TR-IT-0149

A comparative study of Query Reformulation methods on Vector-Space Models

Stephane Auberger Eiichiro Sumita Hitoshi Iida

January 31, 1996

Abstract

Relevance feedback is a well-known method developed to improve the effectiveness of information retrieval systems. It is based on the automatic and iterative improvement of the textual queries supplied by the users. After a brief overview of the system performance, this paper describes several different approaches and further refinement of a standard query reformulation method ("Ide-dec hi formula"). The main research was focused on using information about the origin of each particular term in the query ("modified Ide-dec hi formula") and especially information on terms in non-relevant documents ("Common Term System"). Also reviewed are less-expensive methods for decreasing the retrieval time as well as the size of query and document vectors ("fixed expansion size" and "fixed vector size"). In general, all methods are found to be effective in improving the performance on a standard test suite, the Virginia Disc One collections.

Contents

1 Introduction

Text retrieval systems are computer-based systems the function of which is locate user-requested documents in text databases. The requested documents which are commonly stored in electronic form may include news articles, technical abstracts, office memos, electronic mail messages, among many others. In fact, the recent widespread of such electronic documents has greatly enhanced the importance of text retrieval systems and made them vital components in modern information systems. A wide variety of operational text retrieval systems already exist, however, their performance vary significantly with respect to retrieval effectiveness and retrieval speed.

We used a parallel vector processing text retrieval system. This system is implemented on the KSR parallel computer; a MIMD multiprocessor machine which has large memory units capable of accommodating sophisticated document and query word weights. These systems are based on the vector space model in which documents and user queries are modeled as weighted vectors. The retrieval operation consists of scoring documents vectors as to how well they match the query vector, and then returning the top ranked documents to the user. The retrieval effectiveness of these systems is usually high by virtue of using weighted vectors and ranking. Furthermore, these systems can be implemented to yield fast retrieval speed as they are amenable to parallel implementations using large-scale computers, where a large number of vectors can be accommodated and processed in parallel.

An overview of the retrieval system and its implementation is given in section 1. Performance of the system with respect to retrieval effectiveness and retrieval speed is evaluated in section 2, as well as many different query reformulation methods in section 3. Concluding remarks and future work direction are given in section 4.

1.1 System Overview

The system consists of four components; text database, weighted vectors generator, weighted vectors database, and parallel query processor. The vectors generator and the text database are implemented on the system's host (a SUN Sparc workstation), and the weighted vectors database and the parallel query processor are implemented on the KSR parallel machine.

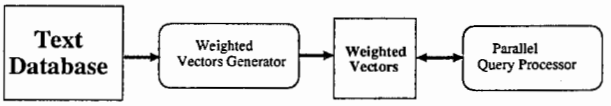


Figure 1: Bolck diagram of the system

The database contains the texts of the documents or articles to be searched. The generator operates directly on the text database to produce a much smaller weighted vectors database. The parallel query processor operates directly on the weighted vectors database by performing a parallel match operation between a given query vector and all document vectors. It also runs the *relevance feedback* operation by reformulating a new relevance feedback query depending on the query's relevance information.

1.2 Database Representation

The first step in the implementation of a vector-model text retrieval system is to represent each document in the given full-text by a weighted vector. The generation of a weighted vector is carried out on the host machine by running the following three pre-processing operations:

The weighted vectors generator consists of three main components; stop list words filter, suffix stripping stemmer, and word weights assignment function [1]. A brief description of each component is given below:

- The stop list filter removes from the text of each document or query the most frequently occuring words in English such as $(and, of, or, but, the, etc, ...)$. These words are poor discriminatories, and their removal would have no effect on the retrieval effectiveness. Moreover, the filtering process reduces storage requirements and increases query processing speed. The filter was applied using a stop list consisting of 425 words derived from the Brown Corpus [2].

- The suffix stripping stemmer replaces the words preserved by the stop list filter to their stem forms. For example, the stemmer replaces a variety of different forms such as *analysis, analyzing, analyzes,* and *analyzed* by a common word stem *analy.* The stemming operation reduces storage requirements since many words are replaced by a single stem word. Furthermore, it might increase the retrieval effectiveness since the stem word has a higher frequency of occurrence than that of the words replaced. In this system we used the well-known Porter stemming algorithm [3].

- The weight assignment function assigns a real-number weight to each word stem produced by the stemmer. The weight distinguishes the degree of importance of the word in the document (query), and thus leads to improved retrieval effectiveness. Moreover, it adds user-friendliness to the system as it facilitates ranking of the retrieved documents. In this system, we used the following weight assignment function for both the documents and the queries [4].

$$w_i = \frac{(0.5 + 0.5\frac{f_i}{\max f}).log\frac{N}{n_i}}{\sqrt{\sum_{j=1}^{j=W}((0.5 + 0.5\frac{f_j}{\max f})^2.(log\frac{N}{n_j})^2)}}$$

where,
w_i: weight of word i in the document (query).
f_i: frequency of occurrence of word i in the document (query).
n_i: number of documents(queries) to which word i is attached.
N: number of documents (queries) in the database.
W: total number of words in the document (query).

The denominator of the function above is a weight normalization component which ensures that the lengths of document (query) vectors are equal. The function assigns weights varying between 0 and 1, where 0 represents a word that is absent from the vector, and 1 represents a fully weighted word.

1.3 Query Processing

The parallel query processor scores documents vectors as to how well they match a given query vector, and then returns a ranked list the top scored documents. It also executes the computation-intensive relevance feedback method which is applied to improve retrieval effectiveness. A detailed description of the processor's operation is given below:

i. Read into the main memory of KSR the documents weighted vectors and the query weighted vector.

ii. Distribute the processing of the query vector matching operation on all KSR processing elements by assigning different document vectors to different processing elements. This assignment attempts to achieve evenly-balanced processing so as to assure high utilization of the parallel machine resources.

iii. Perform in parallel the query vector matching operation. This corresponds to performing an inner product between the query vector and each document vector. Every inner product operation produces a real-number score which corresponds to the similarity between the document vector and the query vector.

iv. Rank the documents in decreasing order of their similarity scores, and retrieve the top n documents to judge for their relevance to the example query. This experimental system runs in a batch mode operation, and thus the relevance judgment is made automatically be referring to a relevance information file which contains names of the example query's relevant documents (in an interactive mode of operation, however, the relevance judgment would be made by the user).

v. Reformulate the query vector by expanding and re-weighting its elements according to the following Ide dec-hi relevance feedback method [5]:

$$Q^{new} = Q^{old} + \sum All \ \ rel.docs \ - \ Top \ \ nonrel.doc$$

Q^{new} is the new query vector which is obtained by (1) adding to the previous query vector Q^{old} all words and corresponding weights of all relevant documents vectors, and (2) subtracting from the new query vector all words and corresponding weights found in the top ranked non-relevant document vector. Query reformulation using the Ide dec-hi method has proved to be superior to many other relevance feedback methods [6].

vi. Repeat steps $(iii - v)$ for a given number of iterations.

2 Relevance feedback using the Ide-dec hi formula

We evaluated the performance of the relevance feedback system using five experimental document collections (LISA, CACM, CISI, CRAN and MED) covering various subject areas and taken from the Virginia Disc One files - statistics about these collections are presented in Appendice (A) [7]. The performance data reported in this and the following sections are mainly those obtained using two collections only; the LISA library science collection which consists of 6004 documents and 35 example queries, and the MED collection, which consists of 1033 documents and 30 example queries. Results for the other collections are presented in Appendice. We conducted a series of retrieval experiments especially on the LISA collection, in each of which we used every query in the collection. We evaluated performance of the system with respect to its retrieval effectiveness and retrieval time.

2.1 Retrieval Effectiveness

Retrieval effectiveness of text retrieval systems is normally evaluated using the recall and precision measures. Recall is defined as the proportion of relevant documents that are retrieved from the document collection, and precision is defined as the proportion of retrieved documents that are relevant. All recall and precision ratios given in this paper were computed under the assumption that the top 20 documents retrieved in each search iteration were judged for relevance, and the weighted words contained in all the relevant documents and the top non-relevant document were used to reformulate the query vector.

To evaluate the true effectiveness of the relevance feedback process, it is necessary to compare the performance of the feedback iterations search with the results of an initial search performed with the original query vector by retrieving the same total number of documents (without the use of relevance feedback methods).

In addition to the average recall and precision measures over all the queries in the collection, the average query length (in number of terms, i.e. a stemmed word produced by the stemmer) and the average number of relevant documents retrieved were also computed. To represent the performance of the system by a single number, an average Recall-Precision measure was computed: it is proportional to the area below the well-known Recall-Precision graphs (to be precise it is the value of this area multiplied by 10^4).

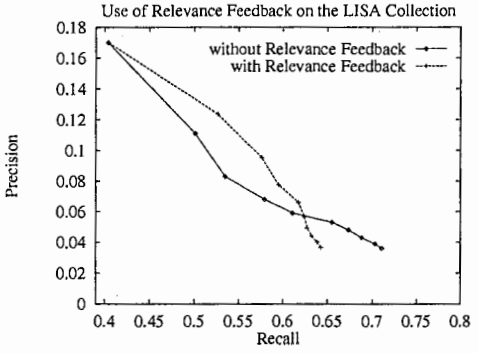
2.2 Retrieval Time

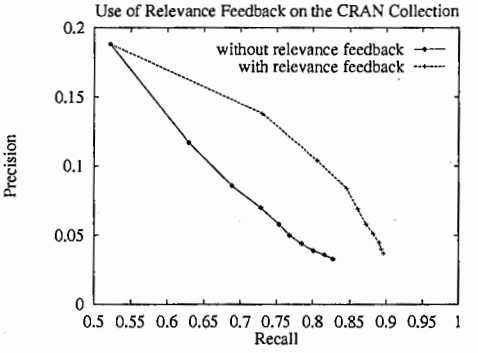
The measured retrieval time corresponded to the time spent by KSR to match the query vector against all the document vectors plus the time spent to rank the similarity scores in decreasing order.

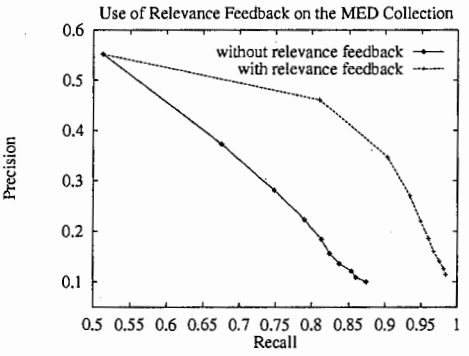
The experiment was conducted using either 25 processors or a single processor. A speedup ratio (S) was computed as Run Time(1PE)/Run Time(25PE). Dividing (S) by the total number of processors, gives the Speed-up ratio.

2.3 System Performance

Below are presented the retrieval graphs for LISA, CRAN and MED, and the other collections are in appendice (E). More complete results for LISA are given in Appendice (B).







On average, the relevance feedback system is very effective on every collection except the LISA collection. For CRAN, MED, CACM and CISI, the improvement in term of average Recall-Precision is respectively +65%, +64%, +69% and +84%. For LISA, although the performance with relevance feedback is good in the first couple of iterations, the recall increases very slowly then and after 100 documents are retrieved, the performance is lower than when using no relevance feedback process (the final average recall improvement is +2%). In other words retrieving 100 documents and judging them for relevance directly gives on average the same or better results than retrieving 100 documents using 10 or 20 search iterations of the relevance feedback (see graphs 4&5 in Appendice B).

2.4 Conclusion and Further Studies

Given the results using the Ide-dec hi formula, the following factors are of main concern:

- Accuracy: With respect to the number of feedback iterations, although it has been set beforehand, results on example queries show that the relevance feedback operation should be suspended once the previous iteration produced no new relevant documents. The average Recall and Precision measures taking into account only the previous option show a clearer separation between results with relevance feedback and without relevance feedback (see figure 8 in Appendice (B) for example). This beeing said, the problem is that the "no new relevant document" case occurs in general very early during the relevance feedback process: out of 35 queries, 30 already have been discarded after the second iteration (see table 1 Appendice (B)). As a result this technique prevents the system from achieving high recall and would leave the user very frustrated. The aim of the further studies will be to find a query reformulation that may give some improvement.

  With a closer study of example queries (for example query 35 in the LISA collection), one can see that very often the original focus of the query is completely lost because the terms in the original query generally appear in non-relevant documents and the decreasing of the term weight in that case is quite strong. This problem is particularly worrying when no new relevant item is discovered, in that case the query reformulation consists only in decreasing the weights of words that are in the top non relevant document. There is no relevant document(s) to compensate this fact by increasing the weights of these terms and the final effect is that these terms which, even if in non relevant documents, are none-less describing the focus of the query, are "removed" iteration after iteration in favor of terms that have much less to do with the focus of the query. This is one of the problems that we tried to solve in the further studies, notably in sections 3.1 and 3.2.

- Retrieval Time: The other purpose of the futher studies is to accelerate as much as possible the speed of the retrieval program. To decrease the retrieval time, we tried to bound the sizes of the different query and document vectors (sections 3.3 and 3.4).

3 Comparison of Query Reformulation Methods

3.1 Modified Ide-dec hi Formula

Previously, the query vector was expanded and re-weighted according to the Ide dec-hi formula:

- The original Ide dec-hi formula:

$$\begin{aligned} Q^{new} = \quad & Q^{old} \\ + \quad & \sum_{reldocs} (wt\ (terms\ in\ relevant\ documents) \\ - \quad & wt\ (Top\ nonrel\ doc) \end{aligned}$$

Several modifications were performed and tested on this original formula, using a modified Ide dec-hi formula, and introducing coefficients α, $\beta 1$, $\beta 2$ and γ:

-

$$\begin{aligned} Q^{new} = \quad & \alpha \times Q^{old} \\ + \quad & \sum_{reldocs} (\beta 1 \times wt_{\ terms\ in\ the\ old\ query} \\ & \quad + \beta 2 \times wt_{\ terms\ not\ in\ the\ old\ query}) \\ - \quad & \gamma \times wt_{\ terms\ in\ top\ non-relevant\ document} \end{aligned}$$

The introduction of coefficients on the modified formula allows to study the effectiveness of each part of the formula and compare the importance of each type of terms. Terms not in the old query are the terms that are only in the relevant documents (and not in the old query- i.e the query from the previous feedback iteration) and could be called "new terms".

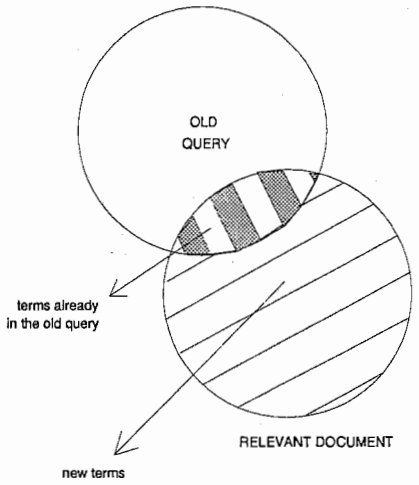


Figure 2: different types of terms in the expanded query

As one can see, the original IDE-DEC-HI formula is the modified IDE-DEC-HI formula whose coefficients are all equal to 1. The reformulated query was evaluated against the test set of documents. A large number of coefficients was experimented, computing Recall-Precision graphs on all collections and especially on LISA. From these experiments we can make the further assumptions:

- α should be put at 1.

- $\beta 2$ should be inferior to $\beta 1$. That is to say that terms in the original query ($\beta 1$) should have more importance than new terms ($\beta 2$, terms in relevant documents only).
- γ should be very small, showing that terms in non-relevant documents are relatively insignificant.

Although all collections have different responses, in general, the most effective batch of coefficients was (α= 1, $\beta 1$= 0.75, $\beta 2$= 0.5, γ= 0) and this batch was used in all later experiments. Assigning γ to 0 may seem rather strong at first but the experimental result in doing this produces always a real improvement. α= 1, $\beta 1$= 0.75, $\beta 2$= 0.5, γ= 0.25 is also a good choice on most collections (except on LISA).

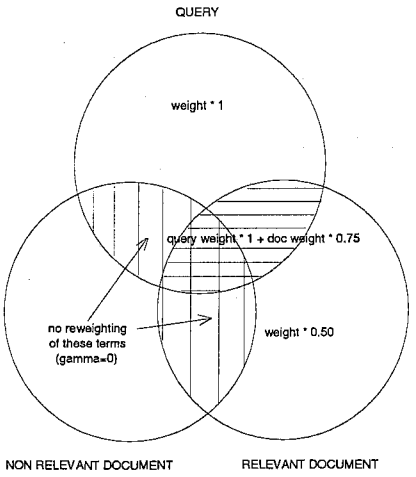


Figure 3: Modified Ide dec-hi formula

The results are given in the table below:

Method	*Average Recall-Precision*	*Improvement*	*Coefficients*
Without relevance feedback	273	-	-
Original Ide dec-hi formula	279	+ 2%	1, 1, 1, 1
Modified Ide dec-hi formula	404	+ 48%	1,.75,.5,0

Table 1: Original and modified Ide dec-hi formula (see Fig.1 Appendice E) * LISA

The list of coefficients gives the values of respectively α, $\beta 1$, $\beta 2$ and γ.

Retrieval times were also computed:

Method	*Run Time with 25 PE (sec)*	*Run Time with 1 PE (sec)*	*Speedup Ratio*	*Coefficients*	*average Query Size*
Original formula	0.542	12.24	22.6	1, 1, 1, 1	147.9
Modified formula	0.610	14.00	22.9	1,.75,.5,0	162.6

Table 2: Run Time for the Ide dec-hi formulas * LISA

In conclusion, the modified formula performs 45% better than the original formula on LISA, and requires 13% more retrieval time. As one can see the query size is larger for the modified formula because it is retrieving more relevant documents, therefore allowing more query expansion. As a result, this increase in query size naturally leads to a proportional increase in the retrieval time - see Appendice (E) for more

details on this subject.

Other Collections: On the CACM collection the modified formula performs 12% better than the original one. On the other collections the improvement is: +5% (CISI), +4% (CRAN), +0.5% (MED) [see the corresponding Recall-Precision graphs].

3.2 Common-Term System

The previous results show that assigning 0 to the coefficient γ improves retrieval effectiveness. The only problem is that it means using no informations from non-relevant documents in the relevance feedback process (in the original formula we subtracted from the query vector all weights corresponding to the words found in the top ranked non-relevant document).

The idea behind the Common-Term System is once again to distinguish between the terms that are in the original (old) query and the terms that are added to the query because they are in relevant documents. As a matter of fact, terms in the query at the beginning of the relevance feedback iteration have been proven more important than terms *exclusively* in relevant documents (see the effectiveness of the modified Ide dec-hi formula). Therefore this system keeps track of the 'origin' of the term, and allows the reduction of the weight only for terms that are both in relevant and non-relevant documents, but not in the 'unexpanded' (i.e. old) query. It has also been tried to assign 0 to the weight of these 'common' terms (if they are in relevant and non-relevant documents, but not in the query, they may have most likely little to do with the actual query).

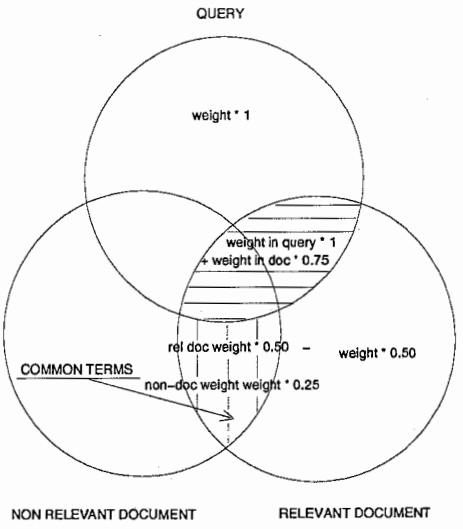


Figure 4: Common Term System (modified formula)

The results are given is the table below: * means "using the common term system".

Method	*Average Recall-Precision*	*Improvement*	*Coefficients*
Original formula	279	-	1,1,1,1
With common term weight modified	381	+ 37%	1,1,1,1*
With common term weight to 0	373	+ 34%	1,1,1,-*
No use of non-relevance info	376	+ 35%	1,1,1,0

Table 3: Common Term System, Original Ide dec-hi formula (1,1,1,1) (see Fig.6 App E) LISA

The Common Term System present a significant improvement over the original formula: The (1,1,1,1*) batch performs 37% better than the (1,1,1,1) batch (original formula), confirming our hypothesis regarding the importance of terms in the old query. However, using no relevance information at all (1,1,1,0) produce approximately the same results in term of average Precision-Recall (+35%).

The same experiments were conducted on the modified formula:

Method	*Average Recall-Precision*	*Improvement*	*Coefficients*
Modified formula	404	-	1,.75,.5,0
With common term weight modified	411	+ 1.6%	1,.75,.5,.25*
With common term weight to 0	409	+ 1.2%	1,.75,.5,-*

Table 4: Common Term System, Modified Ide dec-hi formula(1,.75,.5,0)(see Fig.7 App E) LISA

These results confirm the fact that using a common-term system or using no non-relevance information (γ=0) produce approximately the same performance, with again a very slight advantage to the common-term system.

Method	*Run Time 1 PE (sec)*	*Run Time 25 PE (sec)*	*Speedup Ratio*	*Coefficients*	*average Query Size*
Original formula	0.542	12.24	22.6	1,1,1,1	147.9
Common term weight modified	0.599	13.66	22.8	1,1,1,1*	159.6
No use of non-relevance info	0.604	13.66	22.8	1,1,1,0	159.0

Table 5: Run Time for the common term system on the Ide dec-hi formula * LISA

In comparison to the original formula, the retrieval time is 12% higher in the case of both the common term system and the (1,1,1,0) batch.

MED Collection: For the MED collection, the common term system gives the same performance than the original Ide dec-hi formula (less than 1% of improvement or deterioration) [see fig.12 Appendice E].

In conclusion, the common term system offer a little improvement (around 1%) over the (1,1,1,0) batch of coefficients (which is the original Ide dec-hi formula using no information from non-relevant documents during the query reformulation process), and requires the same retrieval time. It is 12% more time-consuming than the original formula, but this is once again due to the fact that the query is larger.

3.3 Fixed Expansion Size

Relevance feedback is based on the continuous expansion of the original query, by adding to the query vector all words and corresponding weights of all relevant document vectors. Therefore, the size of the query expansion and the choice of the terms added to the query is of main interest. In these experiments, we tried to limit the number of terms added to the query: First by restricting the expansion size to a given number of terms (5, 10, 20 terms only). Or seconds by using only a certain percentage of the added terms (33%, 50%, 67%, 75%). In each case, the terms in the relevant documents were ranked by weight and only the *highest weighted* terms were added. A last option tested was using no query expansion at all, only re-weighting the terms in the query.

Method	*Average Recall-Precision*	*Improvement*	*Average Query Size*
Add ALL terms	279	-	147.9
No Query Expansion	246	- 12%	20.9
Add 10 terms only	221	- 21%	60.7
Add 20 terms only	264	- 5%	101.0
Add 50% of the terms only	231	- 17%	92.5
Add 75% of the terms only	276	- 1%	130.7

Table 6: Comparison with the ORIGINAL Ide dec-hi formula (1,1,1,1) * LISA (see Fig.10 App E)

Method	*Average Recall-Precision*	*Improvement*	*Average Query Size*
Add ALL terms	404	-	162.5
No Query Expansion	372	- 8%	20.6
Add 5 terms only	331	- 18%	43.3
Add 10 terms only	350	- 13%	66.8
Add 20 terms only	390	- 3%	111.1
Add 33% of the terms only	348	- 14%	75.0
Add 50% of the terms only	367	- 9%	104.1
Add 67% of the terms only	390	- 3%	132.3
Add 75% of the terms only	405	+ 0%	144.3

Table 7: Comparison with the MODIFIED Ide dec-hi formula (1,.75,.5,0) * LISA (see Fig.8&9 App E)

The advantages of such methods are that they require less data and less retrieval time:

Method	*Run Time with 1 PE (sec)*	*Run Time with 25 PE (sec)*	*Speedup Ratio*	*Coefficients*	*average Query Size*
Original formula	0.542	12.24	22.6	1, 1, 1, 1	147.9
Add 10 terms	0.229	5.17	22.6	1, 1, 1, 1	60.7
Add 20 terms	0.367	8.35	22.8	1, 1, 1, 1	101.0
Add 50% of the terms	0.339	7.47	22.0	1, 1, 1, 1	92.5
Add 75% of the terms	0.467	10.62	22.7	1, 1, 1, 1	130.7
No Query Expansion	0.068	1.42	20.7	1, 1, -, 1	20.6

Table 8: Run Time for the ORIGINAL Ide dec-hi formula * LISA Collection

For the LISA collection, the more added terms you use the better the result is. Using the first 5 or 10 terms, or up to the first 50% of the terms requires much less retrieval time (40 to 60% less than the full

expansion), but presents a large deterioration of the performance in return (more than 10%). Surprisingly, using no query expansion is more preferable, giving a better performance than the above-mentioned techniques, requiring an extremely low retrieval time (-90%!) [unfortunately, this is very LISA-specific and is not true for all other collections]. Also surprising is the poor efficiency of the first added terms and the large improvement between adding 10 terms or 50% and adding 20 terms or 75% of them, signifying that latter terms are more important than the highly weighted ones. A good choice, however, is using 75% of the terms, and 67% or the first 20 terms, although the performance falls a little bit. They allow a significant gain of time (13 to 30%) for almost the same performance than a fully expanded query (especially for the 75% choice).

In all cases, the speedup ratio remains steady at 22.7.

MED Collection: The results on this collection are given in the tables below:

Method	*Average Recall-Precision*	*Improvement*	*Average Query Size*
Add ALL terms	207	-	568.0
No Query Expansion	145	- 29.7%	10.9
Add 5 terms only	157	- 24.3%	86.1
Add 10 terms only	195	- 6.0%	165.0
Add 20 terms only	207	- 0.1%	299.4
Add 33% of the terms only	206	- 0.4%	267.7
Add 50% of the terms only	214	+ 3.3%	375.7
Add 67% of the terms only	201	+ 1.4%	463.1
Add 75% of the terms only	209	+ 0.9%	500.6

Table 9: Comparison with the ORIGINAL Ide dec-hi formula (1,1,1,1) * MED Collection

Method	*Average Recall-Precision*	*Improvement*	*Average Query Size*
Add ALL terms	208	-	569.1
No Query Expansion	146	- 29.7%	10.9
Add 5 terms only	171	- 17.9%	88.7
Add 10 terms only	203	- 2.5%	167.8
Add 20 terms only	213	+ 2.4%	302.8
Add 33% of the terms only	210	+ 1.1%	269.7
Add 50% of the terms only	217	+ 4.1%	377.1
Add 67% of the terms only	214	+ 2.8%	465.9
Add 75% of the terms only	211	+ 1.6%	502.2

Table 10: Comparison with the MODIFIED Ide dec-hi formula (1,.75,.5,0) * MED (Fig 13&14 App E)

These tables show that using no query expansion or the first 5 terms gives poor results. Selecting a fraction of the added terms seems a very good technique, and for 50% and up, the average Recall-Precision is higher than the one for the full query expansion. The excellent results obtained are not so surprising. The average size of the MED query is very high (570 terms) and the documents don't contain more than 199 terms (see table 11). The query is so big that it tends to cover (too) many topics, therefore a reduced query is as good if not better as a fully expanded one.

Overall, we can draw the same conclusion as for the LISA collection (except for the performance of the non-expanded query which is rather poor in MED): using the first 20 terms or a given percentage of the added terms produces approximatly the same, if not a slightly better performance while requiring much less retrieval time. Moreover, for both collections, the modified formula gives slightly better results than the original one.

3.4 Fixed Vector Length

As we have seen before, the relevance feedback process tends to be very costly in term of memory, and this leads to high retrieval times. Therefore, the aim of the following experiments is to limit the size of the vectors representing both documents and queries.

- Documents: The size of the document vectors is limited according to a threshold value, and the lowest weighted terms are removed (the reduced vector is used for matching, query expansion, etc..).
- Queries: We previously tried to limit the expansion size, but the selection concerned the *added* terms only. In these experiments, all terms are added to the query, and *then* the query is sorted and only the highest weighted terms are kept.

collection	*number of documents*	*average (& maximum) document length*	*number of queries*	*average query size iteration 0*	*average query size all iterations*
LISA	6004	34.7 (95)	35	20.6	147.9
MED	1033	53.3 (199)	30	10.9	568.0

Table 11: DOCUMENTS & QUERIES: average sizes (all lengths in number of terms)

The results are shown in the table below:

maximum Document size	*maximum Query size*	*average recall-precision*	*Improvement*	*average query size*
95 (all)	all	279	-	147.9
95	40	270	- 3.2%	35.1
95	60	284	+ 2.1%	49.0
95	80	287	+ 6.7%	61.9
95	100	293	+ 5.1%	73.6
95	140	292	+ 4.7%	92.1
20	80	234	- 16.2%	54.9
60	80	296	+ 6.2%	61.9
60	60	281	+ 0.7%	49.0
80	60	283	+ 1.4%	49.0

Table 12: FIXED VECTOR LENGTH - Original Ide dec-hi formula * LISA Collection (Fig. 11 App. E)

This method is much more powerful than the fixed expansion size because it makes the selection on a much more global level, i.e. the fully expanded query. Again one can notice in the previous tables that the average query size during the relevance feedback process is quite big compared to the average (or even maximum) size of the documents. Therefore it is not surprising to have similar results even when heavily bounding the size of the query vector (this is particularly true for MED, as we will see later).

maximum Document size	*maximum Query size*	*Run Time with 25 PE (sec)*	*Run Time with 1 PE (sec)*	*Speedup Ratio*
Ide dec-hi	original	0.542	12.24	22.6
60	60	0.160	3.55	22.1
80	60	0.161	3.59	22.3

Table 13: Run Time - Fixed vector length * LISA

The two selections (60,60) and (80,60) of threshold values perform approximately 1% better in term of effectiveness, and offer a gain of 70% in term of retrieval time.

Space Overhead: We calculated the document and the query overhead, taking into account the number of terms in the queries and documents.

maximum Document size	*maximum Query size*	*average Document size*	*improvement*	*average Query size*	*change*
Ide dec-hi	original	34.7	-	147.9	-
60	60	34.4	- 1%	48.8	- 67.0%
80	60	34.7	-	48.8	- 67.0%

Table 14: Space overhead - Fixed vector length (length in terms) * LISA

The best thing to do is to bound the size of the query as much as possible: it affects every matching operation whereas a threshold value on documents concerns only a certain number of them. Moreover, bounding the size of documents too much is dangerous and recall decreases (particularly in the first iterations). Therefore bounding the query size is a good way to save a lot of retrieval time.

MED Collection:

maximum Document size	*maximum Query size*	*Average Recall-Precision*	*Improvement*	*Average Query Size*
200 (all)	all	207	-	568.0
200	10	196	- 5.1%	10.1
200	50	204	- 1.5%	46.1
200	70	206	- 0.4%	64.1
200	100	208	+ 0.7%	91.1
200	200	209	+ 0.8%	180.8
80	60	201	- 2.7%	55.1
80	70	205	- 0.9%	64.1
80	80	207	- 0.1%	73.1
50	70	200	- 3.5%	64.1
100	70	204	- 1.4%	64.1

Table 15: FIXED VECTOR LENGTH - Original Ide dec-hi formula * MED (Fig. 15 App. E)

Run time & Space Overhead:

maximum Document size	*maximum Query size*	*Run Time with 25 PE (sec)*	*Run Time with 1 PE (sec)*	*Speedup Ratio*
Ide dec-hi	original	0.553	11.594	21.0
80	80	0.063	1.381	21.9

Table 16: Run Time - Fixed vector length * MED

maximum Document size	*maximum Query size*	*average Document size*	*improvement*	*average Query size*	*improvement*
Ide dec-hi	original	53.3	-	568.0	-
80	80	50.9	- 5%	73.1	- 87%

Table 17: Space overhead - Fixed vector length * MED

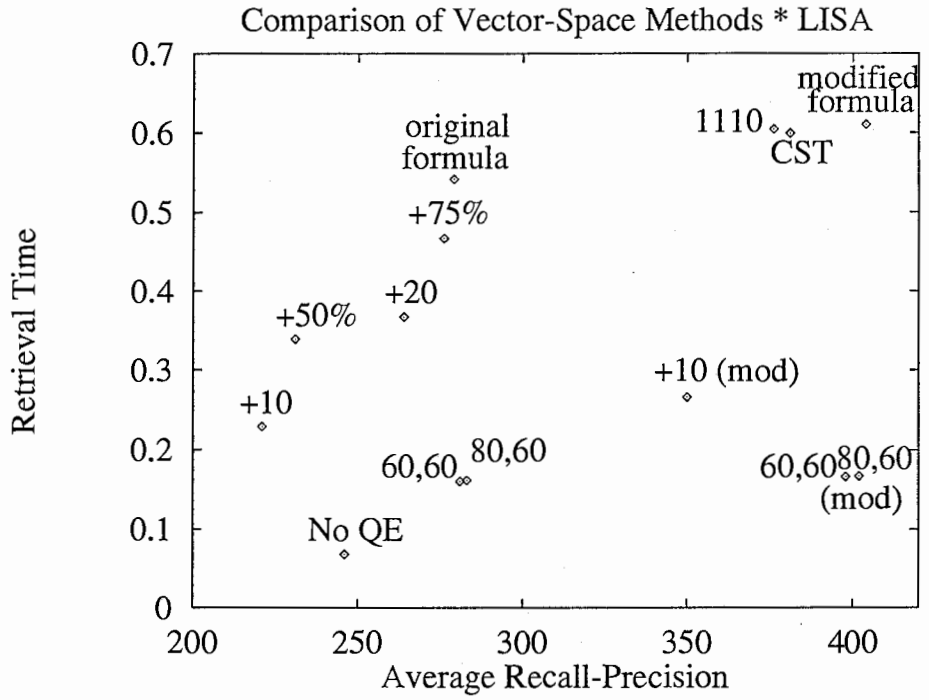
Although there is no improvement (the relevance feedback process performs originally very well on the MED collection), bounding the size of the documents and query vectors offer and large improvement with respect to the retrieval time and space overhead. Again, this is mainly due to the fact that we are bounding the query size. Moreover, for MED, the bounded query performs better in the first iterations but present a lower recall at the last iteration.

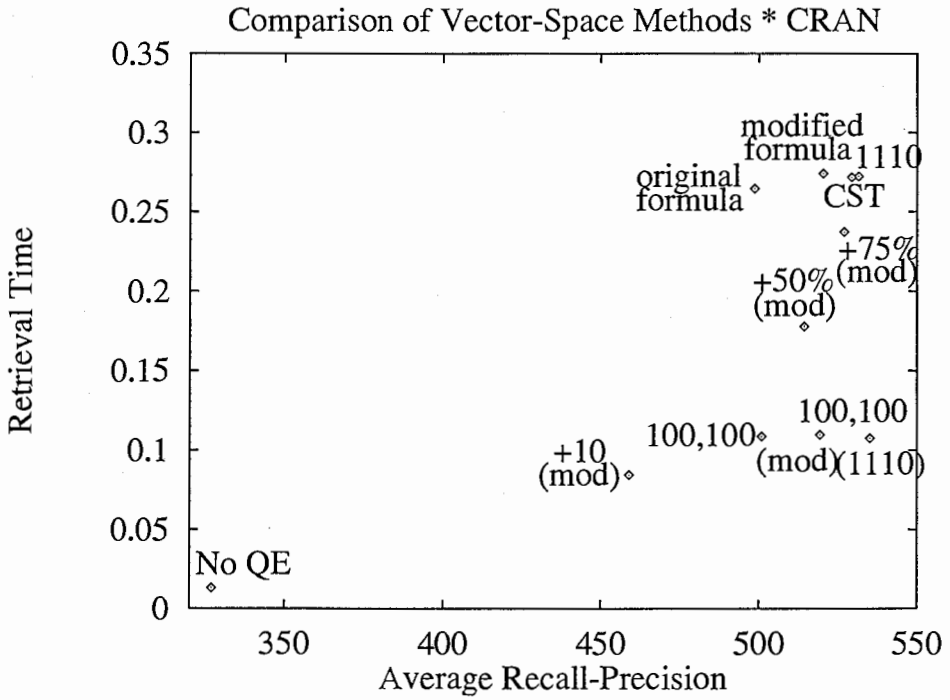
Using the modified formula (LISA & MED)

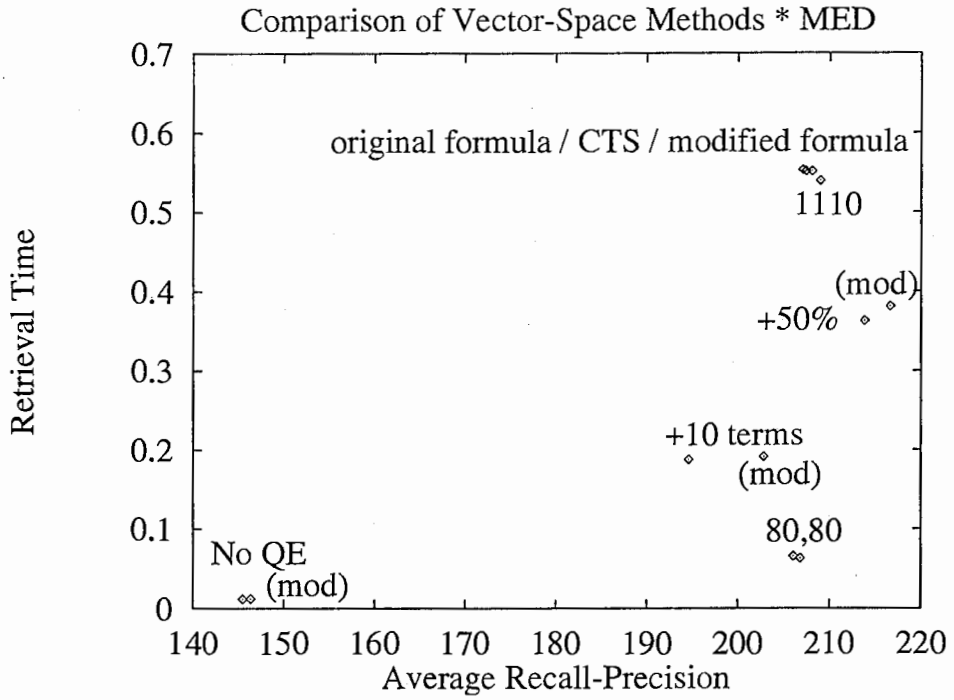
This improvement in retrieval time can be applied on the modified Ide-Dec-Hi formula which present an excellent pormance in average Recall-Precision. On LISA, bounding the query size to 60 terms and the document size to 80 terms, presents the same performance in average recall precision than the normal modified formula and gives a 73% gain in retrieval time. On MED, the (80,80) batch of coefficients requires 89% retrieval time for a 1% decrease in average recall-precision.

4 Conclusion and Future Work

The graphs below show the relationship between accuracy (average Recall-Precision) and retrieval time for the various vector space methods used in this paper. Collections tested are LISA, CRAN and MED:







CTS	Common Term System.
+50%	Fixed Expansion Size, add only 50% of the terms.
+10	Fixed Expansion Size, add only the first 10 terms.
No QE	No Query Expansion.
1110	No Use of non-relevant document information (α, $\beta 1$, $\beta 2$ =1, γ=0).
40,60	Fixed Vector Length: gives respectively the *documents* and *query* threshold values.
(mod)	indicates that the query expansion method was performed using the modified formula.
	If nothing is specified, it means that the original ide dec-hi formula was used.

Once again, the results are still very dependent on the collection, however all methods produce an improvement of some sort and the following general assumptions can be made:

- The modified formula produces a better performance in term of retrieval effectiveness.
- The common term system gives a very small improvement over the formula with which it is used.
- By selecting only a given number of the added terms (fixed expansion size), we can save a lot of retrieval time.
- The fixed vector length method is preferable to the previous one because it gives a lower retrieval time with no loss of the performance. Using this technique on the modified formula produces the best result in both efficiency and effectiveness.

Future work could consists of combining the different query reformulation techniques described in this paper, for example applying the common term system to the fixed size methods. Detailed performance using the TREC collection should also be investigated. Our ultimate goal is to develop an effective and efficient retrieval system to support the on-going massively paralell example-based spoken language translation project at ATR Interpreting Telecommunication Research Laboratories.

A Statistics on the Virginia Disc One Collections

We measured the sizes of each collection in the Virginia Disk One Collections.

Six collections were considered: ADI, CACM, CISI, CRAN, LISA and MED.

The items considered were the minimum, maximum and average lengths of the vectors representing the queries and documents, as well as the standard deviation of these vector lengths. Also measured were the number of vectors in the collection and the total size of the collection. These lengths were measured twice:

1. Before the filtering operation.
2. After the weighting operation (therefore after the consecutive filtering, stemming, counting and weighting operations).

The lengths were measured:

i. In number of bytes. This is the number of bytes of the whole file.

ii. In number of words or terms.
A word is considered as in its everyday meaning and concerns the documents in their original forms (a word repeated twice in the text will be counted twice). A term is the component of a file after the filtering, stemming, counting and weighting operations. It is the result of a stemmed word - in that way, different words, identical or not, can be represented by the same term. Therefore the number of terms is in general significantly lower than the number of words thanks to the filtering and stemming techniques.

Here is an example:

- Before the filtering operation, we consider here the original document. For example, considering the document file number 5458 of the LISA collection,

  5458
  LITERARY PERIODICALS.
  REVIEWS 46 SELECTED LITERARY PERIODICALS.
  ****

  This document has 9 words (115 bytes).

- After the filtering,stemming, counting and weighting operations and considering the same example, the file is now stored the following way:

  6060 0.800520
  7796 0.416218
  8866 0.296909
  9300 0.312689

  The first column contains the code number associated to a term, the second column contains the corresponding weights of this term.

  This document has 4 terms (58 bytes).
  (note that the number of bytes takes the weights into account contrary to the number of terms)

Note: some documents contains numbers. For example CISI document number 1:

.I 1
.T
18 Editions of the Dewey Decimal Classifications
.A
Comaromi, J.P.
.W
The present study is a history of the DEWEY Decimal Classification. The first edition of the DDC was published in 1876, the eighteenth edition in 1971, and future editions will continue to appear as needed. In spite of the DDC's long and healthy life, however, its full story has never been told. There have been biographies of Dewey that briefly describe his system, but this is the first attempt to provide a detailed history of the work that more than any other has spurred the growth of librarianship in this country and abroad.
.X
1 5 1
92 1 1
262 1 1
556 1 1
1004 1 1
1024 1 1
1024 1 1
This numbers were counted as words in the statistics (these numbers are removed through the filtering operation, consequently they don't appear in the statistics after weighting).

This first table gives the number of DIFFERENT terms in the whole collection, before filtering and after weighting (in this table only, a word repeated twice or more in the text will be counted only *once*). We computed the compression ratio defined by:

$$compressionratio = \frac{number\ of\ different\ terms\ after\ weighting}{number\ of\ different\ terms\ before\ filtering}$$

Collection	*number of different terms before filtering*	*number of different terms after weighting*	*compression ratio*
ADI	1780	996	0.560
CACM	30158	7223	0.240
CISI	23289	6725	0.289
CRAN	16661	5303	0.318
LISA	42236	11739	0.278
MED	21041	8826	0.419

Table 18: Number of different terms in collection

Table 19: QUERIES: *(All lengths in number of WORDS - Before FILTERING)*

Collection	*Number of vectors*	*Total Size of the collection*	*Average Length*	*Standart Deviation of vector length*	*min length of a vector*	*max length of a vector*
ADI	35	604	17.26	7.39	7	38
CACM	64	2064	32.25	16.91	11	94
CISI	112	10180	90.89	69.67	7	361
CRAN	225	4719	20.97	7.12	9	49
LISA	35	1986	56.74	21.81	19	110
MED	30	679	22.63	14.58	5	65

Table 20: DOCUMENTS: *(All lengths in number of WORDS - Before FILTERING)*

Collection	*Number of vectors*	*Total Size of the collection*	*Average Length*	*Standart Deviation of vector length*	*min length of a vector*	*max length of a vector*
ADI	82	5897	71.91	21.58	29	194
CACM	3204	401264	125.24	107.34	28	1566
CISI	1460	439886	301.29	155.06	34	1005
CRAN	1400	266299	190.21	89.95	45	708
LISA	6004	530513	88.36	37.21	9	309
MED	1033	161925	156.75	82.25	23	652

Table 21: QUERIES:*(All lengths in BYTES - Before FILTERING)*

Collection	*Number of vectors*	*Total Size of the collection*	*Average Length*	*Standart Deviation of vector length*	*min length of a vector*	*max length of a vector*
ADI	35	4076	116.46	51.23	48	248
CACM	64	13683	213.80	113.77	73	621
CISI	112	66632	594.93	455.78	48	2204
CRAN	225	28111	124.94	44.95	50	278
LISA	35	14000	400.00	157.68	117	779
MED	30	4608	153.60	105.65	28	444

Table 22: DOCUMENTS: *(All lengths in BYTES - Before FILTERING)*

Collection	*Number of vectors*	*Total Size of the collection*	*Average Length*	*Standart Deviation of vector length*	*min length of a vector*	*max length of a vector*
ADI	82	37158	453.15	142.88	167	1303
CACM	3204	2187740	682.82	566.78	119	6486
CISI	1460	2119351	1451.61	643.81	172	5107
CRAN	1400	1648226	1177.30	556.75	282	4390
LISA	6004	3849946	641.23	244.49	115	2031
MED	1033	1091357	1056.49	548.90	174	4219

Table 23: QUERIES: *(All lengths in number of TERMS - After WEIGHTING)*

Collection	*Number of vectors*	*Total Size of the collection*	*Average Length*	*Standart Deviation of vector length*	*min length of a vector*	*max length of a vector*
ADI	35	261	7.46	3.11	3	17
CACM	64	973	15.20	7.23	4	37
CISI	112	3547	31.67	23.04	3	99
CRAN	225	2012	8.94	3.18	3	21
LISA	35	721	20.6	8.60	9	41
MED	30	327	10.90	6.73	2	30

Table 24: DOCUMENTS: *(All lengths in number of TERMS - After WEIGHTING)*

Collection	*Number of vectors*	*Total Size of the collection*	*Average Length*	*Standart Deviation of vector length*	*min length of a vector*	*max length of a vector*
ADI	82	2189	26.70	8.45	14	70
CACM	3204	92935	29.01	20.00	7	155
CISI	1460	67228	46.05	19.18	9	165
CRAN	1400	78231	55.89	22.46	14	159
LISA	6004	208441	34.72	13.51	4	95
MED	1033	55099	53.34	24.44	9	199

Table 25: QUERIES:*(All lengths in number of BYTES - After WEIGHTING)*

Collection	*Number of vectors*	*Total Size of the collection*	*Average Length*	*Standart Deviation of vector length*	*min length of a vector*	*max length of a vector*
ADI	35	3201	91.46	38.13	37	207
CACM	64	12417	194.02	92.65	49	474
CISI	112	56370	414.02	301.38	39	1288
CRAN	225	25770	114.53	40.72	39	269
LISA	35	9189	262.94	109.77	116	521
MED	30	4111	137.03	84.40	25	376

Table 26: DOCUMENTS: *(All lengths in number of BYTES - Aft WEIGHTING)*

Collection	*Number of vectors*	*Total Size of the collection*	*Average Length*	*Standart Deviation of vector length*	*min length of a vector*	*max length of a vector*
ADI	82	28247	344.48	109.19	179	904
CACM	3204	1281664	400.02	277.35	95	2146
CISI	1460	930048	637.02	265.42	125	2287
CRAN	1400	1077721	769.80	309.37	14	2185
LISA	6004	2920772	486.47	189.40	54	1333
MED	1033	764401	739.98	339.06	125	2764

B Evaluation of relevance feedback for the LISA collection.

We studied the effectiveness of relevance feedback in term of precision and recall using all the 35 example queries from the LISA library science collection. The performance of the system was evaluated through three different experiments.

The first experiment consisted in retrieving 20 new documents at each iteration, considering 10 iterations of relevance feedback (that is to say retrieving a total number of 200 documents in all). In addition to the average recall and precision measures over the 35 queries in the collection, the average query length (in terms) and the average number of relevant documents retrieved were also computed. The results were compared to the results achieved by retrieving the same total number of documents, but without the use of relevance feedback methods.

The second and third experiments followed the same principle, but retrieving respectively 10 new documents at each iteration (over 10 iterations) and 5 new documents at each iteration (over 20 iterations)- both experiments retrieving 100 documents in all.

System Performance:

The results are given in the Recall-Precision figures 4 to 6. The following main assumptions can be made:

- On average, the relevance feedback system is not very effective on the LISA collection. In other words retrieving 100 documents (for example) and judging them for relevance directly gives on average the same or better results than retrieving 100 documents using 10 or 20 search iterations of the relevance feedback.
- A comparison of the three graphs shows that relevance feedback is more effective when a large number of documents is retrieved and judged for relevance in each iteration: As a general rule executing 5 iterations using the top 20 documents retrieved to reformulate the query vector gives better results in the end than executing 20 iterations using the top 5 documents retrieved to reformulate the query vector.
- With respect to the number of feedback iterations, although it has been set beforehand, results on example queries show that the relevance feedback operation should be suspended once the previous iteration produced no new relevant documents. The average Recall and Precision measures taking into account only the previous option are shown in tables 1 to 3 and their corresponding graphs (figures 7 to 9).

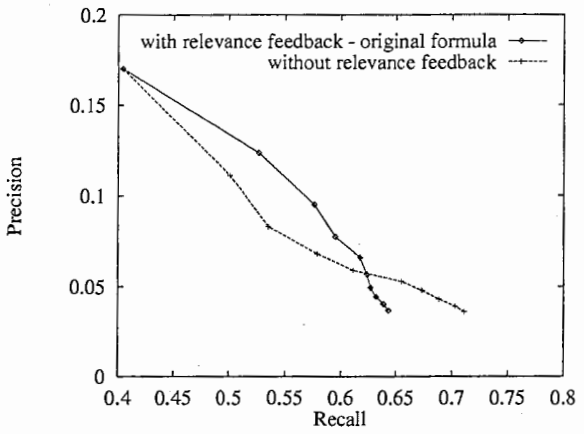


Figure 5: Use of relevance feedback on the LISA collection - Retrieving 20 documents per iteration

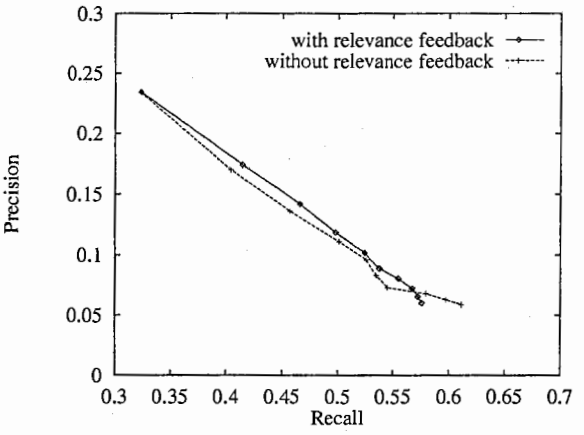


Figure 6: Use of relevance feedback on the LISA collection - Retrieving 10 documents per iteration

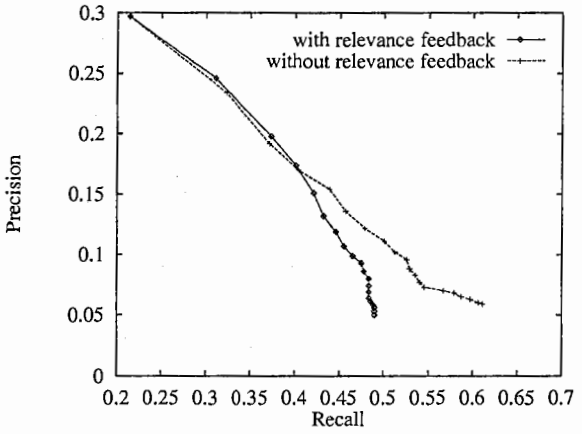


Figure 7: Use of relevance feedback on the LISA collection - Retrieving 5 documents per iteration

Case when the relevance feedback operation was suspended automatically when the top retrieved documents were judged to include no relevant documents:

Table 1: Retrieving 20 documents per iteration:

total number of retrieved docs	*average Recall*	*average Precision*	*number of queries computed*	*iteration of feedback*
20	0.404	0.170	35	0
40	0.599	0.191	20	1
60	0.560	0.168	12	2
80	0.571	0.200	5	3
100	0.664	0.207	4	4
120	0.631	0.233	2	5
140	0.755	0.286	1	6
160	0.830	0.275	1	7
180	0.849	0.250	1	8
200	0.906	0.240	1	9

Table 2: Retrieving 10 documents per iteration:

total number of retrieved docs	*average Recall*	*average Precision*	*number of queries computed*	*iteration of feedback*
10	0.323	0.234	35	0
20	0.536	0.325	16	1
30	0.571	0.312	11	2
40	0.648	0.329	6	3
50	0.701	0.330	4	4
60	0.547	0.483	1	5
70	0.623	0.471	1	6
80	0.642	0.425	1	7
90	0.660	0.389	1	8

Table 3: Retrieving 5 documents per iteration:

total number of retrieved docs	*average Recall*	*average Precision*	*number of queries computed*	*iteration of feedback*
5	0.214	0.297	35	0
10	0.456	0.444	16	1
15	0.492	0.496	9	2
20	0.512	0.500	7	3
25	0.586	0.472	5	4
30	0.585	0.433	3	5
35	0.377	0.571	1	6
40	0.396	0.525	1	7
45	0.434	0.511	1	8
50	0.491	0.520	1	9
55	0.509	0.491	1	10
60	0.528	0.467	1	11

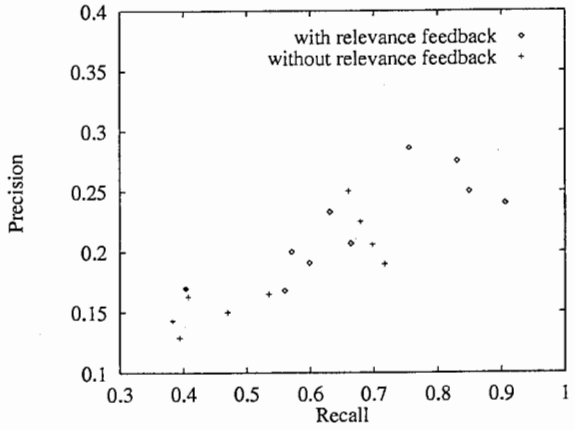


Figure 8: Use of relevance feedback on the LISA collection - Retrieving 20 documents per iteration

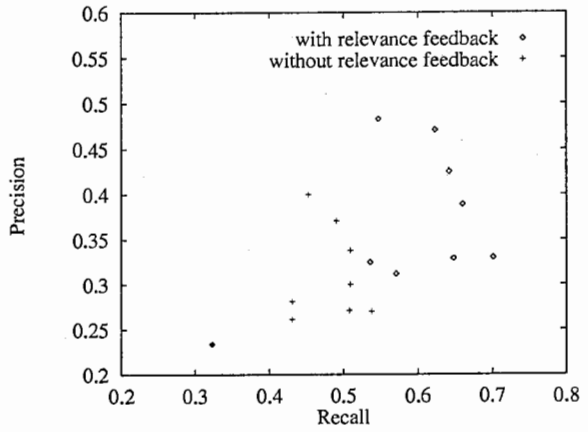


Figure 9: Use of relevance feedback on the LISA collection - Retrieving 10 documents per iteration

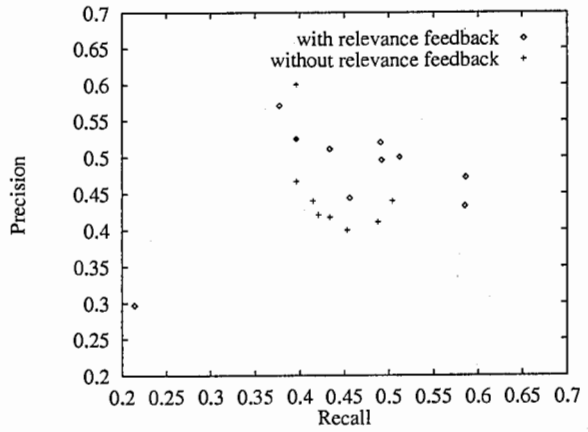


Figure 10: Use of relevance feedback on the LISA collection - Retrieving 5 documents per iteration

C Implementation details

- **Modified Ide-dec hi formula:** See file make_query_modified.c

  This formula is very easily implemented by adding coefficients to the various term weights:

  - $\beta 1 = 0.75$ is done with the line:
    (query_cod+j)->weight = (query_cod+j)->weight + .75(doc_cod+i)->weight;*

  - $\beta 2 = 0.5$ is done with the line:
    (query_cod+qcod_num+new_term)->weight= .5(query_cod+qcod_num+new_term)->weight;*

  - $\gamma = 0$ is done with the line:
    / delete_from_relevance_documents(init_params, queries, iterations); */*

- **Common-Term System:** See file make_query_common.c

  The query is copied in a 3 fields structure (q_large whose type is COD_DATA_LARGE) during the relevance feedback process -one field more than the previous structure (COD_DATA size is 18 bytes whereas COD_DATA_LARGE size is 24 bytes). This new field of type int ("*seen*") is used to store the "origin" of the term.

  In the query reformulation process, the first step is to use informations from relevant documents to modify query weights and add new terms. During this step, the query vector is copied in the enlarged structue through:
  small_to_large (query_cod,q_large,qcod_num);

  The field *seen*, initialized at 1 is then modified during the use of information from relevant documents: new terms have their *seen* field assigned to 3, terms from the query whose weight is modified have their *seen* field assigned to 2.

  During the use of the top non-relevant document step, only common terms are affected by the following operation:
  if (((q_large+j)->seen)==3)
  (q_large+j)->weight = (q_large+j)->weight - ((doc_cod+i)->weight);

  At the end of the function, the modified and/or enlarged query vector is restored in the initial format used in all remaining programs.
  large_to_small (q_large,query_cod,qcod_num + new_term);

  (Of course processor.h was modified to create the COD_DATA_LARGE type).

- **Fixed expansion size:** See file make_query_expansion.c

  The document is sorted by highly weighted terms (*qsort(doc_cod,dcod_num, sizeof(COD_DATA), maxw_minw);*) and the selection mode (percentage or given number of terms) is given by:

  dcod_num = ((dcod_num <= 10) ? dcod_num : 10);
  (selects only the 10 highest weighted terms)
  *dcod_num =((1*dcod_num)/2);*
  (selects a given percentage of the added terms)

  dcod_num=aux ; (*aux* contains the size of the original document) and
  sort(doc_cod,dcod_num,sizeof(COD_DATA),maxp_minp);
  restitute the document in his original form (real size and sorted by padding number).

  maxw_minw(a, b) and *maxp_minp(a, b)* are functions used to sort the vector by highest weighted terms and by padding number respectively.

- **Fixed vector length:** See files make_query_size.c and read_code_file_size.c

  - The threshold QUERY size value is implemented in *make_query_size.c*:

    After all modifications are performed, the query vector is first sorted by highly weighted terms (*qsort(query_cod, qcod_num + new_term, sizeof(COD_DATA),maxw_minw)*) and then a threeshold value is selected by:

    qcod_num_final=80;
    qfd->coding.word_num =
    ((qcod_num+new_term<=qcod_num_final) ? qcod_num+new_term:qcod_num_final);

    The query vector is then sorted again by padding number, using this new value of the vector size (qsort(query_cod, qfd->coding.word_num, sizeof(COD_DATA),maxp_minp))

  - The threshold DOCUMENT size value is implemented in *read_code_file_size.c* and using the same method:

    if (type==Documents) [for documents only]

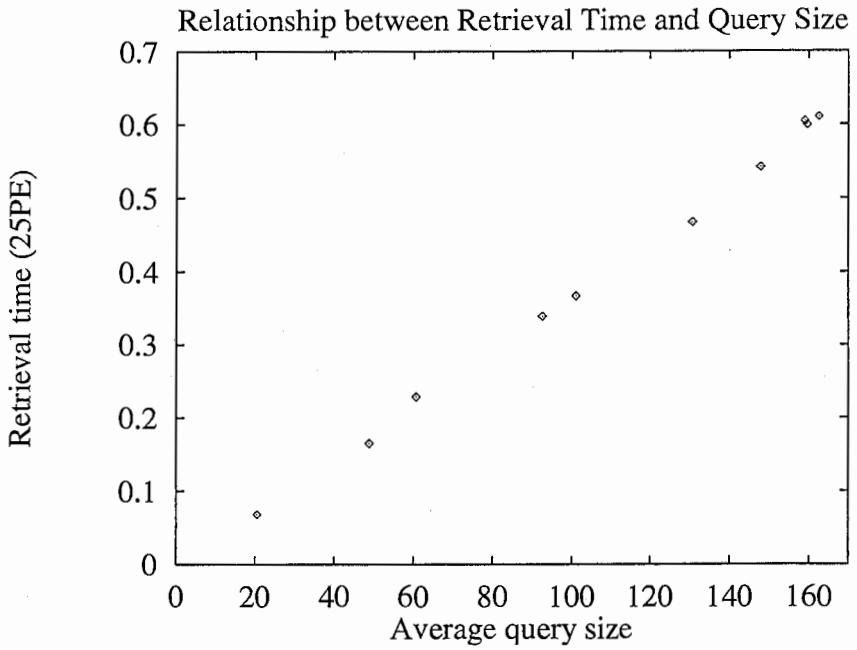
    {
    qsort(fd->coding.data,fd->coding.word_num,sizeof(COD_DATA),maxw_minw);
    (sorts by weight)

    top=80; [threshold value]
    fd->coding.word_num= ((fd->coding.word_num<= top) ? fd->coding.word_num : top);

    qsort (fd->coding.data, fd->coding.word_num, sizeof(COD_DATA), maxp_minp); [sorts by padding number with new size]
    }

D Relationship between Query Size and Retrieval Time

As a matter of fact there is an roughly linear relation between the retrieval time and the average size of the query. If a vector space method performs better than another one, the query size will be larger because more relevant documents will be retrieved and used for query reformulation purposes. As a result the retrieval time will increase to some extent "naturally". Presented below is the graph giving the retrieval time as a function of the query size, covering various experiments used in this study:



E Graphs

Comparison between Recall-Precision graphs for retrievals without relevance feedback, with relevance feedback using the original Ide dec-hi formula and with relevance feedback using the modified Ide dec-hi formula:

Fig 1: LISA collection

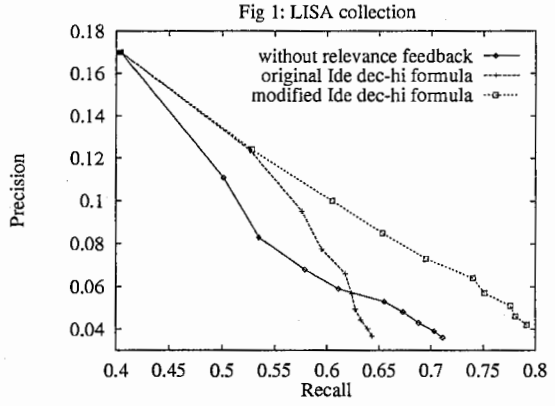


Fig 2: MED collection

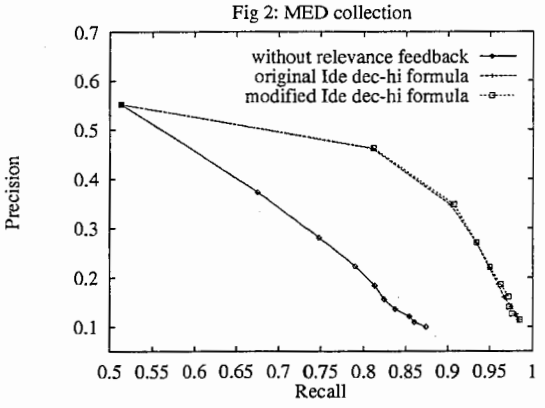


Fig 3: CACM collection

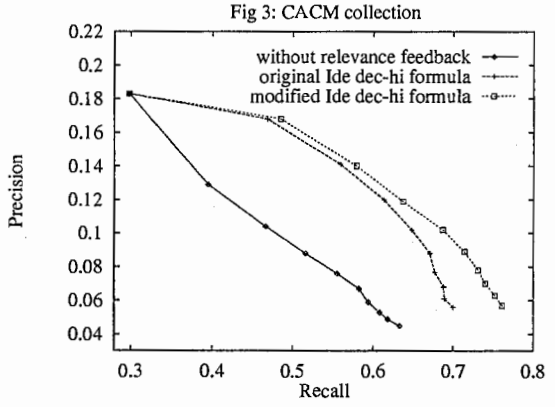


Fig 4: CRAN collection

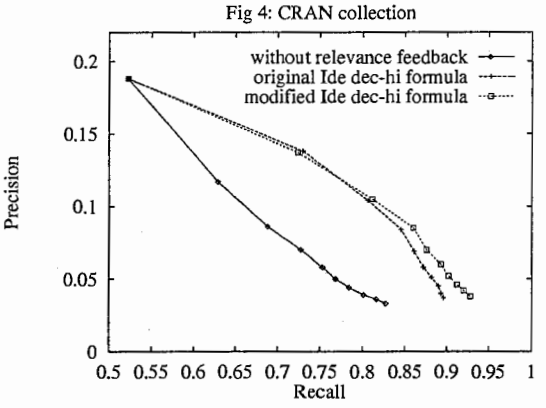


Fig 5: CISI collection

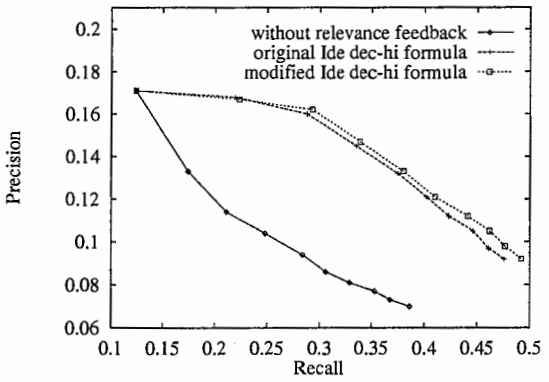


Fig 6: COMMON TERM SYSTEM with the ORIGINAL Ide dec-hi formula * LISA

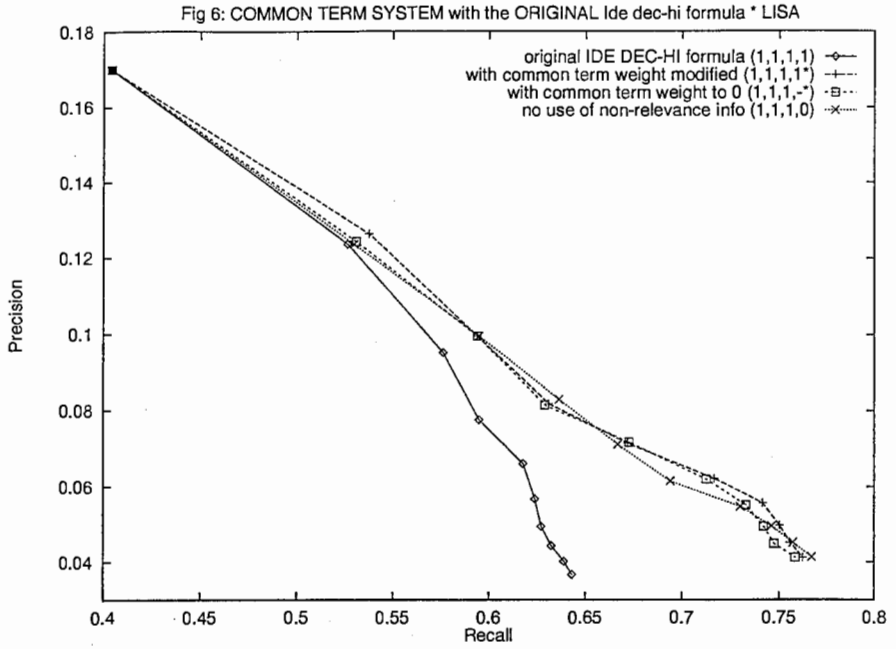


Fig 7: COMMON TERM SYSTEM with the MODIFIED Ide dec-hi formula * LISA

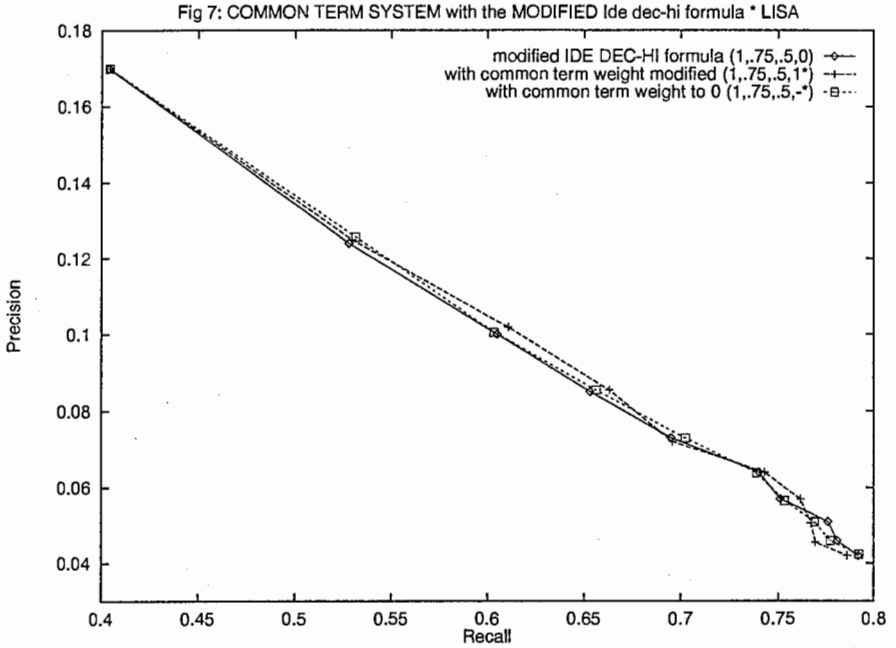


Fig 8: FIXED EXPANSION SIZE with the MODIFIED Ide dec-hi formula * LISA

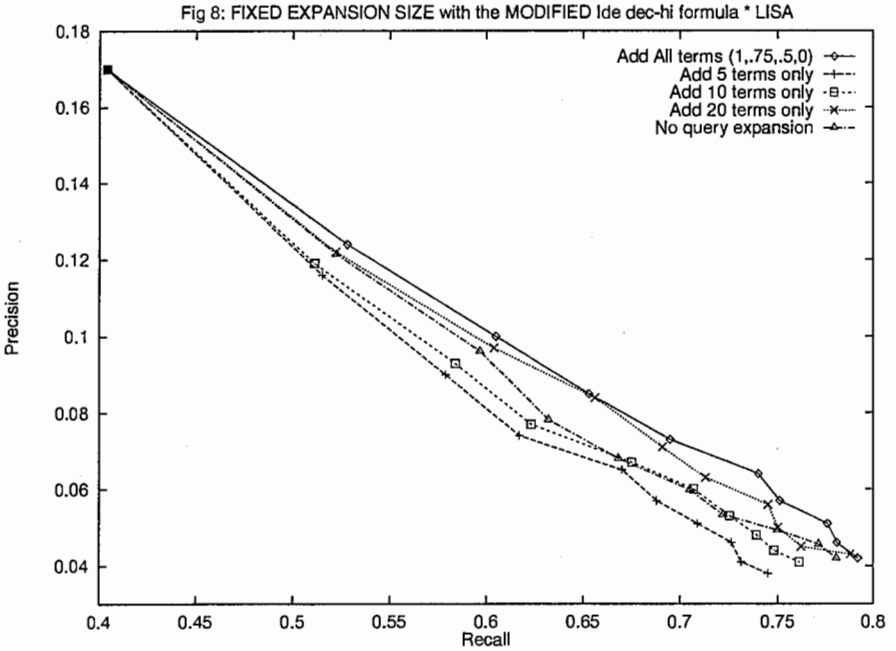


Fig 9: FIXED EXPANSION SIZE with the MODIFIED Ide dec-hi formula * LISA

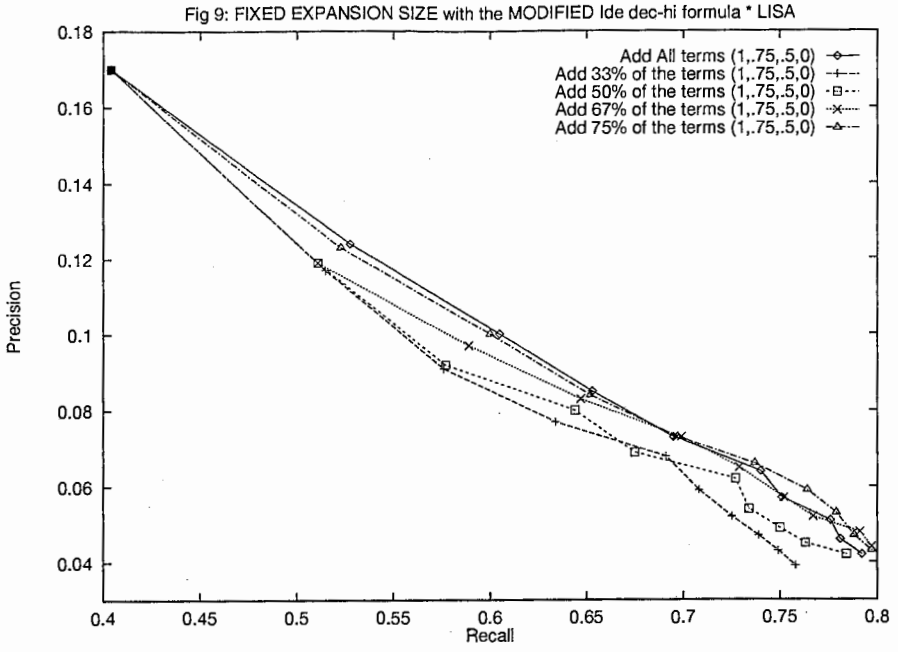


Fig 10: FIXED EXPANSION SIZE with the ORIGINAL Ide dec-hi formula * LISA

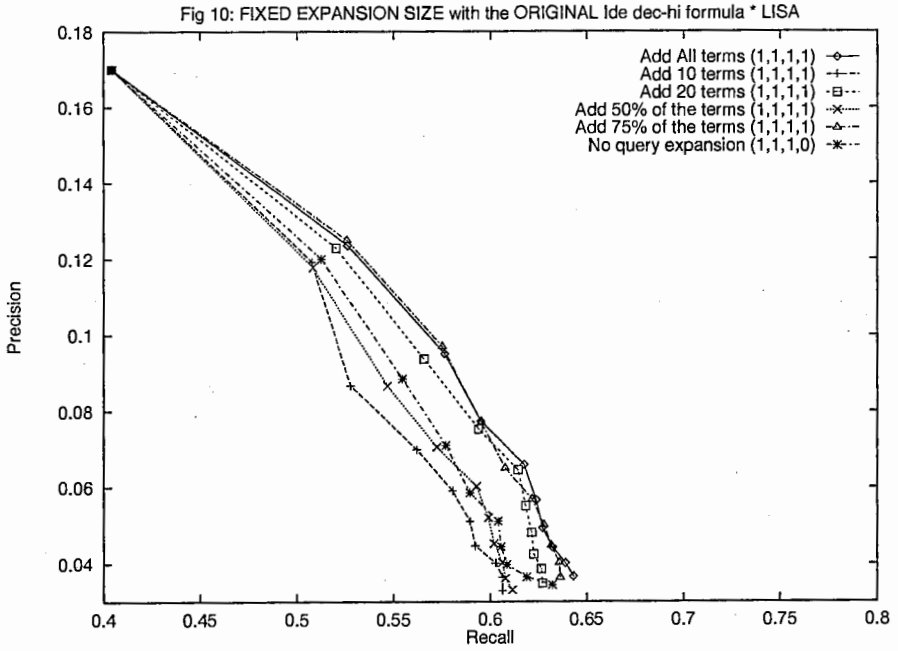


Fig 11: FIXED VECTOR LENGTH * LISA

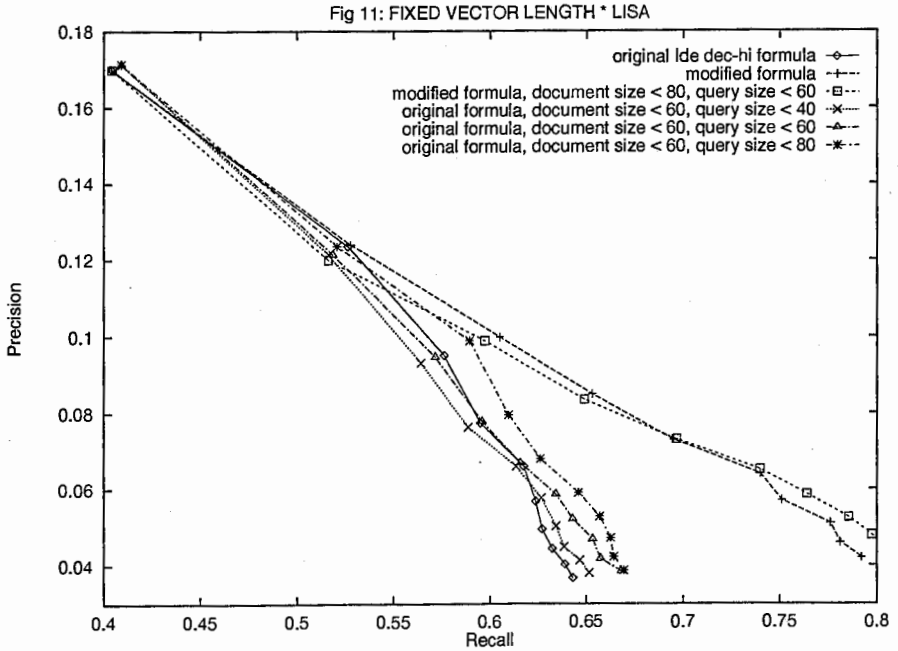


Fig 12: COMMON TERM SYSTEM with the ORIGINAL Ide dec-hi formula * MED

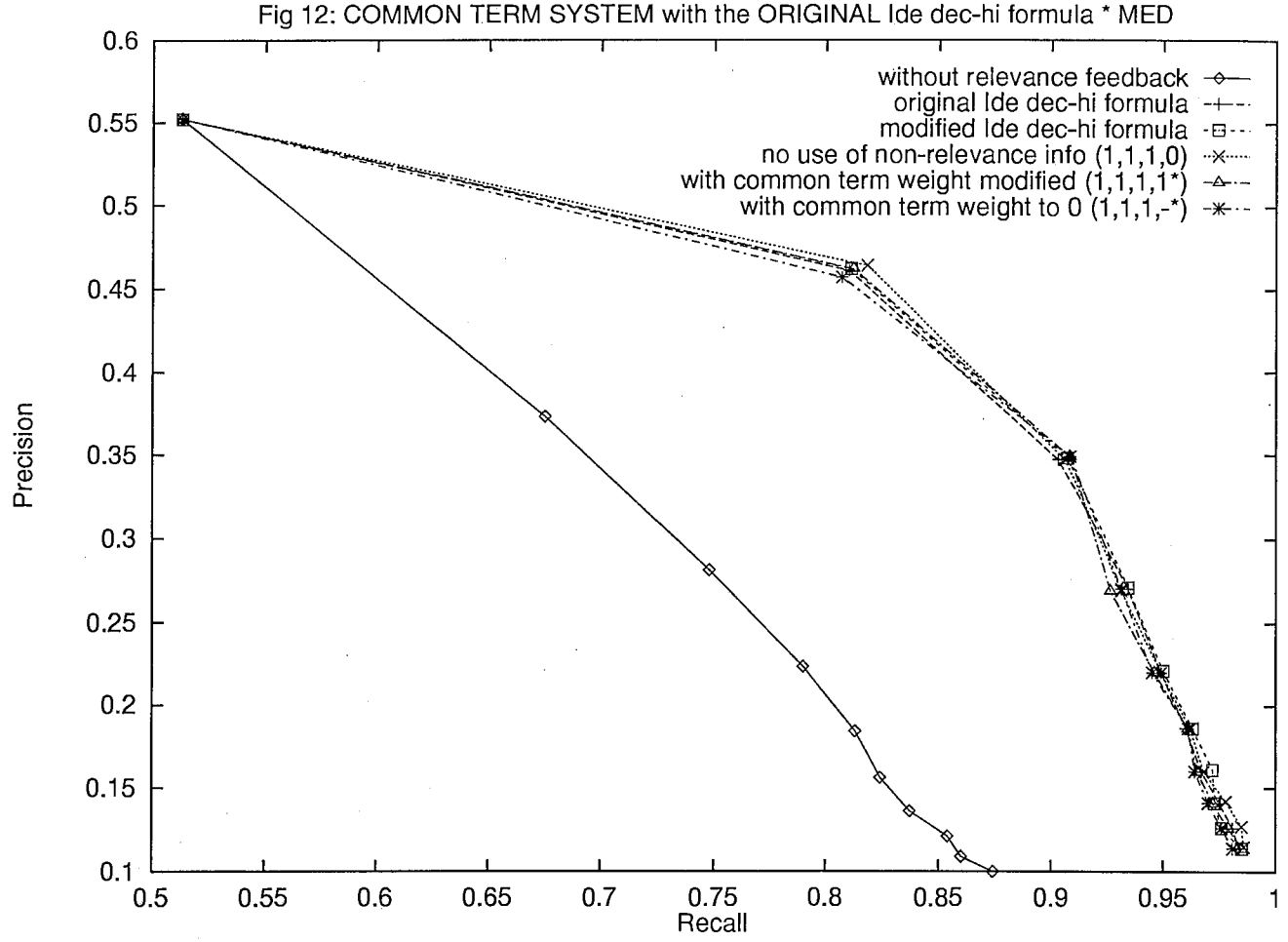


Fig 13: FIXED EXPANSION SIZE with the original Ide dec-hi formula * MED

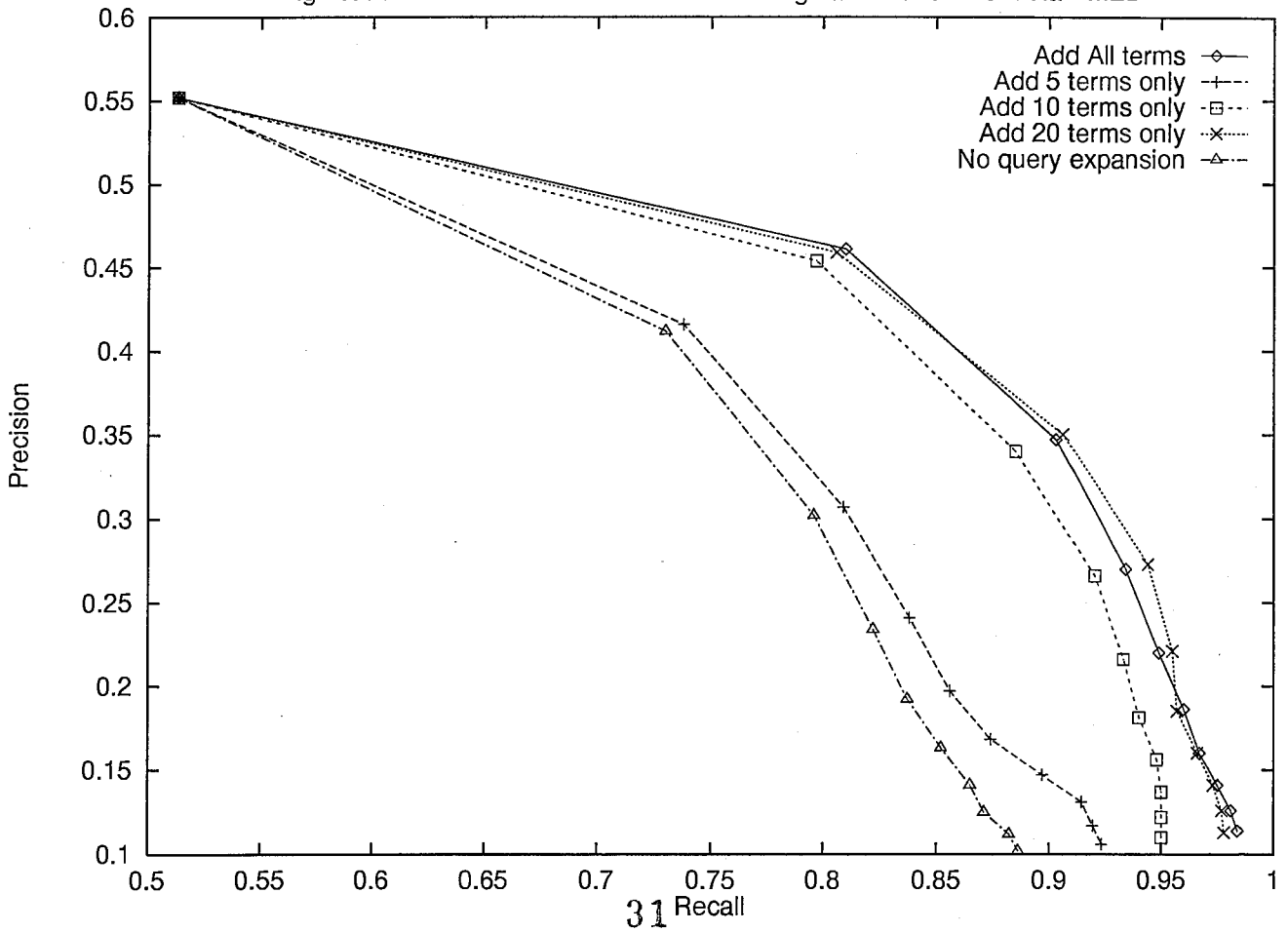


Fig 14: FIXED EXPANSION SIZE with the original Ide dec-hi formula * MED

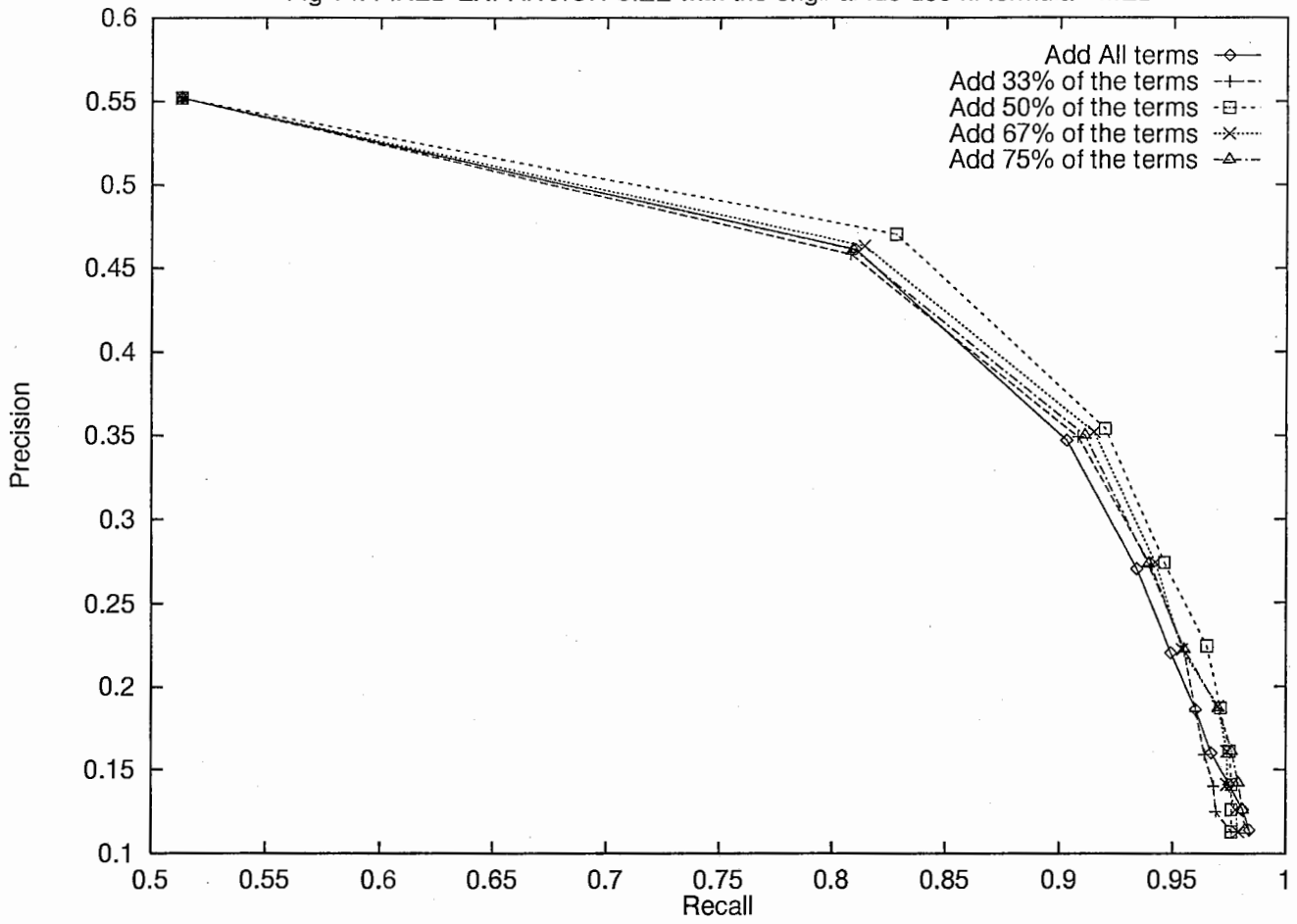
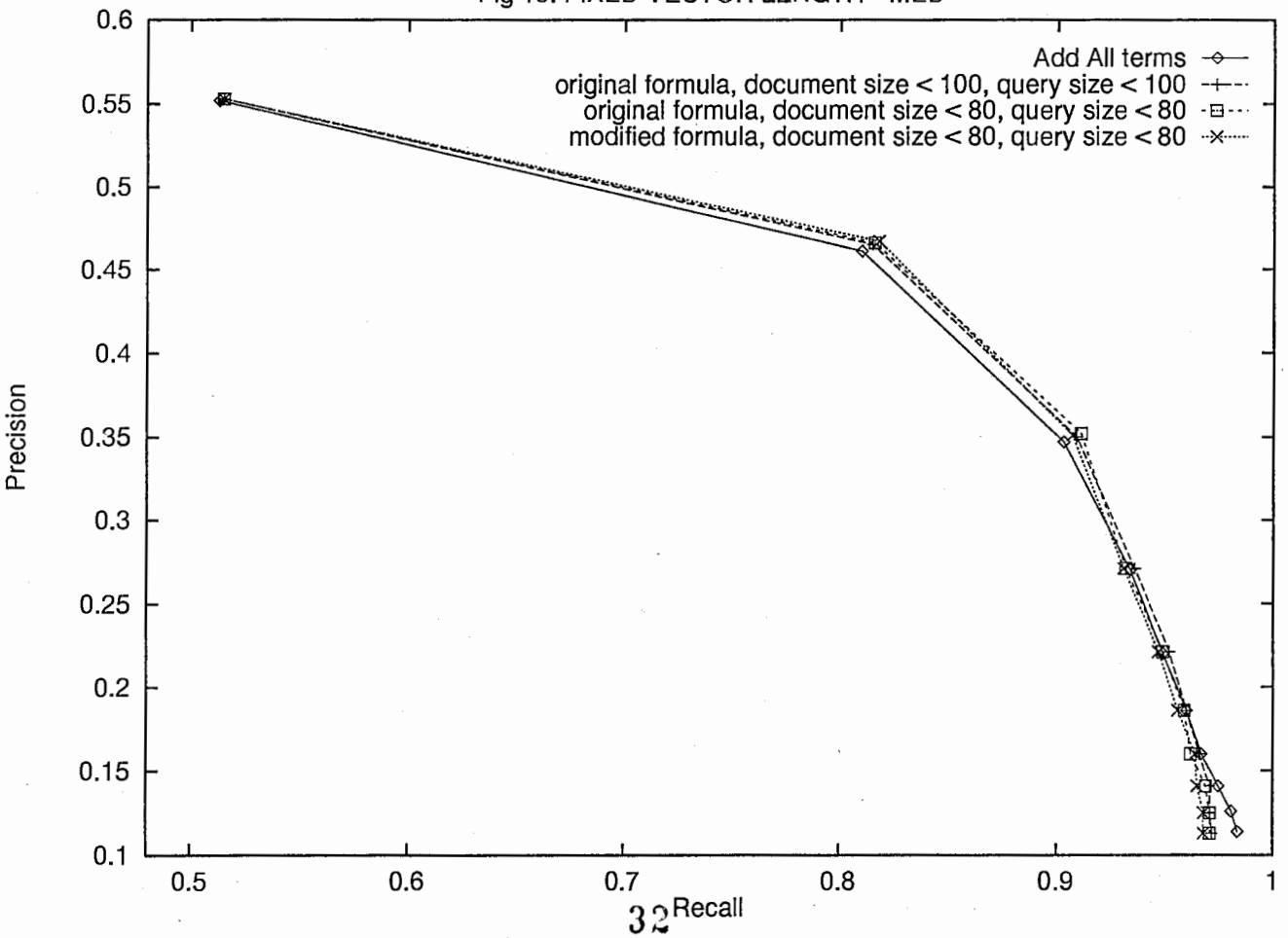


Fig 15: FIXED VECTOR LENGTH * MED



References

[1] Frakes, W. and Baeza-Yates, R., *Information Retrieval Data Structures & Algoirthms,* Prentice Hall, New Jersey, 1992.

[2] Fracis, W. and Kucera, H., *Frequency Analysis of English Usage,* Houghton Mifflin, New York, 1982.

[3] Porter, C., "An Algorithm for Suffix Stripping," in *Program,* 24(3), pp. 56-61, 1980.

[4] Salton, G. and Buckley, C., "Term weighting approaches in automatic text retrieval," in *Information Processing and Management,* 24, pp. 513-523, 1988.

[5] Ide, E., "New Experiments in Relevance Feedback," in *The SMART Retrieval System,* ed. Salton, G., pp. 337-354, Prentice Hall, New Jersey, 1971.

[6] Salton, G. and Buckley, C., "Improving Retrieval performance by Relevance Feedback," in *Journal of the American Society for Information Science,* 24, pp. 288-297, 1990.

[7] Fox, E., ed., Virginia Disk One, Virginia Polytechnic and State University, Blacksburg, 1990.