides 100 queries for the full set of documents. The corresponding relevant documents for these queries are spread all over the 1,000,000 documents. If a smaller document set is used, relevant documents for some queries may no longer be included. Therefore, we used only those queries that actually have corresponding relevant documents in the document set. As expected, only 6 queries with these characteristics remain for E294, while 26 queries could be run for E2108, and 37 for E7543.

Some results of the experiments are given in the lower portion of table 2. These are the recall and precision values for certain combinations of number of query subjects and the number of document subjects (q and d parameters in the formulas of section 3.5). The values are an arithmetic average over all queries executed in the respective experiment. For example, regarding only the most probable subject of a query (q=1) and only documents ranked first for this subject (d=1), experiment E2108 had a $recall_{1,1}$ of .1594 and a $precision_{1,1}$ of .0706. Obviously, recall increases with the number of subjects and documents taken into account, i.e. increased q and d parameters. For the same experiment, $recall_{19,19}$ is .8556 and $precision_{19,19}$ is .0189.

While recall values are in reasonable orders of magnitude, precision seems to be very low in the first place. Low precision values indicate that the number of retrieved documents is very high in comparison to the number of relevant documents. It is expected that precision dramatically increases for the end user, however. The AIM subject prototype is only concerned with choosing a portion of the original search space. Then, a text retrieval engine searches this portion and will bring down the number of retrieved documents and thus increase precision. For reasons stated in section 3.3, the performance of a text retrieval engine is not included in the experiments. Regarding recall, AIM must achieve high values for the chosen search space, because if AIM misses relevant documents, a text retrieval engine will miss them, too.

As the results of table 2 suggests, recall increases with the increasing values of parameters q and d. Figure 1 plots recall with respect to the number of query and document subjects. Recall was computed for any combination of q and d with q=1, 4, 7, ..., 19 and d=1, 4, 7, ..., 19. The figure clearly shows that recall increases with the number of document subjects taken into account. An increased number of query subjects also improves recall but to a less extent. Also, the improvement is more obvious for smaller values of d (and q) than for larger values. Numerically, recall was never less than .1594 and never greater than .8556.

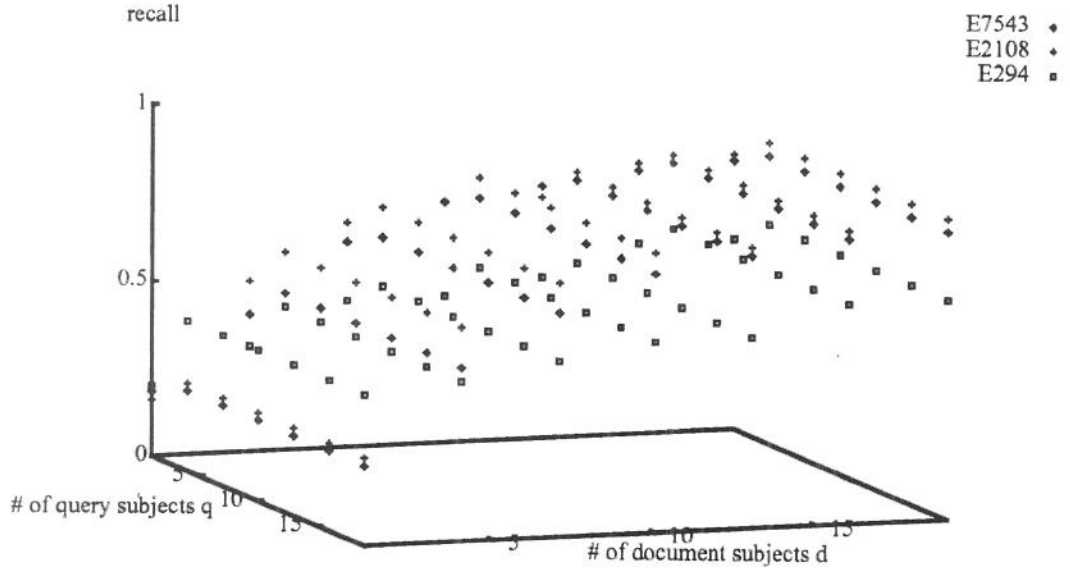

Figure 1. Avergage recall with respect to the number of query and document subjects

Another way of looking at recall and precision is with respect to the search space AIM selected. Figure 4 and figure 5 show the average recall and precision for all three experiments. The values on the x-axis are based on q and d parameters. For a given query, each combination of q and d selects a number of documents from the whole document set. This number dived by the total number of documents in the respective experiment is the portion of space searched. Again, the numbers shown are the arithmetic average over all queries in the experiment. For example in E7543, the number of retrieved documents for q = d = 10 was 425. Thus, the portion of space searched is 425 /7543 = .0563. Because for the same combination of q and d, $recall_{10,10}$ is .7391, the graph of figure 4 goes through point .0563/.7391. The precision values of figure 5 are computed in the same manner.

Whereas recall more or less increases with more search space added, precision decreases at the same time. In other words, although more and more search space is selected by AIM, recall never reaches 1 in our experiments. It only happens that more and more documents were chosen as candidates for search without including the missing, relevant documents. Clearly, our experiments do not cover measurements for search space selections greater than 11,5%. A valid question would be what is the minimum search space to get all relevant documents. At this point, we miss the answer.

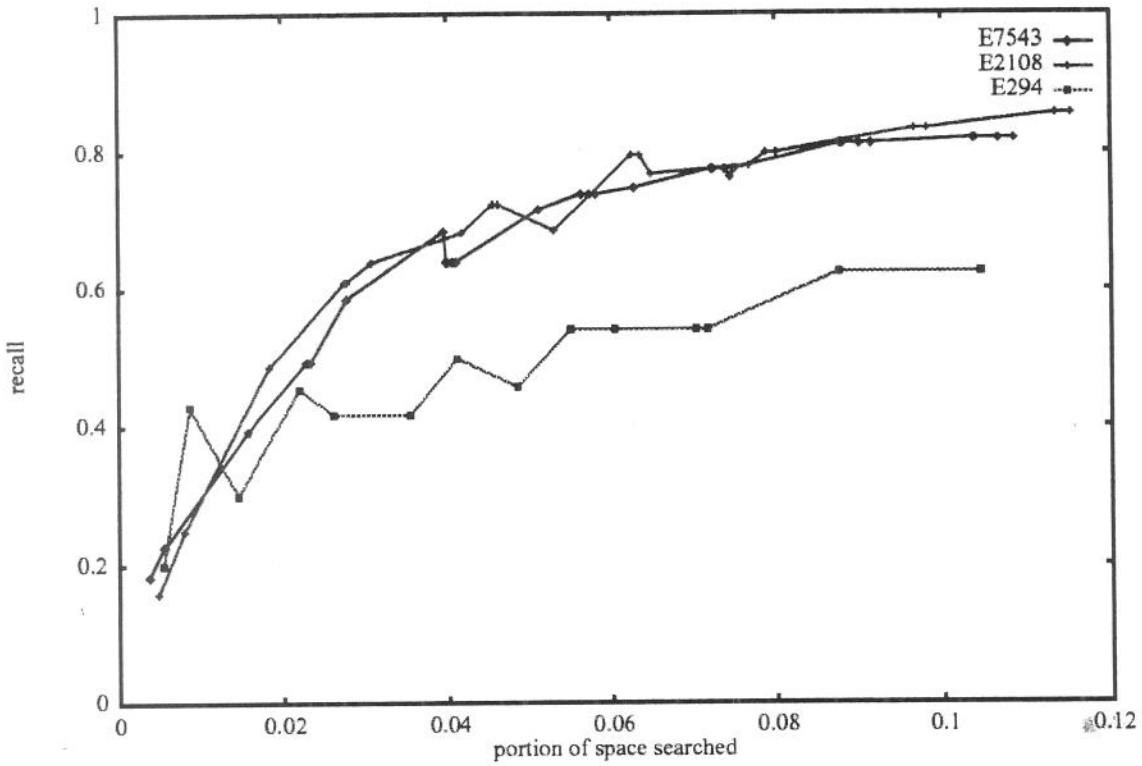

Figure 4. Average recall with respect to search space

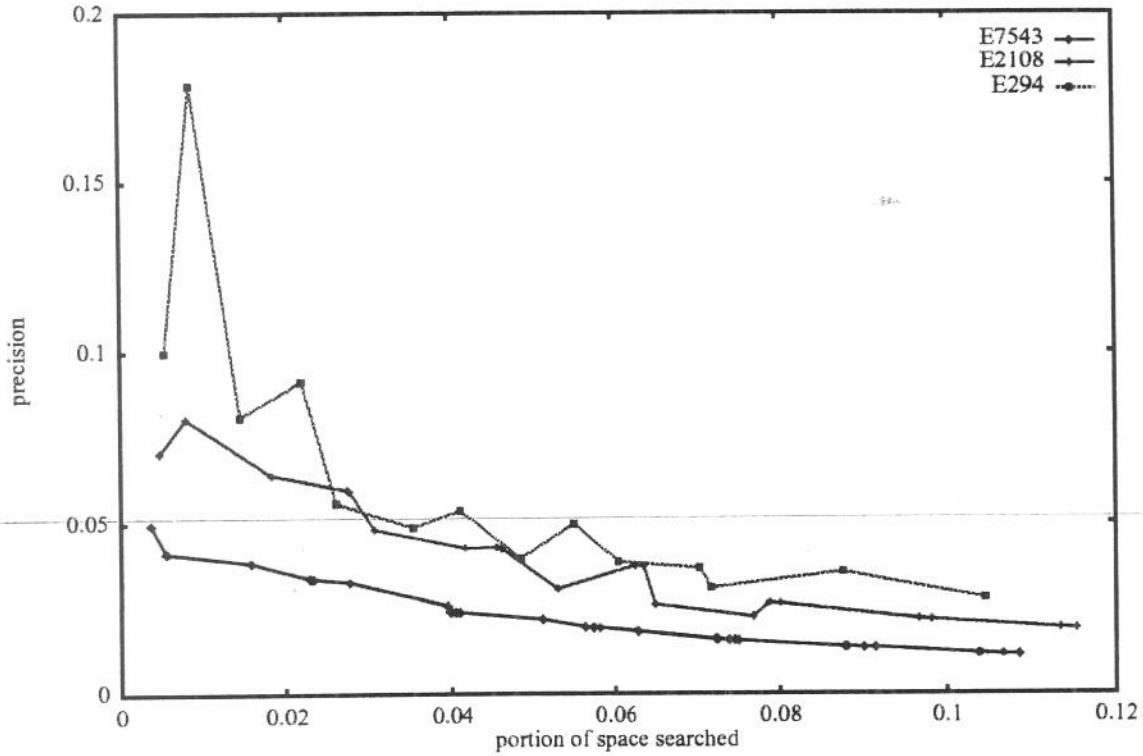

Figure 5. Average precision with respect to search space

However, answering the minimum question is not so important. We showed that the AIM's significant overhead on a text retrieval engine is well worth the effort and very suitable for loosely coupled information systems. It's well worth it, because by only looking at 2% of all documents, a recall of 50%, or by looking at 4% some 70% recall can be achieved (cf. figure 4). AIM is well suited for loosely coupled information systems like the WWW, because in such systems recall never reaches 1, anyway. There is no way of obtaining *all* relevant documents. Hence, recall values good enough is more what people are looking for.

All in all, this section showed that the AIM subject prototype can achieve a reasonable recall while narrowing down significantly the search costs. The approach of supporting search by subject tags is very promising.

4 Road maps: Another kind of information discovery tool

In this section, a topic is discussed that is related to information discovery in the WWW, but not directly related to AIM. The issue emerged from thinking of other navigational support than indexes, yellow pages, and search engines. We call the idea "usage based WWW road map".

The current situation of navigating through the WWW in pursuit of some information is as follows: Starting from a document related to the sought information, links pointing to other documents are explored. The big question is, however, which links are promising with respect to the retrieval task, and which are not. To be more precise, a first drawback in finding related documents is that HTML links are directed. Only documents that are pointed to can be reached, but there is absolutely no hint on documents that point in the opposite direction towards the current document. Obviously, documents that link to the current document are relevant for the search, too. A second problem is that a user sees only links to documents that are one hop away although she might be interested in documents that are pointed to from the linked documents and so on. There is no look-ahead greater than one hop in the WWW.

To overcome the above problems of directed links and limited look-ahead, a WWW road map could be employed. In such a map the WWW is laid out as a graph, where the nodes are documents and the edges are HTML links. With a WWW road map on hand, one can easily see what documents point to the current document and what documents are two, three or more hops away. Moreover, documents with a high fan-out of links could be identified and serve as good starting points for a search. The intended model of usage would be to have the road map open in a separate window when browsing through the WWW (Figure 6). The document currently displayed in the browser window is in the center of the road map window and highlighted (bold in figure 6). All nodes are labelled with the URL or the document title. The road map tells now what documents can be reached in how many hops and what documents point to the current document, if any. Either by clicking on the links in the browser, or by clicking on nodes in the road map window, new documents can be fetched.

Although not difficult in theory, building a WWW road map in practice is difficult. A Web robot (also called Web wanderer or Web spider) must first retrieve all documents and then write down all links between the documents. Assuming that a majority of the available documents can actually be found, the resulting graph would contain some 5,000,000 nodes (as of June 1995; [CMU95a]) and a number of edges in about the same order of magnitude. Even if the road map considers only HTTP servers instead of documents (and excluding ftp, gopher, WAIS servers etc.)

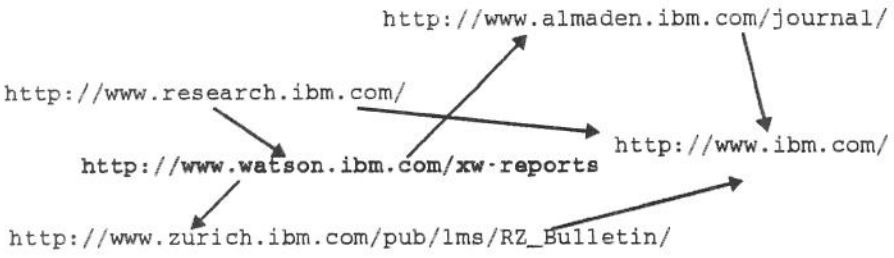

Figure 6. Road map

the graph would have between 23,550 (as of April 1995; [CMU95b]) and 40,644 nodes (as of July 1995; [Amer95]). The good news is now that such a comprehensive road map is not needed to provide reasonable navigational support and there are other sources than Web robots to build a smaller road map. Because this road map comprises only nodes and links actually used (with respect to a certain environment/site/institution/business), we call it "usage based WWW road map".

A question that comes natural is "what are the useful documents (servers) and the useful links for my business?". Fortunately, the answer is already given and ready-to-use in the server log files and in the user's browser history files. Server log files, as written by all popular HTTP servers, state what documents were retrieved at what time. (In fact, they state what method was applied to what file with what parameters from what host, and some more information, but we neglect this for now). Browser history files, as written by all popular Web browsers, retain the fetched URLs together with a time-stamp. Server logs are generally used to create statistics on the popularity of documents, and history files are used by the browser to mark followed links in a different color than untouched links.

Both server log files and browser history files are essentially a time-ordered list of documents. This list shows the sequence in which useful documents were accessed. Due to the information contained in the respective files, the list contains either documents of a particular server, or documents of a particular user. To construct a road map of useful Web documents for a work group/department/company, several individual lists of a work group/department/company have to be combined. Let us start with browser history files. figure 6 shows the merging process graphically for three history files with at most four documents each. The linear structure of the individual history files is transformed into a general graph by superimposing the genuine lists. An immediate improvement of the road map is to emphasize heavily used links. In figure 6 an increased line weight indicates links used more often.

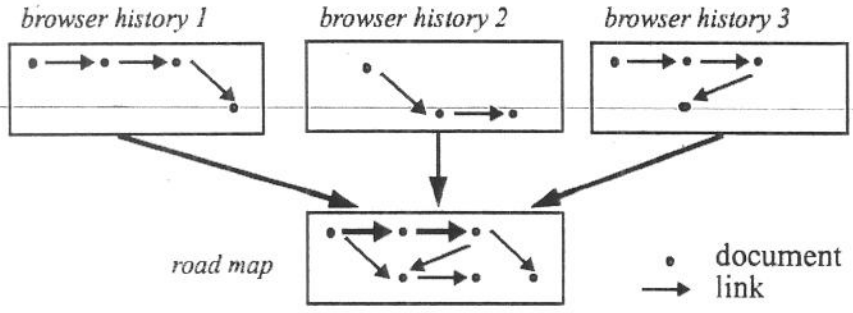

Figure 7. Merging three browser histories

Although the very same entries (documents) may be found in server log files and browser history files, there is a major structural difference between these two files. Log files contain one entry for each document retrieval (method invocation) and in general documents appear more than once in a log. History files mark only the last access to a document, so each document appears only once in the history. Consequently, histories are linear, time-ordered lists whereas server logs are general graphs (networks). Thus, if a history road map is further superimposed by lists created out of server log files, most probably only a small part of the road map would be affected but the number of links would be dramatically boosted in that region. In our opinion, much more road map information is available from history files, whereas the benefits of server log information is limited. We therefore suggest to build road maps by using history files exclusively.

With respect to the implementation of a usage based WWW road map, at least three problems arise. First, converting a multiply linked list in a handy graph in a non-trivial task. A layout has to be found that puts heavily interconnected nodes in one another's proximity. Additionally, the graph should be as planar as possible, i.e. exhibit few or none crossing edges. Total planarity is not achievable for general graphs. Second, it is the intention to update the road map area according to the document currently displayed in the browser window, and vice versa. This road map/browser interaction requires a browser API that is currently lacking for most browsers (Netscape is most advanced, however, the API could only be used to display URLs, but not to question the status of Netscape). Third, taking browser history files is a possible offense to a user's privacy, because they provide detailed insights into what user's like and how much time they spent "surfing the net".

5 A brief survey on approaches to query routing and subject description

For the related work discussed in this section, the focus lies on query routing and subject descriptions. Query routing is concerned with (re-)directing queries to a resource provider covering the subject of the query. Routing is of paramount importance for a scalable architecture. The way how documents or queries are assigned to subjects goes under subject descriptions. Besides the description and classification technique used, pre-defined or open classification systems may impact the usability and generality of the system. Table 3 summarizes the related work and the AIM approach with respect to these two issues.

Tool/Concept	*query routing*	*subject description*
Content Router	manually, by query refinement and selecting collections	attribute/values based on third site descriptions of the resources (WAIS catalog files)
Harvest	none; queries run against one replica of an information broker.	document types, determined by file names and URL names at a specific resource provider site.

Table 3. Comparison between query routing and subject description in popular resources discovery systems and AIM

Tool/Concept	*query routing*	*subject description*
WHOIS++	automatic, by using the "forward knowledge" of index servers in the directory mesh.	centroids, i.e. attributes used and words occurring in a given attribute in any record.
AIM	automatic, by calculating the query's topic (subject) and looking up servers dealing with that subject.	full word frequency vectors based on resources

Table 3. Comparison between query routing and subject description in popular resources discovery systems and AIM

Content Router ([DuSh94], [SDWO94]) directs user queries to appropriate, queryable resource servers, for example WAIS servers. Query routing is based on content labels describing a collection of resources or further collections by attributes and values. Typical attributes include "hostname", "owner", "author", and a catch-all attribute "text". They can vary for each collection, depending on the WAIS catalog files used for that collection. The content label of a collection describes their subject. Provided with information on the maintained attributes and possible values, a user picks up or drops off collections and refines her query by adding or deleting search terms and values. Execution of the query (called expansion) is done is parallel for all active collections and the results are combined afterwards. Content Router relies on the registration of resources to collections and requires that all members fit completely under the content label. Consequently, updates are limited to not changing the hierarchy of collections.

Harvest ([BDHM94]) is not a particular tool for resource discovery. Rather, Harvest is an architecture consisting of a number of interchangeable sub-systems for information extraction (Essence [HaSc94a]), storage of descriptive information (broker systems), caching of objects, replication of Brokers, and indexing and search (Glimpse [MaWu94], Nebula [BDBC94]). Documents belonging to some subject are managed by one broker (modulo replicas). Subjects are defined by first selecting manually resource provider sites to be used during the process of information gathering. Then, documents of a desired types are added to the index currently built. The document types are determined again via manually edited configuration files that contain regular expressions for file names (byname.cf), URLs (byurl.cf), and unwanted document types (stoplist.cf) ([HaSc94b]). There is no query routing in Harvest, however, a super broker (called Harvest Server Registry) can be used to select brokers covering most probable the subject of the query.

WHOIS++ ([DeSF94], [FaSW94], [WeFS94]) provides access to distributed information in the Internet. Originally an extension to WHOIS, a centralized information system about Internet users, WHOIS++ now allows for automatic query routing over independent indexes, as long as the information is structured in attribute/value pairs. Each WHOIS++ server describes its subject by centroids. Centroids are basically sets of attributes together with a list of values that appear under the attribute in any record of the server. For query routing through the directory mesh of servers, clients are provided with a list of servers to ask, in case no answers were found. This list is computed with the help of server centroids. The clients decides to redirect and to refine the query, or to return the answers available so far.

AIM uses a pre-defined subject classification which can be very large, for example the over 7000 news groups of the Usenet. Each document is automatically assigned to one or more subjects. First, word/frequency vectors are computed for each document. Then, the document vectors are compared to the subject vectors. The document is assigned to a subject when document vector and subject vector coincide to a certain degree. A similar mechanism is used to determine the subject of a query. Here, query vectors are compared to subject vectors. Because AIM knows which server covers which subject(s), the query can be routed automatically to the servers dealing with the queries subject. AIMs differs from all approaches discussed above with respect to its automatic document subject classification. With the exception of WHOIS++, AIM uniquely fosters automatic query routing.

REFERENCES

Amer95 America Online, Inc.: *WebCrawler Facts: Web Size.* WWW document `http://webcrawler.com/WebCrawler/Facts/Size.html`, 1995

BaSe95 Barrett, R., Selker, T.: *AIM: A New Approach for Meeting Information Needs.* Draft paper. IBM Almaden Research Center, 1995

BDBC94 Bowman, C., Dharap, C., Baruah, M., Camargo, B., Potti, S.: A File System for Information Management. *Proc. Conf. on Intelligent Information Management Systems (Washington (DC), June),* 1994, avail. as `ftp://ftp.cse.psu.edu/pub/bowman/doc/iims.ps.Z`

BDHM94 Bowman, C., Danzig, P., Hardy, D., Manber, U., Schwartz, M.: *Harvest: A Scalable, Customizable Discovery and Access System.* Technical Report CU-CS-732-94, Department of Computer Science, University of Colorado, Boulder, July 1994 (avail. as `ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Harvest.FullTR.ps.Z`)

CMU95a Carnegie Mellon University: *Lycos Frequently Asked Questions.* WWW document `http://www.lycos.com/lycos-faq.html`, 1995

CMU95b Carnegie Mellon University: *Measuring the Web with Lycos.* WWW document `http://lycos.cs.cmu.edu/lycos-websize.html`, 1995

DeSF94 Deutsch, P., Schoultz, R., Faltstrom, P.: *Architecture of the WHOIS++ service.* Internet Draft, August 1994 (avail. as `ftp://ietf.cnri.reston.va.us/internet-drafts/draft-ietf-wnils-whois-arch-01.txt`)

DuSh94 Duda, A., Sheldon, M.: Content Routing in Networks of WAIS Servers. *Proc. Conf. on Distributed Computing Systems (Poznan, Poland, June),* 1994 (avail. as `http://www-psrg.lcs.mit.edu/ftpdir/pub/papers/icdcs94.ps`)

FaSW94 Faltstrom, P., Schoultz, R., Weider, C.: *How to interact with a Whois++ mesh.* Internet Draft, July 1994 (avail. as `ftp://ietf.cnri.reston.va.us/internet-drafts/draft-ietf-wnils-whois-mesh-00.txt`)

HaSc94a Hardy, D., Schwartz: Customized Information Extraction as a Basis for Resource Discovery. *Technical Report CU-CS-707-94,* Boulder (CO): University of Colorado, 1994 (avail. as `ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Essence.Jour.ps.Z`)

HaSc94b Hardy, D., Schwartz, M.: *Harvest User's Manual. Technical Report CU-CS-743-94,* Boulder (CO): University of Colorado, 1994 (avail. as `ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Harvest.UserManual.ps.Z`)

IBM95 IBM Corporation: *SearchManager/2, SearchManager/2 LAN Server/2 and Client/2, SearchManager/2 Client/DOS for Windows Programming Interfaces.* Version 2. Draft (K/O500 level), Version drv0703, part # SH12-5987-00, 1995

IBM94 IBM Corporation: *SearchManager for AIX - Installation and Administration Guide.* Version 1. Prerelease Edition, part # SH12-6127-00, September 1994

MaWu94 Manber, U., Wu, S.: GLIMPSE: A Tool to Search Through Entire File Systems. *Proc. of the USENIX Winter Conference (San Francisco (CA), January),* 1994, p. 23-32 (avail. as `ftp://cs.arizona.edu/reports/1993/TR93-34.ps.Z`)

NIST95 National Institute of Standards and Technology: *Text REtrieval (TREC) Home Page.* WWW document `http://potomac.ncsl.nist.gov/TREC/`, 1995

SaMc83 Salton, G., McGill, M.: Introduction to modern information retrieval. New York: McGraw-Hill, 1983

SDWO94 Sheldon, M., Duda, A., Weiss, R., O'Toole, J., Jr., Gifford, D.: Content Routing for Distributed Information Servers. *Proc. Conf. on Extending Database Technology (Cambridge, England, March),* Springer LNCS 779, 1994 (avail as `http://www-psrg.lcs.mit.edu/ftpdir/pub/papers/edbt94.ps`)

WeFS94 Weider, C., Fullton, J., Spero, S.: Architecture of the Whois++ Index Service. Internet Draft, July 1994 (avail. as `ftp://nri.reston.va.us/internet-drafts/draft-ietf-wnils-whois-03.txt`)

Part II: The implementation viewpoint

6 Operational overview of AIM

The AIM prototype is a distributed information retrieval software for text documents. In a preparation phase, the documents are indexed, and later on queries are answered with document identifications (file names). Unlike conventional information retrieval, the indexing is based on the subject of a document, and not on the word occurrences in the document. (However, word occurrences are used internally to determine a document's subject). Most of the AIM testing was carried out using the TREC data set.

6.1 Overview of the core AIM system

AIM consists of three types of server components and three types of client components which are all realized in separate AIX processes. The processes can reside on the same or different computers. Figure 8 gives an illustration.

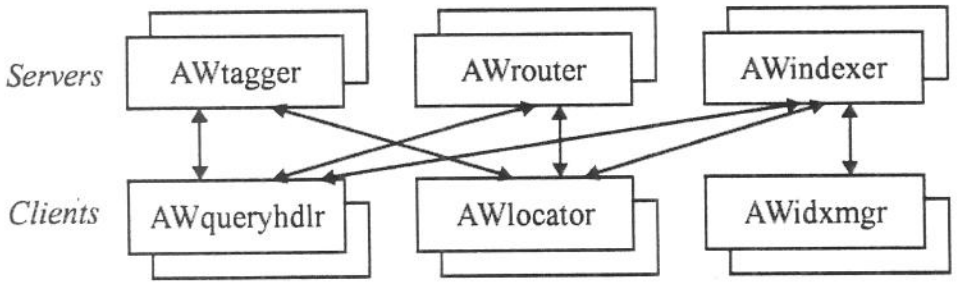

Figure 8. Client and server processes in AIM and their interprocess communication

The *tagger* server (executable name: `AWtagger`) does the subject indexing of documents and queries. A document or a query is assigned to a list of subjects. The number of query or document subjects is a parameter of the tagger.

The *router* server (`AWrouter`) computes for a subject the indexer which maintains documents of that subject. If a list of subjects is presented to the router, it will return a list of indexers with no duplicates. The router is primarily parameterized with a lookup table file.

An *indexer* server (`AWindexer`) maintains documents of one or more subjects. Indexers can add and remove documents to their index, and retrieve document names after being queried. The underlying technology is IBM's SearchManager product, a general purpose text retrieval engine.

A *locator* client (`AWlocator`) is used to load documents into the system. For this purpose, the locator takes the file name of the document, gets for the document a list of subjects from the tagger, asks the router for indexers handling these subjects, and finally forwards the document to the indexers which add the document to their index.

A *query handler* client (`AWqueryhdlr`) processes queries in two steps. At first, the query is sent to a tagger server and mapped there to a list of subjects. Second, the subject list is trans-

formed to indexers by the router, and the query is sent to the indexers. Eventually, the list of document names received from the indexers are displayed.

The *index manager* client (`AWidxmgr`) is the administrative front end to the indexers. It allows to create, drop, and maintain indexes through a simple command language.

6.2 Overview of the AIM/TREC supplements

The stand-alone executable `TRECtag` is pretty much the same as a tagger server and corresponding client combined, but for use with TREC queries and documents. `TRECtag` runs in query and document mode and maps a set of queries or documents, respectively, to a ranked list of subjects. This list is the input for computing recall and precision values.

The perl script `TRECrpPerQuery.pl` computes recall and precision values for certain queries. Input to this script is a ranked list of document subjects, the output are the recall and precision values per query, and per query subject and document subject combination.

With the perl script `TRECrp.pl` the per-query recall/precision values are averaged into one value per query subject and document subject combination.

To generate graphs out of the recall/precision values, a perl script called `TRECrp2ps.pl` is used. Input is list of recall and precision values per query, such as produced by `TRECrpPerQuery.pl`. Using public domain `gnuplot`, graphs are rendered on the screen or are written in postscript format.

`TRECoptRoutingTable.pl` is another perl script that generates a routing table, as used by the router `AWrouter`. The table does a uniform distribution over for a number of indexers with respect to a certain document set.

A comparison between TREC queries and TREC documents is performed by `TRECcompQueryAndDocument.pl`. This perl scripts counts how many words of the query are actually in the document.

7 AIM setup from scratch

Before AIM is able to answer queries, ab AIM system has to be set up and documents have to indexed. In this section, a step-by-step description of the involved design decisions and the start-up procedure is given. The main steps are:

1) starting SearchManager
2) building a routing table
3) starting the servers processes `AWtagger`, `AWrouter`, and `AWindexer`
4) create SearchManager indexes
5) loading documents (`AWlocator` client)
6) processing queries (`AWqueryhdlr` client)
7) miscellaneous tasks (`AWidxmgr` client)

If AIM should work with pre-existing indexes, only the steps 1), 3), and 6) are required. All steps are discussed in the subsequent sections 7.1 to 7.7.

7.1 Starting SearchManager

SearchManager is a separate text information retrieval product from IBM which is used in AIM. To use SearchManager, a couple of SearchManager server processes have to be started. SearchManager maintains indexes that belong to search services, and search services belong to one SearchManager instance. Search service and instance names have to be specified at start up, index names are specified later (section 7.4). See section 8.3.2 for instructions how to create an new instance and search service.

To see the defined SearchManager instance names, list the full contents of directory `/etc/SM6000/`:

```
% ls -a /etc/SM6000/
.               .sm6v1r1        .tii            excccxcom.cfg   exciq.dat
..              .ti             excadmtb.dat    excdocin.dat    exckeep.it
```

Here the instances named `SM6V1R1`, `TI`, and `TII` are defined. Despite the files are spelled in lower case letters, they have to be specified in capital letters later on. To see corresponding search service name for an instance, type

```
% /usr/bin/crtcfg <instance name>
```

for example

```
/usr/bin/crtcfg TII
```

The upcoming screen returns the search service name along with additional information. In the example, the search service name might be `TTWOS`. Before starting SearchManager check that it is not already running:

```
% /usr/bin/sm6000 status <search service name> <instance name>
```

for example

```
% /usr/bin/sm6000 status TTWOS TII
SearchManager status for service: TTWOS
SearchManager controller is not running
SearchManager communication service is not running
The requested operation completed successfully.
```

An output similar to the above indicates that you can go ahead with starting SearchManager. Do not start SearchManager under `root`, but use a login that belongs to group of SearchManager administrators (`smadmin`).

```
% /usr/bin/sm6000 start <search service name> <instance name>
```

for example

```
% /usr/bin/sm6000 start TTWOS TII
The requested operation completed successfully.
```

Now, SearchManager should be operational. If the output differs from the above, go to section 7.8

7.2 Building a routing table

The routing table is used by `AWrouter` to map subjects to indexers. It is an ASCII file with the following structure:

routing table file structure	routing table example (extract)
<number of entries>	1431
1 <subject number>	1 0
< host name> < port number>	THINKER.ALMADEN.IBM.COM 2390
1 <subject number>	1 1
< host name> < port number>	THINKER.ALMADEN.IBM.COM 2390
...	...

In the above example, the routing table contains 1431 subjects numbered from 0 to 1430. Subject 0 is mapped to an indexer process listening at port number `2390` at a computer with the IP host name `THINKER.ALMADEN.IBM.COM`. Subject number 1 is handled by the very same indexer process.

A sample routing table is in `$AIMAFSHOME/aimweb/src/router/router.in`. It maps 1431 subjects to the same port (`2330`) on the same computer (`THINKER.ALMADEN.IBM.COM`). If only one indexer process is employed in the AIM setup, no further action has to be taken. Otherwise, the sample routing table may be modified using a common text editor.

Another way of creating a routing table is the perl script `$AIMAFSHOME/aimweb/bin/TRECoptRoutingTable.pl`. This script takes a file containing document to subject mappings. Given a desired number of indexers, the routing table built from that input will assign to each indexer roughly the same number of documents. The format of the document mapping input file is like this:

document to subject mapping file structure	example (extract)
<doc name>:<subject number>:<rank>:<score>	AP890630-0001:1342:1:3150.7
<doc name>:<subject number>:<rank>:<score>	AP890630-0001:245:2:1060.5
...	AP890630-0018:264:1:1089.1
	...

The individual lines constitute an association between a document (`doc name`), a subject it is mapped to (`subject number`), how many subjects are considered more appropriate for the document (`rank`), and a numerical score, indicating the distance between two ranks of the same document (`score`). The command line parameters of `TRECoptRoutingTable.pl` are:

```
% TRECoptRoutingTable.pl [-vv]
      -n <# of indexers> -p <port # to start>
      -h <hostname where the indexers listen>
      -e <# of subjects to in routing table>
```

If specified, `-vv` gives a more verbose output on stderr. The created routing table will assign subjects to `-n` indexers whereas the first indexers listens at a port number `-p`, the second on `-p` + 1, the third on `-p` + 2, and so on. Subjects not covered in the input file are assigned to port number `-p` + `-n` + 1, the nullport. All routing table entries contain the value of `-h` as the host name for the indexer. Finally, the value of parameter `-e` sets the number of subjects written to the routing table. For example

```
% TRECoptRoutingTable.pl -vv -n 3 -p 2390 -h thinker.almaden.ibm.com
    -e 1431 AP89063.bydocs
```

will create a routing table for 3 indexers listening at ports 2390, 2391, and 2392. Port number 2393 is the nullport and will be used for all subjects not found in AP89063.bydocs. The host name used for all indexer entries is thinker.almaden.ibm.com, and 1431 entries will be written.

The obvious limitation of TRECoptRoutingTable.pl that only one host name can be specified (-h option) seems to be justified, because there is currently only one machine running SearchManager (thinker.almaden.ibm.com). Editing of the routing table will allow for other host names, where necessary.

7.3 Starting the servers processes AWtagger, AWrouter, and AWindexer

After having started SearchManager, AIM's server processes can be started. In the simplest case of only one AWtagger, AWrouter, and AWindexer process, almost no parameters have to be specified, because reasonable defaults are built-in. All processes are started from the command line at the host where the server process is supposed to run. The server processes run under a normal user id, i.e. not root. The user id of the AWindexer must be capable to use SearchManager which means that this user must have read access to SearchManager commands, indexes, work directories, and so on. No special SearchManager authorization is required in the installed version.

Common to all server processes is the MyPortNum parameter which is the port number the clients connect to. Usually, the defaults work fine, but if on the same host more than one process of the same kind should be started, unique port numbers have to be assigned manually.

To start the tagger, the number of document/query subjects has to be specified. Although these are different parameters in principle, AWtagger takes only one parameter and interprets it as the number of document subjects when a document is tagged, and as the number of query subjects when a query is tagged. Section 3.4 explains these parameters.:

```
% $AIMAFSHOME/aimweb/bin/AIX/AWtagger
    [-n <# of subjects to be assigned for one doc/query>]
    [<MyPortNum>]
```

Defaults/example:

```
% AWtagger -n 10 2310
```

The tagger's operation depends on two files not specified as a parameter. One file contains a dictionary, and the other file contains the names of the subjects (usenet newsgroups). Because these files hardly ever change, their full path names are compiled into the tagger executable. When the tagger is started, the contents of both files is read into memory and preprocessed which takes a couple of minutes. An informational message will tell when initialization is completed.

A router process' main parameter is the file containing the routing table. Section 7.2 explained how to create such a file.:

```
% $AIMAPSHOME/aimweb/bin/AIX/AWrouter
    [-i <RouterInFile>] [<MyPortNum>]
```

Defaults/example:

```
AWrouter -i router.in 2320
```

When an indexer is started, it immediately tries to establish a session with a SearchManager service and to open an index. The names of the SearchManager service and the index are given as parameters. To determine the already defined services name for an SearchManager instance, use `crtinst <instance name>` (section 7.1). To find out about the index names maintained by a search service, the SearchManager intended way would be to write a little program for its cumbersome API. Another way to get the desired information from the AIX command line is to first get the current path of the index directory. To do so, look at the variable `EXCDATASRV` in file

```
% /cat /etc/SM6000/.<instance name>
```

For example

```
% cat /etc/SM6000/.tii
EXCDATASRV=/smi2
EXCWORKSRV=/smi2w
EXCNLPSSRV=/usr/lpp/SM6000/dicts
```

Here, the index directory is `/smi2`. Now look at the strings in binary file `/<index directory>/excmastr.dat`. For each maintained index, three entries exists: The name of the index, the location of the index directory, and the location of the work directory. For example:

```
% strings /smi2/excmastr.dat
IA2390
/smi2/IA2390/
/smi2/IA2390/
IALLDOCS
/smi2/IALL/
/smi2w/IALL/
...
```

The index names in the above case are `IA2390` and `IALLDOCS`. Remember to write them in capital letters, despite they may be displayed in lower case letters. Besides search service name and index name, an appropriate port number has to be found for each indexer. The routing table (section 7.2) used by `AWrouter` lists the different host name/port number combinations that must run indexers. The following AIX command gives you an idea:

```
% egrep -v '^1 ' <router table file>| sort -u
```

For example

```
% egrep -v '^1 ' $AIMAPSHOME/aimweb/experim/second/router.TONES.in | \
    sort -u
1431
```

```
THINKER.ALMADEN.IBM.COM 2390
THINKER.ALMADEN.IBM.COM 2391
THINKER.ALMADEN.IBM.COM 2392
...
```

With the exception of the first number (1431) which is the number of entries in the routing table, the other lines show the host name/port number combinations. It is important to run the same index names on the same hosts/ports at all times, because this is the mapping between router port numbers to the actual data. In other words, the mapping form subjects to host names/port numbers and on to SearchManager indexes must never change in one AIM setup. Otherwise, queries would be routed to SearchManager indexes maintaining completely different subjects than the query covers. Now all parameters are known and the indexer server can be started finally, using the following command:.

```
% $AIMAFSHOME/aimweb/bin/AIX/AWindexer
[-s <SearchMgrService>] [-i <SearchMgrIndex>]
[<MyPortNum>]
```

For example

```
% AWindexer -s "TTWOS" -i "IA2390" 2390
```

Here, an indexer is started on the host where the command is issued. It uses search service TTWOS and provides access to an index named IA2390. By coincidence, the indexers listens on port number 2390. If the indexer is started for the first time, the following output will occur:

```
% AWindexer -s "TTWOS" -i "IA2390" 2390
try to establish indexer at port 2390 using SearchMgrService "TTWOS" and
SearchMgrIndex "IA2390"
SM/2 Index is not fully operational -- only administrative task are
allowed.
INDEXER Server ready for requests at thinker.almaden.ibm.com 2390
```

As the second line indicates, the index has not been created yet, and hence the only allowed task is to create indexes (see the following section 7.4). Once both search service and index are accessible at start up, the above hint will disappear.

7.4 Create SearchManager indexes

With at least SearchManager and one AWindexer running, indexes can be administrated. The other two kinds of AIM servers AWtagger and AWrouter are not necessary for this task. AWidxmgr acts as the client to the AWindexer server, that is administrative requests are sent to the server, executed there, and return codes and results are sent back to the client. Although AWidxmgr can process a couple of tasks (section 7.7 gives a detailed listing), only the creation of indexes is discussed here. To create an index, first the characteristics of the index are described using a command language. Using your favorite text editor, an ASCII file containing the statements of this command language is created. Afterwards, this file is processed by AWidxmgr.

A minimal command file with all statements necessary to create an index is this:

```
/*
 * example to create an index with AWidxmgr
 */
```

```
start administrative session "TONES";

create index name "IONE"
type linguistic
LSCE "EHSLSCFS" LSSE "EXCLSSFS"
XLOC "/smi2/IONE" WLOC "/smi1w/IONE";

end session;
```

There are three statements: start session, create index, and end session which are terminated with a semicolon (;). To start a session the name of the search service is basically specified. It is sufficient, if SearchManager is installed on the same computer where the `AWindexer` server is running, because the port number could then be taken from `/etc/services`. Otherwise, communication type and port information has to be provided at this point (section 7.7). The optional keyword `administrative` is needed in this case, because creating an index is a task that requires administrative authorization by SearchManager.

The second command determines the characteristics of the index. Check SearchManager documentation ([SaMc83]) for more information on the following items. Only uppercase characters A-Z and numbers 0-9 are allowed for the index name which is `IONE` in the above case. SearchManager index types are either `linguistic`, `precise`, or `dual`. Next comes the name of the library services server executables (`LSSE "EXCLSSFS"`) and library services server executables (`LSCE "EHSLSCFS"`). These libraries must be in directory `/usr/lpp/SM6000/lib/`. The last two parameters of the create index statement are the full path names of index directory (`XLOC "/smi2/IONE"`) and working directory (`WLOC "/smi1w/IONE"`). It is recommend to set them up as separate file systems, although not mandatory. Check the file protections and ownerships for theses two directories (section 7.8).

Last, the sequence of commands ends with an end session statement. To execute the commands, the file must be processed by `Awidxmgr` which parameters are:

```
% $AIMAFSHOME/aimweb/bin/AIX/AWidxmgr [-P <MyPortNum>]
    [-i <IndexerHostNam>] [-I <IndexerPortNum>]
    <taskfile>
```

Defaults/example:

```
% AWidxmgr -P 2350 -i thinker.almaden.ibm.com -I 2330 createindex.inp
```

The parameter `MyPortNum` is again the port number (`2350`) on which the `AWidxmgr` client resides. Because it acts as a client, it also has to know the host name (`thinker.almaden.ibm.com`) and the port number (`2330`) of the indexer server. The file `createindex.inp` holds the commands. `AWidxmgr` waits for the execution of all statements and reports their success or lack thereof.

7.5 Loading documents (`AWlocator` client)

From the user's perspective, loading documents into the AIM system is a process that comprises two different steps.

1) tagging, routing, and scheduling the document for indexing
2) updating the index

To perform the first step, a client called AWlocator is available. The second step requires a small command language script and the use of AWidxmgr, as shown later on. The syntax of AWlocator is as follows:

```
% $AIMAPSHOME/aimweb/bin/AIX/AWlocator [-L <MyPortNum>]
      [-t <TaggerHostNam>] [-T <TaggerPortNum>]
      [-r <RouterHostNam>] [-R <RouterPortNum>]
      <fully qualified filename of document>
```

Defaults/example:

```
% AWlocator -L 2300 -t thinker.almaden.ibm.com -T 2310
     -r thinker.almaden.ibm.com -R 2320
     /TREC/data/ap/ap890630D/AP890630-0147
```

AWlocator assumes that at least one of each kind of AIM servers (AWtagger, AWrouter, and AWindexer) are up and running. Similar to the equally spelled server parameter, MyPort-Num is the port number where the AWlocator process is supposed to run. The default must be changed only in the case of more than one locator process. AWlocator first sends the document to the AWtagger server process specified with the -t and -T option. Second, the result of the tagging process is forwarded to the router server identified with the -r and -R option. Note that the default port numbers where tagger and router are listening (section 7.3) correspond to these defaults. There is no need to hand over to AWlocator the address of an indexer, because host name and port number information are computed as part of the routing process. The document is sent to the indexer automatically. AWlocator reads documents in any ASCII-encoded textual format. However, tagger and the SearchManager merely work on a sub set of the document, i.e. the words, sentences, and paragraphs they recognize.

After a successful run of AWlocator, the document sits now waiting to be indexed in SearchManager's document queue. This queue is either processed on a regular on a on demand basis. To make SearchManager process the queue immediately, a script similar to the following can be executed with AWidxmgr.

```
/*
 * update index
 */

start session "TONES";
open index "IA2390"; update index; close index;
open index "IA2391"; update index; close index;
end session;
```

All statements and their parameters are explained in section 7.7. Note that more than one index (IA2390, IA2391) can be updated in the same session (TONES), as long as all indexes belong to the same search service (TONES). To execute the script, put it into a file (e.g. update.script) and run

```
% AWidxmgr [-P <MyPortNum>] [-i <IndexerHostNam>]
     [-I <IndexerPortNum>] update.script
```

With IndexerHostNam and IndexerPortNum only one AWindexer process of the TONES search service family has to be characterized, although several AWindexer processes may be executing concurrently. Every single AWindexer server is capable to update all indexes within

its search service. More details on the command line parameters of `AWidxmgr` are given in section 7.4.

7.6 Processing queries (`AWqueryhdlr` client)

Executing queries against AIM is the whole purpose of the AIM system. Unlike other text retrieval software, AIM takes two queries to retrieve document names. The first query is used to select indexers with a high probability of carrying documents that match this query. The second query is run against all selected indexers. The rationale behind the two step query processing is that the user can refine her query, depending for example on the selectivity and the chosen indexers. Of course, the first and the second query can be identical, although they use a different syntax (see below). To run both queries, the `AWqueryhdlr` client is used. As a prerequisite, at least one `AWtagger`, `AWrouter`, and `AWindexer` server must be up and running. The syntax is like this:

```
% $AIMAPSHOME/aimweb/bin/AIX/AWqueryhdlr [-Q <MyPortNum>]
    [-t <TaggerHostNam>] [-T <TaggerPortNum>]
    [-r <RouterHostNam>] [-R <RouterPortNum>]
    [-n <max # of results>]
    <file w/ 1st query> <file w/ 2nd query>
```

Defaults/example:

```
% AWqueryhdlr -Q 2340 -t thinker.almaden.ibm.com -T 2310
    -r thinker.almaden.ibm.com -R 2320 -n 10 <file> <file>
```

As `AWqueryhdlr` acts as a customized `AWlocator` (or `AWlocator` is a specialized `AWqueryhdlr`), most of the `AWqueryhdlr`'s parameters are similar to those of `AWlocator`. In particular, the `-t` and `-T` option specify an `AWtagger` server process, and the `-r` and `-R` options an `AWrouter`. The host and port number of an appropriate indexer process are computed as part of the routing process and need not be given here. Also, in case port number `2340` is already used by other instances of `AWqueryhdlr` processes, the `MyPortNum` parameter lets you again change the default port number on which the `AWqueryhdlr` client is supposed to run. Unique to `AWqueryhdlr` is the `-n` option that specifies the maximum length of the result list returned by one `AWindexer` server. As indicated above, `AWqueryhdlr` needs the two file names with queries as mandatory parameters, eventually. After the first query is tagged and before the resulting subject list is forwarded to the router, processing is suspended. `AWqueryhdlr` asks whether the user wants to proceed with the second step query, specify a new first step query, or edit the second step query.

The first query can be written virtually syntax-free. AIM is only interested in the words and the number of their occurrence and skips punctuation and so on. This is an example of a first step query:

```
investment banking, investment banker, arbitrage, NOT currency
```

The second query is processed by SearchManager and thus must be a boolean combination of search terms. A vertical bar (|) or `OR` indicates OR, the ampersand (`&`) or `AND` is AND, `NOT` is the negation. Because NOT precedes AND, and AND precedes OR, one level of parentheses is provided, too. To include those keywords in the query, or to use special characters, words and phrases can be enclosed in double quotes (`"`). Anything in between C style comments (`/*` and `*/`)

will be skipped during interpretation. All queries start with the keyword QUERY followed by an integer. An equivalent second step query of the above first step query would be:

```
QUERY 4711
investment banking AND investment banker AND NOT arbitrage
```

AIM's output on the query will be a list of file names. However, the file names are different from the ones handed to AWlocator during indexing. The reason for this lies in a limitation of SearchManager which always returns the file name that was used during indexing. The distributed architecture of AIM requires to create temporary files at AWindexer's computers, and the full path names of this temporary files (as taken by the SearchManager) are obviously different from the original files. In order to maintain the connection between original and temporary files, AIM retains the file name part of the fully qualified path name, and creates temporary files with that name. Hence, the path names returned by SearchManager have only in the file name in common with the document file.

Each indexer returns a separate list of file names which is ranked with respect to the number of word occurrences. The AIM prototype currently makes no effort in merging lists from different indexers into one list. Instead, the individual lists are displayed.

7.7 Miscellaneous tasks (reorganizing/dropping indexes)

All SearchManager indexes within the AIM system are solely managed by the AWindexer server. Consequently, besides running SearchManager (section 7.1), AWindexer has to be started (section 7.3). The other servers AWtagger and AWrouter don't have to run during index administration, although this would not do any damage. The corresponding client to the AWindexer server is AWidxmgr. This client essentially takes a file containing commands, and cares for their execution at the server site. In repetition of section 7.4, this is the command syntax of AWidxmgr:

```
% $AIMAPSHOME/aimweb/bin/AIX/AWidxmgr [-P <MyPortNum>]
    [-i <IndexerHostNam>] [-I <IndexerPortNum>]
    <taskfile>
```

MyPortNum is the port number which AWidxmgr uses to communicate with an AWindexer server. This AWindexer resides at the host name IndexerHostNam and the port number IndexerPortNum. taskfile is an ASCII file containing statements in a command language which is defined in the remainder of the section. In addition to this documentation, the rationale behind SearchManager parameters is explained in [SaMc83].

Regarding the syntax description, keywords are in uppercase, parameters and non-terminals are in lowercase. The parser, however, will recognize both all lowercase and all uppercase keywords. '[' and ']' embrace optional elements, '{' enclose '}' related syntactical structures, the vertical bar (|) separates alternatives. Double quotes (") are part of the actual syntax. All statements are terminated with a semicolon (;). C style comments (enclosed in /* and */) are allowed anywhere in the script.

A complete command sequence consists of a start session statement, a non-empty list of further statements, and an end session statement. A grammar-like description would be like this (with S as the start symbol):

```
S:
        startsession_stmt
```

```
        statementlist
        END SESSION SEMICOLON

statementlist:
        statementlist statement SEMICOLON |
        statement SEMICOLON

statement:
        createindex_stmt |
        openindex_stmt |
        updateindex_stmt |
        deleteindex_stmt |
        closeindex_stmt |
        scheduledocument_stmt |
        listindex_stmt
```

In the sequel, the syntax of the particular statements is described and exemplified.:

startsession_stmt syntax	startsession_stmt example

`START [ ADMINISTRATIVE ] SESSION`
`{ "service-name" |`
`{ COMMUNICATION TYPE { TCPIP | PIPE }`
`HOST "machine-name"`
`PORT "port-number" }}`
`[ USER "user-name"`
`PASSWORD "password" ]`
	`start session`
	`communication type tcpip`
	`host "thinker"`
	`port "1625"`
	`user "kirsche"`
	`password "guesswhat"`

With the start session statement, a SearchManager environment for a sequence of commands is set up. Basically, a session could be of type `ADMINISTRATIVE` or not. If it is not authentication uses the information provided after the `USER` and `PASSWORD` keywords. Because of a bug in the current search manager version (see file `ehwapsec.h` in SearchManager's API distribution), authorization does not work properly. The `ADMINISTRATIVE` keyword makes `USER` and `PASSWORD` information obsolete by replacing them with a dummy access code that is permitted to execute all administrative operations. To issue regular operations like scheduling documents, a session must not be declared `ADMINISTRATIVE`. Even though `USER` and `PASSWORD` should become evaluated in this case, this does not happen in the current version which lets one skip this part of the statement.

The other parts of the start session statement deal with the identification of a search service. If SearchManager runs on the same computer as `AWindexer` and the search services are listed in the `/etc/services` file, a search service could be identified by its name or aliases. Otherwise, information on the `COMMUNICATION TYPE`, the `HOST` of SearchManager service, and the `PORT`

where this service listens has to be specified. The PIPE option provides a local optimization over TCPIP for a local SearchManager installation, but it's not implemented yet.

createindex_stmt syntax	createindex_stmt example
CREATE INDEX NAME "index-name" TYPE { LINGUISTIC | PRECISE | DUAL } { LIBRARY SERVICE CLIENT EXECUTABLE | LSCE } "lib-name" { LIBRARY SERVICE SERVER EXECUTABLE | LSSE } "lib-name" [{ LIBRARY IDENTIFIER | LIBID } "lib-id "] XLOC "index-path" WLOC "work-path"	create index name "I1A2390" type linguistic LSCE "EHSLSCFS" library service server executable "EXCLSSFS" XLOC "/smi1/i1" WLOC "/smi1w/i1"

Section 7.4 already elaborated on the create index statement, so a brief description is given here just for completeness. A first parameter is the index's name, which has to follow the rules of SearchManager names (characters A-Z and 0-9 only). The type of the index is stated next. To read and index documents, SearchManager can be customized with so-called library service executables on server and client side. The name of these libraries residing in /usr/lpp/SM6000/lib/ is given after the abbreviations LSCE and LSSE or their longer equivalents. The library identifier is optional, depending on how the library should be initialized by the SearchManager. Finally, the XLOC keyword leads to the index directory, and WLOC to the working directory of the newly created index.

openindex_stmt syntax	openindex_stmt example
OPEN INDEX "index-name"	open index name "I1A2390"

After an index has been created, it may be opened using the open index statement. The only parameter is the name of the index to be opened. SearchManager requires an open index statement before all update index, delete index, and close index statement, though the grammar does not. Because the open index statement fully specifies the index name, the update, close, and delete index statements do not need this name as a parameter. The syntax of these three statements is very simple.

updateindex_stmt syntax	updateindex_stmt example
UPDATE INDEX	update index

closeindex_stmt syntax	closeindex_stmt example
CLOSE INDEX	close index

deleteindex_stmt syntax	deleteindex_stmt example
DELETE INDEX	delete index

Scheduling of documents is usually done using AWlocator, because the AIM system must first decide which index is going to be used. For debugging purposes and when the appropriate

indexer is already known, documents can be scheduled for indexing directly. Note that the document file has be fully specified, and must not be relative to the current working directory.

scheduledocument_stmt syntax	scheduledocument_stmt example
`SCHEDULE DOCUMENT [ ADD | REMOVE ] "full pathname"`	`schedule document add "/TREC/data/ap/ap890630D/AP890630-0007"`

The following list index statement is honored by the parser, but never executed. It's an hook for future implementation of information services in the command language.

listindex_stmt syntax	listindex_stmt example
`LIST INDEX`	`list index`

7.8 Setup problems

There are a number of problems one might encounter during setup and operation of AIM. In the sequel, some known problems and their solutions are described.

7.8.1 AIM setup without SearchManager

Server and client processes bind to port numbers. If a process tries to bind to a port number already in use, an output similar to the following is written to stderr:

```
% AWrouter -i router.ONEINDEX.in
try to establish router at port 2320 reading "router.ONEINDEX.in"
PrepareToRecv -- could not bind to port!
aw_msghndlr::constructor -- Unable to configure receiving!
main -- can't initialize communications!
```

The defaults port numbers and how to alter them are described in section 7.3 for the servers, and in sections 7.4, 7.5, and 7.6 for the clients. One could either start the process at a different port, or find and kill the process currently using the default port. In the example above, port 2320 seems to be used already. To verify this, check:

```
% netstat -a | grep 2320
tcp        0      0  *.2320                 *.*                    LISTEN
```

An output similar to the above indicates that the port (socket) is currently used. There is no simple way to find out the process currently using a specific port, as far as we know. (The program `/afs/tor/common/progs/lsof/lsof | grep <port number>` will do it, though, but it has to run under root). Hence, look through the output of `ps -ef` and guess the process to be killed.

7.8.2 AIM operation without SearchManager

In the current AIM prototype, there are no error messages. Instead, numerical error codes are used. For historical reasons, it is also quite cumbersome to compute at least a symbolic error message from a given error number, because there is no central error code data base. An example will demonstrate how one would try to determine the symbolic error code, anyway. Assume that `AWidxmgr` was started without an `AWindexer` running: The following would be written to stderr.

```
% AWidxmgr inp.updateindex
AWidxmgr -P 2350 -i thinker.almaden.ibm.com -I 2330 inp.updateindex
AdminIdx Request!  addr=101 thinker.almaden.ibm.com 2330
msgid=0
aw_msgrequest::sendrequest -- Failed to send request!
IndmgrClientHandleObj::go(0) -- sendrequest error: 525
Indmgr client closed
```

The relevant error code is 525. Because all error numbers are assigned with respect to a component-specific basis, the first task is to determine that basis. Familiarity with code base tells that the `msgio` module assigns error codes with a basis of $0x0200 = (512)_{10}$:

```
% cat $AIMAFSHOME/aimweb/src/core/msgio.h
...
#define MSGIO_BASE 0x0200
#define MSGIO_WRONGLENGTH (MSGIO_BASE + 1)
...
#define MSGIO_COMMINITFAIL (MSGIO_BASE + 12)
#define MSGIO_CONNECTFAIL (MSGIO_BASE + 13)
#define MSGIO_NOLISTEN (MSGIO_BASE + 14)
...
```

Because 525 - 512 = 13, the symbolic error constant would be `MSGIO_CONNECTFAIL` in this case. Another weakness of the prototype is that there is no detailed description of the symbolic error constants.

7.8.3 SearchManager setup

Again, all setup has to be done under a normal userid, not under `root`. This userid must actively belong to the `smadmin` group. Check with the AIX `groups` command.

Sometimes, SearchManager does not start up due to leftovers from a previous run. First, check that there are no SearchManager processes running:

```
% ps -ef | grep exc
% kill ...
```

Second, check that there are no shared memory segments belonging to SearchManager:

```
% ipcs -m
% ipcrm ...
```

On a workstation running the X windows, usually only one shared memory segment is active for one user. Then try again to start SearchManager.

For newly defined instances, index paths and so on, problems may be caused by inappropriate file protections. The file ownerships for index directory and working directory (these paths are defined with `crtinst`) should be like this. Notice the set gid bit.

```
drwxrwsr-x   2 root     smadmin       512 Sep  6 10:25 /smi1
drwxrwsr-x   2 root     smadmin       512 Sep  6 10:26 /smi1w
```

Last, newly defined search services must be assigned a port number greater than 1024. Port numbers lesser than 1024 are for root usage, only. Because the SearchManager processes are not running under root, they must use ports greater than 1024.

7.8.4 SearchManager operation

In addition to returning error codes to the client, the `AWindexer` server prints all error codes returned from SearchManager to `stderr`. These numerical error codes are listed in file

```
% cat /usr/include/ehwapic.h
...
#define  RC_UNKNOWN_SERVER_NAME              16
#define  RC_INCORRECT_AUTHENTICATION         17
#define  RC_DATASTREAM_SYNTAX_ERROR          18
#define  RC_QUERY_TOO_COMPLEX                22
...
```

For example, error code 18 corresponds to `RC_DATASTREAM_SYNTAX_ERROR`. Given the name of the constant associated with the error, an error description can be looked up in [SaMc83], chapter 4. Some of more common errors are:

1. RC_DATASTREAM_SYNTAX_ERROR

This error typically occurs during processing of the second step query. The second step query, as handed over to `AWqueryhdlr`, is translated into a SearchManager data stream on a 1:1 basis. If an operator like | (OR) or & (AND) is missing in the query, it will miss in the data stream, too, and raise the above error. Hence, check your second step query for correct syntax.

2. RC_UNEXPECTED_ERROR

In most of the cases when this error occurs, simply restart the AWindexer that issued this error. This error will probably go away when the SearchManager version gets more stable.

3. RC_IO_PROBLEM

This error refers mostly to file unavailability or file protection violations. Check the names and the file permissions of index directory and working directory of the search service. Both paths were specified with the `crtinst` tool (sections 7.1 and 8.3.2). Also, check the names and the file permissions of the paths specified in a create index statement (section 7.7).

4. RC_UNKNOWN_SERVER_NAME

As part of the start session statement of the command language (section 7.7), either the server name or a host name/port number combination have to specified. Due to unknown reasons SearchManager is not able to honor the server name at all times. Hence, try to specify host name and port number. These values are found in `/etc/services` (section 8.3.2).

5. RC_UNKNOWN_SERVER_INFORMATION

Most probably, this error is a result of a start session statement that has a fully qualified host name in it, like `thinker.almaden.ibm.com`. SearchManager currently allows only the letters A-Z and the digits 0-9 as identifiers. Because the dot (.) causes the above problem, try plain `thinker`.

6. Documents get scheduled but index is not updated

The above effect is not accompanied by an erroneous return code, but noticed on the application level when queries that should retrieve an earlier indexed document fail to do so. Both the document scheduling operation and a subsequent update operation return `RC_DONE` which indicates a successful call. Possible causes for not updating indexes lie in file unavailability due do wrong path names or file protection violations. Another reason could be that an `ADMINSTRATIVE`

session was in effect for the SCHEDULE statement (section 7.7). The SearchManager development team decided to treat document unavailability not as an error.

8 Implementation and installation/compilation of AIM

In this section, the code base of the AIM subject prototype, how to compile it, and the installation of SearchManager is briefly discussed. The following table 4 gives an overview of the development environment of the AIM subject prototype. Also, the values of two environment variables are shown. All makefiles and perl scripts use this two environment variables.

Object	Version/Value
OS	AIX 3.5.0
Compiler	CSet ++ V 2.1.3
SearchManager	drv0703
perl	perl4
AIMAFSHOME	/u/kirsche/exp/u/kirsche/exp
AIMLOCALHOME	/home.native/kirsche on thinker.almaden.com

Table 4. Development environment

8.1 Code base

The code base for the AIM subject prototype is basically separated into two parts. One part, called classlib, contains general functions for information retrieval, data structures, and SearchManager wrappers. Another part, called aimweb, comprises the specific AIM code. Both have the same general structure as illustrated in table 4 and are both located under $AIMAFSHOME/classlib and $AIMAFSHOME/aimweb, respectively. Indented entries indicate sub directories, a suffix slash (/) means directory, and a file name otherwise.

directory/file	Comment
bin/	executable binaries
AIX/	for AIX version
OS2/	for OS2 version
incl/	copies of .h files from src/
lib/	libraries
AIX/	for AIX version
OS2/	for OS2 version
obj/	.o object files
AIX/	for AIX version
OS2/	for OS2 version
src/	sources of .cpp and .h files
makefile.AIX	makefile for AIX version
makefile.OS2	makefile for OS2 version

Table 5. sub-directory structure of $AIMAFSHOME/classlib and $AIMAFSHOME/aimweb

Note that although the directory structure is laid out to accommodate both an AIX and an OS/2 image, only the AIX version has been implemented. In the sequel, the more interesting parts of the code base are described. First, some files for the aimweb part are explained.

directory/file	Comment
core/	communication functions
indexer/	AWindexer server
iadmin.lex	lexical analyzer for indexer command language
iadmin.yacc	parser source for indexer command language
indexer.h	class definition of parser class
indexer.cpp	class implementation of parser class
AWindexer.cpp	main() of the indexer server
README	command language and inheritance graph
locator/	AWlocator server
router/	AWrouter server
tagger/	AWtagger server
indmgr/	AWidxmgr client
locator/	AWlocator client
qrhhdlr/	AWqueryhdlr client

Table 6. some sub-directories and files under $AIMAFSHOME/aimweb/src/

Second, some files for the classlib part are discussed.

directory/file	Comment
sm2index/	wrapper for SearchManager/indexer communication
sm2index.h	class definition of wrapper class
sm2index.cpp	implementation of wrapper class
query.lex	lexical analyzer for 2nd step queries
query.yacc	parser source for 2nd step queries
query.h	class definition of parser class
query.cpp	class implementation of parser class

Table 7. some sub-directories and files under $AIMAFSHOME/classlib/src/

Whereas $AIMAFSHOME is AFS filespace, some files are located under $AIMLOCAL-HOME on a local disk (on machine thinker.almaden.ibm.com) for disk quota and performance reasons. This part does not exactly contain parts of the code base, but TREC related files and the SearchManager installation package. The word/frequency table is also found there:

directory/file	Comment
TREC/	TREC related files
README	information on how carry out experiments
bin/	command and perl scripts
queries/	the original TREC queries
results/	data from the experiments
presentation/	graphics, postscript files
qrels.1	relevant documents for AP* TREC queries
sm/	SearchManager installation package
940900-Version/	pre-beta version for AIX
950414-Update/	update with little installation support
950703-Version/	current version used in AIM
master.wgt	word/frequency table
master.sbj	subject numbers and their names

Table 8. some sub-directories and files under $AIMLOCALHOME/

8.2 Compilation

There is no single makefile for the AIM subject prototype. Instead, a series of makefiles are called from one master makefile. To compile all subsystems, cd into $AIMAFSHOME/ and issue `make`. The following table shows all makefiles in the AIM system:

directory/file	Comment
makefile	master makefile
aimweb/makefile.AIX	updates the aimweb part and the classlib part
aimweb/src/core/makefile.AIX	updates only the core part of aimweb
classlib/makefile.AIX	updates only the classlib part

Table 9. makefiles for AIM (all under $AIMAFSHOME/)

As indicated, the makefiles for the AIX version have the extension *.AIX. To compile only parts of the system, use `make -f makefile.AIX`.

8.3 SearchManager

8.3.1 SearchManager installation

The SearchManager software package comes in several parts. The newest version of the software can be ftp'd from `majestix.ae.boeblingen.ibm.com`. Login as user `issc`, cd to `/projects/issc` and look for the needed component. The login/password was no longer valid after August 1995, but a copy of the `/projects/issc` directory is on `thinker.almaden.ibm.com` under `$AIMLOCALHOME/sm/950703-Version/`. The driver package can be installed using smit on `$AIMLOCALHOME/sm/950703-Version/drv0703/sm6000.pkg`, but additional installation is required for the API (see `$AIMLOCALHOME/sm/950703-Version/api/`), the end user interface (see

$AIMLOCALHOME/sm/950703-Version/AdminEUI/), and some tools (see $AIMLOCALHOME/sm/950703-Version/tools/).

8.3.2 Creating a new SearchManager instance and search service

Assuming that SearchManager software is fully available on a computer, so-called SearchManager instances and search services have to be created. This is described in [IBM94], chapter 4. Note that the referenced version of the installation guide is out-dated, but no newer version has been published so far. In this section the relevant part of the creation process is summarized. Four steps are necessary:

1) chose index and working directory
2) create the SearchManager instance
3) update /etc/services
4) create the SearchManager search service

Before creating an instance, two directories have to be chosen for the index and the working directory. It is recommended to use two new file systems for that purpose. The file ownership and file protections of the directories have to follow the rules described in section 7.8.3. Clearly, these actions have to be performed under root authority.

To create a new SearchManager instance, the crtinst tool is used. Chose a name for the instance that contains only uppercase letters A-Z and digits 0-9. Then run crtinst with this name as a parameter. Do not use root, but a user that is part of the smadmin group.

```
% crtinst <instance name>
```

The index and the working directory names have to be filled in. If you want to delete the instance later, remove file /etc/SM6000/.<instance name>. If you are not permitted to run crtinst to its completion, check your access permissions to /etc/SM6000. They should be like

```
% ll -d /etc/SM6000/
drwxrwxr-x   2 root     smadmin       512 Oct  4 16:11 /etc/SM6000/
```

In /etc/services the known network services of the machine are given. To see the already defined search service names, look in /etc/services for lines commented with "SearchManager":

```
% grep -i searchmanager /etc/services
TONE            1624/tcp        TONES tone tones # SearchManager
TTWO            1625/tcp        TTWOS ttwo ttwos # SearchManager
```

Two search services named TONE and TTWO are found. A number of alternate names are given as well. For the new service, chose a search service name following the same rules as for the instance name, and find a port number > 1024 (port numbers < 1024 can only be used by processes running under root authority). Then add an entry like the one above to /etc/services. To update /etc/services you usually must be root.

Now, create a search service using the crtcfg tool. Again, the name of the newly created has to be specified as a parameter. Use a user that is part of the smadmin group.

```
% crtcfg <instance name>
```

The search service name as given in `/etc/services`, the corresponding port number and some execution characteristics have to be filled in the mask. If you are not permitted to run `crtcfg` to its completion, check your access permissions to the index and working directories.

8.3.3 SearchManager support

IBM's SearchManager product is developed and distributed by IBM-GSDL, Boeblingen, Germany. Volker Susok (sus@sdfvm1.VNET, T 01149-7031-16-6784) from GSDL has been very helpful for all sorts of questions. Additionally, Eric Johnson (ericj@bldvma.VNET, T 1-415-855-4306) from ISSC, Palo Alto is kind of responsible for getting the SearchManager code to the people.