Experiences with IR TOP N Optimization in a Main Memory DBMS: Applying 'the Database Approach' in New Domains

Henk Ernst Blok[1], Arjen P. de Vries[2], Henk M. Blanken[1], and Peter M.G. Apers[1]

[1] Computer Science Department, University of Twente,
PO BOX 217, NL 7500 AE, Enschede, The Netherlands
tel. +31 53 489 3690, fax. +31 53 489 2927
{blok, blanken, apers}@cs.utwente.nl
[2] CWI, Amsterdam, The Netherlands
arjen@acm.org

Abstract. Data abstraction and query processing techniques are usually studied in the domain of administrative applications. We present a case-study in the non-standard domain of (multimedia) information retrieval, mainly intended as a feasibility study in favor of the 'database approach' to data management.

Top-N queries form a natural query class when dealing with content retrieval. In the IR field, a lot of research has been done on processing top-N queries efficiently. Unfortunately, these results cannot directly be ported to the database environment, because their tuple-oriented nature would seriously limit the freedom of the query optimizer to select appropriate query plans.

By horizontally fragmenting our database containing document statistics, we are able to combine some of the best of the IR and database optimization principles, providing good retrieval quality as well as database 'goodies' like flexibility, scalability, efficiency, and generality. Key issues we address in this paper concern the effects of our fragmentation approach on speed and quality of the answers, opportunities for scalability, supported by experimental results.

Keywords: top-N, indexing, query optimization, content based retrieval, multimedia, databases

1 Introduction

Data abstraction is the essence of the 'database approach' to data management. Specifying the manipulation and definition of data at a high level of abstraction provides not only data independence, but also enables a database management system to facilitate a wide variety of additional 'goodies', including *efficiency*, *scalability*, and *flexibility*.

The importance of services such as *transaction management* and *concurrency control* in administrative applications – and their excellent support in commercial relational database management systems – has resulted in the database approach being the norm in the domain of business applications. Unfortunately, the benefits of the database approach to data management are not so well established in most other application domains.

This paper demonstrates how two key elements of the database approach (i.e. data abstraction and query optimization) can play a similarly important role in the domain of information retrieval (IR). In general, IR systems are not very flexible, in the sense that e.g. changing the retrieval model (ranking algorithm) is far from trivial. Also, the physical access and storage structures, although often quite sophisticated, are 'hard-coded' into the system. As a result, it is not so easy to turn an existing stand-alone IR system into a parallel and/or distributed system. Neither is generality a common property among IR systems; most systems support document retrieval by content only, and not by other attributes such as author, category, or publication date. In any case, it will be very difficult to add extra attributes in a later stage.

In contrast to the conventional approaches in the IR field, we do not tie our IR retrieval model onto a physical data structure like inverted files, but specify the model declaratively at a high level (allowing flexibility in the choice for retrieval model) as described in [15,11]. The particular advantage of this approach is that it allows us to extend our research DBMS with IR techniques, *without breaking the set-oriented nature of query processing.* Such a combination of IR techniques with traditional database technology is an important ingredient for the development of search engines for large collections of XML documents that allow queries on the combination of structural properties and field values (traditional *data retrieval*) as well as their content (requiring the *information retrieval* techniques). Also, as motivated in [9], and demonstrated in [10] at VLDB'99, the same techniques provide a strong foundation for the implementation of multimedia retrieval systems.

The main objective of our paper is to (1) present a case-study demonstrating how data abstraction is equally useful in non-traditional application domains as in the administrative domain, and (2) motivate why a novel DBMS architecture is required to facilitate such broadening of core database technology for new domains. We start with a state-of-the-art IR retrieval model, that performs very well on retrieval evaluation experiments (see [16]). We then present the integration of these algorithms in our research DBMS, which resulted in a system sufficiently powerful to participate in TREC[1] [19]. This paper discusses the development of new query processing techniques at the logical level of the DBMS, improving its efficiency and scalability on IR query loads. These techniques are applied transparently in the mapping from abstract specification to implementa-

[1] The TREC, Text Retrieval and Evaluation Conference, is a well known IR conference, organized annually by NIST in the US (http://trec.nist.gov/). A key part of the conference submissions involve the benchmarking results of retrieval systems. To support this, standardized sets of documents and queries are provided by the conference organization.

tion, thus retaining the flexibility to combine queries on content with the usual data retrieval, and/or experiment with novel IR models at a declarative level.

The remainder of the paper is organized as follows. First, we outline the intuition underlying our optimization strategy in Section 2, and introduce the MirrorDBMS prototype in Section 3. Next, we describe the proposed query processing techniques in Section 4. Section 5 outlines the experimental setup to evaluate these techniques, and Section 6 presents and analyzes the results of our experimental evaluation. Section 7 presents the conclusions and future work.

2 Problem Statement

In IR systems, users express their information needs using a small number of keywords and relevance judgments on previously retrieved documents (called **relevance feedback**). Similarly, querying multimedia objects requires the user to specify characteristics of the content, e.g. by describing a color histogram of desired images. However, for sake of clarity, we limit the scope of our discussion to the ranked retrieval of text documents. Using the query and relevance feedback as an approximation of the real information need, the system then selects objects with characteristics 'similar' to those specified in the query. Notice that this comparison between objects is usually based on metadata extracted from the original documents, such as words occurring in the texts.

In the straightforward implementation of this process, each interaction step between user and system involves ranking all objects based on their similarity to the query, although only the N 'most' similar objects are presented to the user (the top-N objects). Obviously, top-N query optimization (attempting to compute only the similarity for the top-N documents to be presented) is a natural step to improve the efficiency of (multimedia) information retrieval.

As mentioned briefly, content querying is interactive and iterative: after reviewing, a user gives an indication of the quality of the answer (relevance feedback) which is used to modify the original query. Processing the modified query generates new answers, and so on. Thus, it seems particularly interesting to cut off query processing at a *reasonable* stage, and show the results computed till then to the user for relevance feedback. Although the quality of the answer may be impeded, this may allow for great reductions in computations, possibly without diminishing the effectiveness of the relevance feedback process. Our intuition is that incomplete answers can still provide a good basis for relevance feedback: a quick approximate response may still provide sufficient information to refine the estimate of the user's information need, and consequently improve the effectiveness of retrieval. For each iteration, the users may decide for themselves whether they prefer quicker responses of (generally) lower quality, or slower responses of (hopefully) higher quality.

In handling top-N queries more efficiently, implementations of IR systems have drawn on a combination of domain knowledge – exploiting the Zipfian [20] distribution of terms found in documents (see e.g. [17]) – with smart element-at-a-time cut-off operations derived from the ranking function, like in the implementation of the well-known INQUERY retrieval system [7,6]. Exploiting the element-at-a-time manner of processing, highly accurate cut-off conditions can be updated after evaluating each element, allowing for efficient reduction of obsolete intermediate results being computed. This algorithm and comparable ones (e.g. [8]) exploit a carefully designed ordering of the data, mathematically well-founded by the work of Fagin [14,12,13].

Obviously, the element-at-a-time nature of these native IR algorithms for top-N query processing reduces significantly the number of possible query plans under consideration in the query processor of a DBMS combining IR techniques with data retrieval; which is unacceptable in many cases. So, the goal in this paper is to devise a database approach that is able to satisfy the following goals:

- Improving the efficiency and scalability using (a) domain knowledge and (b) new techniques inspired by the (element-at-a-time) cut-off operations, and,
- Maintaining the flexibility and generality of the database approach to IR.

Summarizing, we have identified two potential approaches to improve efficiency in information retrieval query processing in a database environment: on the one hand, we may reduce the amount of work by ranking fewer documents, and on the other hand, we may take advantage of computing only partial answers in the first iterations of the retrieval process. In the remainder of this paper, we will demonstrate how the database approach allows us to exploit **fragmentation** of the metadata to achieve these ideas in a simple yet effective manner, requiring only minimal changes to the original, declarative specification of the IR retrieval model.

3 IR Query Processing in the MirrorDBMS

The architecture of the MirrorDBMS, our research prototype, consists of two layers: the logical layer, based on Moa object algebra [4], and a physical layer, realized by the binary relational main-memory DBMS Monet [2,3]. The query processor transforms the algebraic query expressions specified in Moa into the – highly efficient – physical operators offered by Monet. The distinguishing feature of the MirrorDBMS is that it is extensible at all levels of its architecture: enabling the encoding of domain knowledge and advanced query processing techniques at the logical level as well as the physical level (please refer to [9, Chapter 2] for more information). Also, the prototype DBMS is well prepared for scalability, as Moa supports shared-nothing parallelism, and shared-memory parallel computing is supported by Monet at the physical level.

3.1 The IR Retrieval Model

A retrieval model specifies how the similarity between a document and the query is computed, given the query and the relevance feedback from one or more previous iterations. Most probabilistic IR models rank the documents based on two parameters:

term frequency: For each pair of term and document, *tf* is the number of times the term occurs in the document.

TF(term, document, tf)	(1)
IDF(term, idf)	(2)
Q(term)	(3)

Fig. 1. Relations

inverse document frequency: For each term, *idf* is the inverse number of documents in which the term occurs.

In a more database like notation we can describe these two statistics as the relations 1 and 2, respectively, as shown in Figure 1.

Furthermore, we introduce the set of query terms, i.e. the unary relation 3 shown in Figure 1.

In most probabilistic retrieval models, the ranking of a document given a query is almost completely determined by the sum of the product between the *tf* and *idf* of the query terms occurring in the document, sometimes normalized with the document length in one way or another. See [19] for more detailed information about our specific ranking formula and retrieval model.

3.2 Query Processing in an IR System

The algorithm shown in Figure 2 in pseudo-code, consisting of three parts, sketches how a typical IR system computes its query results from these tables in a nested-loop manner, thus determining precisely the physical execution order. We will assume that IDF is ordered descending on *idf*.

It may be clear that this algorithm allows a very efficient cut-off, but also is highly inflexible with respect to execution order.

3.3 Set-Oriented IR Query Processing

To reduce this inflexibility of the IR retrieval process, and thus enable smooth integration with traditional DBMS query processing, we reformulate this algorithm at a higher, declarative level.[2]

In the implementation, the information retrieval techniques are supported by extensions at both levels of the MirrorDBMS. While the exact ranking formula requires some minimal extensions at the physical level, the set-oriented formulation of IR query processing is almost completely modeled at the logical level as a Moa extension. Although the real algorithm is specified using the powerful but relatively low-level Monet Interface Language (MIL), we prefer an SQL-flavored syntax for didactic reasons.

Again, the algorithm, as shown in Figure 3, consists of three parts.

[2] For the impatient: the usefulness of this seemingly minor step will become more clear when we present our optimization techniques based on data fragmentation in Section 4.

Part A Limit the TF and IDF to match the terms in query Q:

```
foreach t1 in TF do
  if (t1.term in Q) then
     INSERT t1 INTO TFQ
  endif
end

foreach t1 in IDF do
  if (t1.term in Q) then
     INSERT t1 INTO IDFQ
  endif
end
```

where t1 denotes the tuple-variable associated to the relations.

Part B Loop over the terms to compute the ranking contribution per document-term pair, and update the document ranking incrementally each time a new ranking contribution for that document becomes available. Stop as soon as a test (based on the processed IDFQ and TFQ values, knowing that IDFQ can only decrease, cf. [13]) shows that no document could obtain a ranking better than the current top-N.
Like before, t1 and t2 are tuple-variables. Furthermore we assume the existence of a table RANK that has two columns: document and rank.

```
foreach t1 in IDFQ do
  # Find the matching tf values
  TFQsel = findrecords(TFQ, t1.term)

  foreach t2 in TFQsel do
    tfidf = t1.idf * t2.tf;
    if (t2.document in RANK) then
       updateranking(RANK,
         t2.document, tfidf)
    else
       addranking(RANK,
         t2.document, tfidf)
    endif
    # topN test criterium
    if (!topNcanimprove) then
       exitloops
    endif
  end
end
```

Part C Return the top ranking documents:

```
i = 0
foreach t1 in RANK
do
   if (i < N)
   then
      INSERT t1 INTO TOPRANK
   else
      exitloops
   endif
   i = i + 1
end
```

Fig. 2. IR query evaluation algorithm

Part A Do some initialization given query Q.
Part B Limit the TF and IDF to match the terms in query Q:

$$\mathrm{TF_Q} = \mathrm{TF} \bowtie \mathrm{Q}$$

and

$$\mathrm{IDF_Q} = \mathrm{IDF} \bowtie \mathrm{Q}.$$

Next, place the $\mathrm{IDF_Q}$ values next to the corresponding entries in $\mathrm{TF_Q}$:

$$\mathrm{TFIDF_{lineup}} = \mathrm{TF_Q} \bowtie \mathrm{IDF_Q}.$$

Now, compute the *tf* · *idf* value per term-document pair, aggregating the last two columns into one:

```
TFIDF = SELECT term, document, tf * idf
        FROM TFIDF_lineup.
```

Finally, compute the ranking per document by aggregating all term contributions per document:

```
RANK = SELECT document, AGGR(tfidf)
       FROM TFIDF
       GROUP BY document.
```

Please note that in the actual code, the AGGR()-operator does not exist as one operator, but denotes a combination of several functions that together compute the ranking. We abbreviated it here for reasons of simplicity.
Part C Normalize RANK and select the top-N documents:

$$\mathrm{TOPRANK} = \mathrm{TOP(RANK,\ N)}.$$

Fig. 3. Set-oriented IR query processing

3.4 Discussion

The main performance bottle-neck lies in handling the TF table[3]: TF contains over 26 million entries in the experiments performed for this paper - which is only about a quarter of the complete TREC data set. Only **Part B**[4] in the algorithm described above handles a very large amount of data. At a first glance, the pre-selection on query terms, $\mathrm{TF_Q} = \mathrm{TF} \ltimes \mathrm{Q}$, at the beginning of **Part B**, may potentially reduce the remaining dataset to a manageable size. But, it is a well-known fact in IR experiments that for the average query, roughly about half of the database remains after that pre-selection: still a very large dataset as input for the subsequent computations.

Since, $N = 1000$ or (often) less, pushing the top-N operator into the query could be very profitable. However, pushing it down the query plan implies pushing it through the AGGR()-operator, and therefore through the *tf* · *idf* product.

[3] Since we work on a binary model, several tables represent together the columns of TF. However, for didactic reasons we will stick to the normal table approach since this has no significant consequences for the core of our idea.

[4] From now on, when we refer to **Part A**, **Part B**, or **Part C**, we mean the ones described in the database approach and not in the IR approach as described in Subsection 3.3.

A generic (set-oriented) mathematical solution for this top-N query optimization problem is not a trivial one, despite its innocent look. In the next Section, we therefore propose data fragmentation as another means to prune the search, while keeping (the declarative specification of) the algorithm practically untouched.

4 Data Fragmentation and the Top-N Query Optimization Problem

Since Monet is a main-memory DBMS, the data used in the hot set should always fit in main-memory (to avoid performance degradation due to swapping, or even worse, a crash caused by running out of memory). The natural way to meet this requirement is to horizontally fragment the TF table into a small (yet to be determined) number of suitably sized parts. Such a fragmentation strategy is orthogonal to the actual retrieval algorithm, and can be managed in the mapping from Moa to MIL.

In this paper, we elaborate on the use of additional knowledge for choosing the fragmentation scheme, *specific for query processing in the IR domain.* We show how this enables us to achieve both proposed strategies to improve the efficiency of IR query processing: (1) computing partial answers, and (2) top-N query optimization. Furthermore, the implementation of the fragmentation strategy remains almost entirely orthogonal to the IR retrieval algorithm outlined before. Remark that we will focus on optimization techniques at the logical level of the MirrorDBMS.

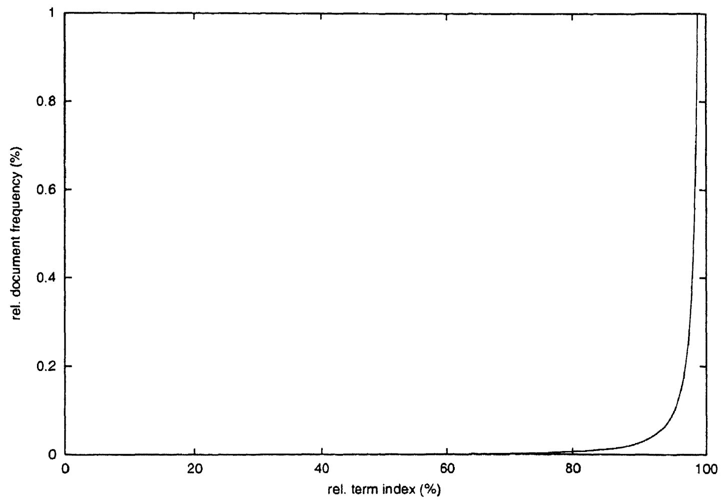

Fig. 4. Relative document frequency (zoomed on y-axis to show lower values)

Restricting query processing to a smaller portion of the metadata is a well-known approach to increase the efficiency of IR system implementations by computing approximate answers. Obviously, this implies that the effectiveness of the answer (measured using precision/recall) will degrade: we trade quality for speed. To minimize the loss on quality, we exploit the properties of the afore-mentioned Zipfian term distribution. The hyperbolic curvature of the document frequency plot, shown in Figure 4, confirms that the data in our test database (see also Section 5.1) indeed behaves as predicted by Zipf, validating the underlying reasoning behind our approach.

4.1 The Fragmentation Algorithm

Now, Figure 5 shows the simplified basics of our fragmentation algorithm for splitting the data up in two fragments using the additional information about the term distribution. To keep the example simple, we take the first fragment such that it contains $s_1 \cdot |\mathrm{IDF}|$ of the terms and the second one the other $s_2 \cdot |\mathrm{IDF}|$ terms, where $s_2 = (1 - s_1)$ and $0 < s_i < 1$, $i \in \{1, 2\}$.

Step 1 Sort the IDF descending on the *idf* values, i.e., terms that occur in many documents get lower in the list compared to terms that occur in less documents.

```
IDF_sorted = SELECT *
             FROM IDF
             ORDER BY idf DESC
```

Step 2 Create two fragments IDF_1 and IDF_2 such that

```
SELECT COUNT(*)
FROM IDF_i
=
SELECT s_i · COUNT(*)
FROM IDF_sorted
```

for $i \in \{1, 2\}$.

Step 3 Create two fragments TF_1 and TF_2 such that

$$\mathrm{TF}_i = \mathrm{TF} \ltimes \mathrm{IDF}_i$$

for $i \in \{1, 2\}$.

Notice that, for $s_1 = 0.95$, TF_1 would now contain approximately 5% (not 95%!) of the tuples of TF, and TF_2 the rest, due to the high skewedness of the data.

Fig. 5. Fragmentation algorithm

Since the terms in IDF_1 have a high *idf* their contribution to the ranking of a document is likely to be higher than for terms in IDF_2 (having lower *idf* values). In other words, the terms in IDF_1 are *a priori* more promising than the terms in IDF_2. Fortunately, these *interesting* terms only use about 5% of the data (in case $s_1 = 0.95$). So, in case all query terms are stored in the first fragment, we only need to compute the results using IDF_1 and TF_1. This would mean that the following semijoin $\mathrm{TF}_Q = \mathrm{TF} \ltimes Q$ in **Part B** of the algorithm would become $\mathrm{TF_Q} = \mathrm{TF_1} \bowtie \mathrm{Q}$, which will be significantly faster due to the much smaller first operand.

In case not all query terms are contained in the first fragment, one might decide to still compute the results on the first fragment only. This could of course result in a different top when too much significant information is ignored that way. Some experiments described later in this paper try to determine the effects of ignoring the second fragment on the quality of the answer.

Finally, notice that the fragmentation algorithm described above can easily be used to handle different fragmentations, for instance for different relative sizes (just choose a different s_i) and/or more than two fragments (have $i \in S$, with $S \subset \{1, 2, 3, \ldots, M\}$, $|S| > 2$, $M = |\mathrm{IDF}|$). For reasons of simplicity, it is sometimes more practical to join TF and IDF before fragmenting the data, and propagate the fragmentation into TF and IDF fragments subsequently. This other method is particularly handy to obtain fragments of (almost) equal data size.[5]

4.2 Fragment-Based IR Query Processing with Top-N Cut-off

The algorithm in Figure 6 shows the top-N cut-off idea in a similar manner like the set-based description of the retrieval algorithm as described in Subsection 3.3 exploiting the fragmentation idea described above.

This algorithm in fact is a sub-set-at-time version of the element-at-a-time version described in Subsection 3.2.

Unsafe Top-N Optimization. Note that this algorithm is a so called *unsafe* top-N cut-off algorithm [6]. Top-N query optimization relies on the cut-off of the query evaluation at a certain stage when certain characteristics concerning the still remaining work[6] provide sufficient evidence that the top-N cannot be improved anymore. However, this also means that at the cut-off moment certain information, e.g. ranking contributions, have not been taken into account. This usually results in a top-N containing the correct documents but with an incomplete ranking. In turn, this can result in a different ordering of the top-N. Unsafe top-N query optimization stops at this 'incorrectly' ordered top-N.

Safe Top-N Optimization. The *safe* alternative to the unsafe method does indeed return the top-N with the correct ranking values and inherently can deliver them in the correct order. To obtain these final ranking values the ranking contribution for the documents in the unsafe top-N needs to be computed for all fragments that have not been taken into account, yet. This of course will (slightly) reduce the profit of top-N cut-off due to the extra work that has to be done.

[5] The fragmentation process itself is part of the physical design of the database, and therefore its performance is not really an issue, at least for mostly static collections.

[6] In the algorithm described here the topNcanimprove variable represents the information needed to make the cut-off decision.

Part A Similar to **Part A** in Subsection 3.3. Set i to the first fragment that contains a query term.
Part B Similar to **Part B** in Subsection 3.3, but this time using fragment i instead of the unfragmented TF and IDF. Let's call the resulting ranking RANK_i.
Part B′ Merge RANK_i into any existing RANK or otherwise set RANK = RANK_i.
Part C Normalize RANK and select the top-N documents:

$$\text{TOPRANK} = \text{TOP(RANK, N)}.$$

Part C′ Compute the lowest ranking in the current intermediate top-N:

$$\text{topLB} = \text{MIN(TOPRANK)}$$

and the highest ranking in the remaining intermediate results:

$$\text{restUB} = \text{MAX(RANK} - \text{TOPRANK)}.$$

Furthermore, compute the highest possible ranking contributions over all fragments $j > i$:

$$\text{contribUB} = \text{MAXRCONTRIB}(\text{TFIDF}_j, \ldots, \text{TFIDF}_{25})$$

and the lowest possible ranking contribution

$$\text{contribLB} = \text{MINRCONTRIB}(\text{TFIDF}_j, \ldots, \text{TFIDF}_{25}).$$

Part C″ Test whether the top-N still can be improved:

$$\text{topNcanimprove} = (\text{restUB} + \text{contribUB} \geq \text{topLB} + \text{contribLB})$$

and limit RANK to those documents that still can move up into the top-N:

```
RANK = SELECT * FROM RANK
       WHERE rank ≥ topLB + contribLB − contribUB
```

as soon as:

$$\text{MIN(RANK)} \leq \text{topLB} + \text{contribLB} - \text{contribUB}$$

and COUNT(TOPRANK) > N and limit all fragments $j > i$ to match this new intermediate ranking.
Part C‴ If topNcanimprove is true, then find the next fragment i containing a query term and return to **Part B** (in this algorithm) . Otherwise, return TOPRANK and quit.

Fig. 6. Fragment-based IR query processing with top-N cut-off

Heuristic Unsafe Top-N Optimization. Going even further on the unsafe principle, we can drop the requirement in **Part C″** that the intermediate rank only can be restricted when

$$\text{MIN(RANK)} \leq \text{topLB} + \text{contribLB} - \text{contribUB}.$$

The algorithm then becomes even 'more' unsafe: as soon as COUNT (TOPRANK) > N, documents that have no ranking yet are ignored, even when they would have received a high ranking otherwise. In turn, this *heuristic unsafe* method is very likely to achieve a much better performance due to the earlier and more restrictive limitation imposed on RANK and all fragments $j > i$. The 'level of unsafe-ness' can be controlled by adding some documents (with initial ranking 0.0) to RANK already during **Part A** using an a priori notion of ranking between the documents. These documents cannot be forgotten anymore, but will keep their 0.0 ranking when they do not contain any query terms, thus not disrupting the ranking process in case they were wrongly added in advance.

In our case we control the level of unsafe-ness using a factor l (where $0.0 < l < 1.0$) to select the $l \times$ no. of documents with the highest document length to be added in advance. The document length appeared to be an interesting, natural measure of a priori document relevance for the IR model we used. However, one can think of many other other means to 'pre-select' documents that should not be ignored (i.e. the documents that are most referenced in a digital library case, or most linked to in the web case). Also note that a too high l will cause the performance to drop rapidly because of the then extremely high number of documents that are forced to be ranked.

5 Experimental Setup

In the experimental evaluation of the ideas put forward in the previous section, we focus on the following three concrete research questions:

1. How can fragmentation improve efficiency for top-N query execution?
   a) What are the consequences for the speed?
   b) What are the consequences for the quality of the query results, also taking into account the impact of safe/unsafe top-N optimization?
2. How can fragmentation improve scalability, to either manage the same database on smaller hardware (like a notebook), or a larger database on the same hardware (such as a search engine for the WWW).

5.1 Data Set and Evaluation Measures

The experiments are performed on the *Financial Times* (FT), a major subset of the TREC data set, using the 50 topics (i.e. queries) and relevance judgments used in TREC-6. Since we want to investigate the trade-off between quality and speed we need a good benchmark for the precision and recall. The used TREC relevance judgments are the most widely accepted retrieval quality benchmarks. Also, the FT document collection is sufficiently large to show the important effects.

We defined four series of experiments, which we evaluate using the measures described in Figure 7.

5.2 Overview and Motivation of Experiments

Here, we discuss the four series of experiments that we performed:

Series I: Baseline. The first series of experiments are meant to show the quality and performance of our system without any special tricks: the Monet DBMS determines whether to build any access structures (usually hash tables) to speed up certain operations (for instance: joins). In this version the main focus was on the quality of the retrieval results and flexibility of the retrieval model. The effort to optimize this system for performance did not exceed the typical exploitation of certain typical alignment issues important in main memory computing.

Average Precision We define the average precision (AP) as:

$$AP \equiv avg(p_i)$$

where p_i is the average 11-point-precision for each query i, $i \in \{1, 2, 3, \ldots, 50\}$. So, the AP measure is actually the average average 11-point-precision.

Average Retrieved Relevant We define the average retrieved relevant (ARR) as:

$$ARR \equiv avg(r_i)$$

where r_i is the number of relevant documents retrieved for each query i, $i \in \{1, 2, 3, \ldots, 50\}$. Notice that the well-known *recall* measure is defined as r_i divided by the total number of relevant documents for query i.

Average Execution Time The average execution time (AET) is defined as:

$$AET \equiv avg(t_i)$$

where t_i is the (wall clock) execution time measured for each query i, and $i \in \{1, 2, 3, \ldots, 50\}$.

Fig. 7. Fragmentation algorithm

Series II: Speed/Quality Trade-off. In the second series of experiments, we concentrate on the effects of ignoring data on the trade-off between quality and efficiency. We defined two variants of these series of experiments:

(a) Always use the first fragment (and forget about the second fragment).
(b) Take the first fragment, unless

$$\mathrm{NIDF_1} \ltimes \mathrm{Q} = \emptyset.$$

We used a term-fragment index to allow an efficient choice, instead of using just this semijoin.

Both types of experiments are executed for several different fragmentations, where the relative size in terms of the first fragment varies from 90% to 99.9%, using the fragmentation algorithm described above.

Obviously, the (a) series can result in loss of quality; if none of the query terms have a high *idf* value, the answer set has been reduced to a random sample from the collection. The (b) series are meant to reduce this negative effect.

We expect that the speed will decrease in favor of the quality with increasing first fragment size. The second experiment should be slower, since the evaluation of the second fragment triggered by some of the queries will increase the execution time considerably; though resulting in a better quality than for series (a).

Series III: Benefits of Fragmenting. Series II focuses on the trade-off between ignoring data to obtain speed compared to the quality of the resulting answers. However, the second fragment is still quite large in terms of data size, impeding main-memory execution. The experiments in series III are mainly intended to investigate the effects of executing our query algorithm on relatively small fragments. To do so, we fragment our database in 25 smaller fragments of equal data size. The number of fragments was experimentally determined based on two constraints: there should be sufficiently many fragments to demonstrate the expected behavior, but, each fragment should still be reasonably large as to obtain the advantage of set-oriented processing.

Again, we perform a couple of variants of these experiments:

(a) This variant studies the effects of the fragmentation procedure described in Section 4 on execution time and quality of results. As in Series IIb, we use a term-fragment index to efficiently determine whether a fragment should be evaluated or not.

(b) This variant uses the same fragmentation as for (a), but this time we allow query evaluation to be cut-off after each fragment. The choice whether to stop processing the query (and after which fragment) is based on estimates whether the top-N can still be improved by processing of any following fragments. This strategy uses the computed lower and upper bounds to restrict the intermediate ranking to those documents that may still move into the top-N, thus limiting the computational efforts needed for any successive fragments still to be evaluated. We evaluate both the safe and (normal) unsafe cut-off principle in this variant.

(c) As described in Section 4 one can relax certain conditions for the unsafe algorithm, obtaining a, what we call, heuristic unsafe method. This variant performs the heuristic version of the unsafe experiments done for the (b) variant, taking $l \in \{0.00,\ 0.05,\ 0.10,\ 0.25,\ 1.00\}$. As explained before we used l to pre-select the fraction of a priori most interesting documents[7] that should not be ignored in case of intermediate result restriction in **Part C$''$**.

Since variant IIIa takes into account all query terms, we expect the AP and ARR to be equal to the figures measured in Series I. The AET will probably be better (e.g. lower) than for Series IIb, since the overhead occurring from using an extra fragment is likely to be lower (since the fragments are smaller).

Of course, the computation of the estimates in series IIIb introduces an overhead in execution costs; this investment only pays off if the profits of the optimization are high enough. The (b) variant of these series are meant to show whether this is still the case when applied to subsets-at-a-time processing rather than the element-at-a-time case studied in [6]. Note that in [6] the results for the safe method showed no real significant performance improvement.

As mentioned before, the quality is likely to be somewhat lower in case of the unsafe variant of the cut-off.

We expect the IIIc variant to outperform IIIa and IIIb (both for safe and unsafe runs) by far for $l = 0.00$ and quality to be lower but not really bad. For growing l we expect the performance to degrade rapidly since the overhead will grow significantly. However, in the case of $l = 1.00$ the quality should reach the same levels as measured for the IIIa variant.

[7] We used the document length which appeared as a natural candidate given the IR model we used. As stated before, other measures might be more appropriate in other environments.

Series IV: Influence of Query Length and Top-N Size. Because we calibrate the quality measurements using the relevance judgments of the TREC-6 queries, the experiments in Series III have been performed with fairly long queries: an average length of 27 terms, and the longest query contains over 60 terms. Also, TREC evaluation requires the top 1000 to be produced for each query.

Series IV try to provide insight in (a) the effect of query length on the AET and (b) the effect of the size of required top-N on the AET. The (a) variant repeats the experiments of Series I (unfragmented case), IIIa (25 fragments, no top-N cut-off), IIIb (25 fragments with normal safe/unsafe top-N cut-off), and IIIc (25 fragments with heuristic unsafe top-N cut-off) for limited query lengths. The new queries are constructed by taking the first k terms of each original query (or the entire original query in case it was shorter than k terms). We let k range from 1 to 25. The (b) variant leaves the queries untouched and computes Series I, IIIa, IIIb, and IIIc for several top-N sizes ranging from 10 to the original 1000.

We expect that the (a) variant will show a relative performance advantage in favor of the top-N cut-off for longer queries compared to the cases without top-N cut-off. For shorter queries, fragmentation alone will already result in quite efficient processing whereas top-N cut-off would only cause extra computational costs without much chance to gain a profit. The (b) variant is expected to demonstrate better AET values for lower N. The shorter the required top-N, the higher the lowest ranking value topLB occurring in the top-N; and, the higher topLB, the higher the value of topLB + contribLB − contribUB used to restrict the intermediate result RANK. This means that more elements in RANK are likely to be cut away. Also, the higher ranking values usually tend to be further away from their closest neighbors. So, the higher the topLB, the higher the chances that restUB will be so much further away (i.e. lower) that the gap cannot be bridged anymore: thus allowing for a top-N cut-off. In both cases the execution time is reduced, either due to less computational load per fragment or fewer fragments being evaluated.

6 Experimental Results

The hardware platform used to produce the results presented in this section is a dedicated PC running Linux 2.2.14, with two Pentium^{TM} III 600 MHz CPUs (of which only one was actually used in the experiments), 1 GB of main-memory, and a 100 GB disk array mounted in RAID 0 (striping) mode; no other user processes were allowed on the system while running the experiments.

The remainder of this section is divided in four parts, corresponding with the four series of experiments. For each of these series of experiments, we included some figures/tables to illustrate the results.

6.1 Series I: Baseline

In this subsection we present the AP, ARR, and AET for the baseline run of the retrieval experiments. Table 1 shows the measured values, next to the values provided by TREC as the benchmark. The AET of course is not available for the benchmark.

Table 1. [Series I] Baseline result statistics (TREC benchmark included for comparison)

	AP (%)	ARR	AET (s)
Benchmark	100.0	31.8	-
Series I	31.0	22.9	44.4

The ARR of 22.9 means that the recall of our unfragmented is

$$\frac{ARR_{unfragmented}}{avg.\ actual\ no.\ relevant} = \frac{22.9}{31.8} = 0.72.$$

Taking into consideration the fact that many IR systems stay below the 30% precision next to this fairly high recall, demonstrates that we used a state of the art IR model indeed.

The AET of 44.1 seconds is of course not very competitive. However, note that this is also due to the relatively large (1000), and therefore expensive, top-N we computed, required to legitimate the use of the TREC benchmarks. In Series IV we demonstrate that much better times can be achieved in case of a smaller top-N. Furthermore, the relatively long queries we used (again because of the use of the TREC benchmarks) also are quite costly when evaluated without any special measures like top-N optimization, which we did not exploit in these series, yet.

6.2 Series II: Cut-off Moment

Series II has been designed to develop an intuitive feel for the trade-off between quality and efficiency. Recall that only two fragments are used: a small fragment containing the 'interesting' terms and a much larger fragment containing mainly 'common' terms.

Series IIa: Use First Fragment Only. Figures 8, 9, and 10 plot the AP, the ARR, and the AET of Series IIa, respectively, together with the baseline performance of the unfragmented case. Figure 9 also plots the average number of relevant documents in the collection, averaged over all topics. The x-axis denotes the term count of the first fragment in ‰ (i.e. tens of percentages) with respect to the total number of terms in the dictionary.

The experiments confirm our expectations. The $AP_{fragmented}$ increases with increasing term count of the first fragment, moving towards the $AP_{unfragmented}$. This also holds for the $ARR_{fragmented}$, respectively $ARR_{unfragmented}$. The shape of the plot in Figure 10 is also not surprising; since the data distribution is highly skewed, the data size of the first fragment grows faster and faster with increasing term count of the first fragment; explaining perfectly how the AET increases ever faster as the term count of the first fragment increases, reaching an AET of just over 44 seconds when the first fragment contains all terms (100%).

If the first fragment contains 99% of the terms, the AET is still 3.8 s while the ARR is 16.2 (or, average recall is 0.51) and the AP is 0.27: half of the documents that should have been retrieved are (on average) indeed retrieved, and, the average precision drops only a few percentages (to a level that various custom IR systems would not reach). In other words, a very reasonable quality can be reached in almost 20 seconds, which is more than 2 times faster than the time required to compute the best possible answers (given our retrieval model).

Series IIb: Use Second Fragment when First One is Unable to Handle Query. Even the best known retrieval models don't exceed the 40% AP level. So, although the results shown in the previous case can be considered quite good compared to many other IR systems, the quality degradation still comes down to moving away further from that – already quite poor – upper limit of 40% AP. Series IIb aims to investigate a possible improvement in the quality at the cost of, hopefully, only a minor fall-back in efficiency.

As is clearly demonstrated by the results shown in Figures 8 and 9, the quality of the results has improved significantly thanks to the switching technique. But, the AET has risen significantly (Figure 10). This observation particularly holds for the fragment size ranges below 98.5%. For larger fragments, the AET has stayed the same.

This behavior can be explained by the following argument. The larger the size of the first fragment, the more terms are handled by the first fragment; so, the higher are the chances that at least one of the query terms is contained in the first fragment. But, this also implies that the chances that a switch is needed drop. Conversely, the smaller the first fragment, the system switches more often, to a rather large second fragment; resulting in quite expensive execution costs.

When increasing the number of terms in the first fragment up to approximately 98.5%, the system switches less and less often to the second fragment; and, as the 'data size' of the first fragment is still relatively small, and the second (more 'expensive') fragment is used ever less, the total execution time drops. Up from 98.5%, the first fragment always contains at least one of the query terms. But, from that same point the data size of the first fragment starts to grow faster and faster: causing the AET to rise. Since from the 98.5% point up only the first fragment is used, the quality and performance coincide with the figures obtained at the previous experiments.

As expected, the efficiency of Series IIb is lower than that of the previous experiment, in particular for smaller first fragment ranges. But, the AET is still always 2 times smaller than for the unfragmented case and the quality exceeds, or at least equals (from around the point of 98.5% terms in the first fragment), the levels reached in Series IIa, as we had hoped. The AP never drops below 0.23, and the ARR always stays above 14. Summarizing, the switching procedure does improve the quality, but in more extreme cases also degrades efficiency quite firmly; caused by either switching to an expensive second fragment (sizes smaller than 95%) or always operating on an often too expensive first fragment (up from 99%).

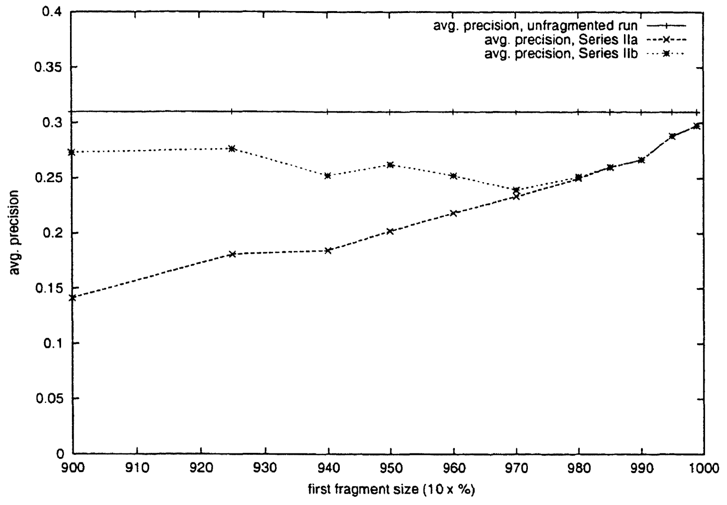

Fig. 8. [Series IIa/b] *AP* for several relative sizes, one fragment used, preferably the first

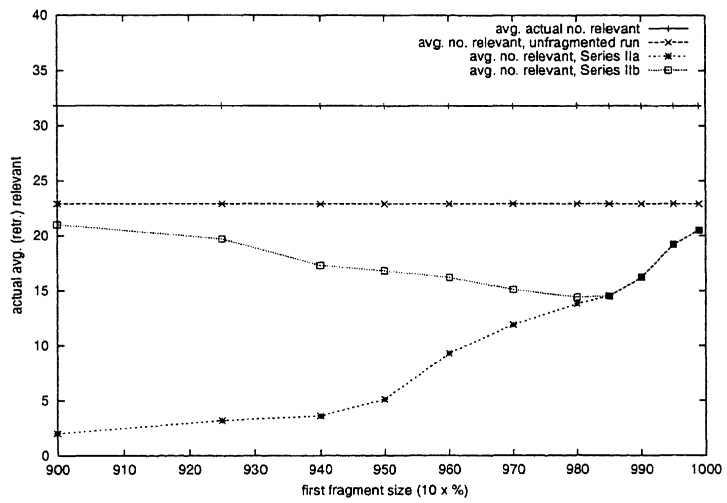

Fig. 9. [Series IIa/b] *ARR* for several relative sizes, one fragment used, preferably the first

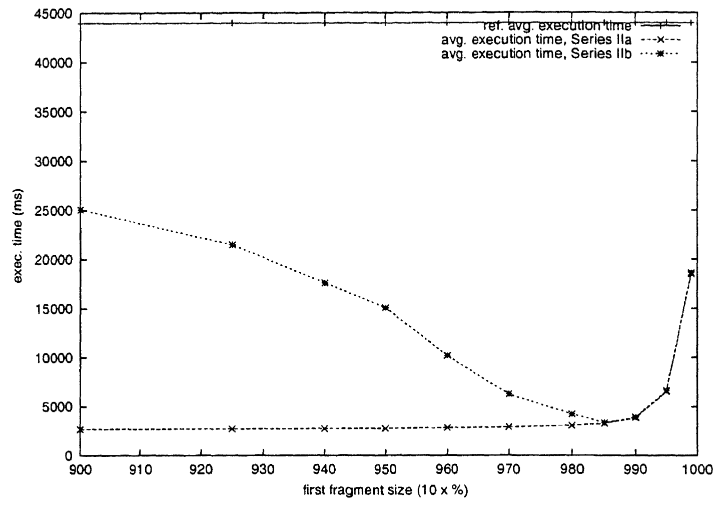

Fig. 10. [Series IIa/b] AET for several relative sizes, one fragment used, preferably the first

Concluding Remarks. These series of experiments clearly show the trade-off between speed and quality. They also demonstrated that, while remaining quite competitive quality, the efficiency of the retrieval process can be increased significantly by using this two-fragment approach.

6.3 Series III: Benefits of Fragmenting

In Series I we already showed the quality and performance results of the 'no fancy tricks' approach. These series focus on the situation where we have many equally sized (in terms of data size) fragments (25 to be precise).

In Table 2 we listed the quality and performance results of the Series IIIa, IIIb, and IIIc experiments. We also included the results of Series I and the TREC benchmark values for comparison.

The results for the Series IIIa, where we only fragmented the database in 25 fragments but did nothing else in particular to speed things up, clearly shows that fragmentation by itself does not introduce any extra costs. Also one clearly sees that the quality has not decreased in any way, as we expected, since all information of any relevance has been taken into account.

The Series IIIb shows the results for the experiments where we used normal safe/unsafe top-N cut-off. As we predicted, the quality has degraded for the unsafe top-N technique (but only slightly) and stayed the same for the safe method.

Although we did anticipate on a poor performance gain for the safe method, the drop in performance was rather unexpected. The unsafe method was expected to perform even more better than the safe approach, but also shows disappointing execution times. This performance degrade of course is the opposite of what we intended to happen. A more close review of our log files learned

Table 2. [Series III] Results of experiments with 25 fragments, with and without top-N cut-off (TREC benchmark and Series I results included for comparison)

	AP (%)	ARR	AET (s)
Benchmark	100.0	31.8	-
Series I	31.0	22.9	44.4
Series IIIa	31.0	22.9	44.8
Series IIIb (safe top-N)	31.0	22.9	50.9
Series IIIb (unsafe top-N)	31.0	22.7	51.0
Series IIIc (l = 0.00)	30.0	15.1	7.9
Series IIIc (l = 0.05)	29.8	15.6	13.5
Series IIIc (l = 0.10)	29.7	15.9	18.7
Series IIIc (l = 0.25)	30.0	17.6	33.0
Series IIIc (l = 1.00)	30.1	22.9	89.1

that the cut-off conditions were too weak, allowing a cut-off in only rare cases. Also the intermediate result restriction technique appeared to suffer from the same weak boundaries resulting in no effective limitation of the computational effort. Due to the extra administrative work needed for the desired but never occurring cut-off this resulted in a performance degrade instead of a performance gain.

However, the reasons for the disappointing results for IIIb also explain the huge performance gain for the IIIc case with low l. For the IIIa (and IIIb) case the computational effort (indeed) appeared to increase for fragments with terms with higher df – e.g. the fragments in the end of the fragment-sequence – due to the Zipfian nature of the data. In case of IIIc the intermediate result restriction did occur with almost no exception, reducing the computational effort per fragment to almost a constant factor. Furthermore, the AP stayed almost the same, while the ARR dropped a bit more. However, the recall still is about 50% in the worst case, which is not really that bad. For the case of l = 1.00 the quality indeed equals the values measured for the IIIa case, as expected. However, the performance for this case is very bad, as one can expect of this naive approach to forcefully rank all documents.

6.4 Series IV

Here we describe the measured performance and quality results when we relax the requirements we used to comply with the TREC evaluation standards till now. The (a) variant shows the effects of shorter queries, whereas the (b) series show what happens when a smaller top-N is delivered.

Series IVa: Influence of Query Length. Figures 11, 12, and 13 plot the AP, the ARR, and the AET of Series IVa, respectively, together with the baseline performance of the unfragmented case.

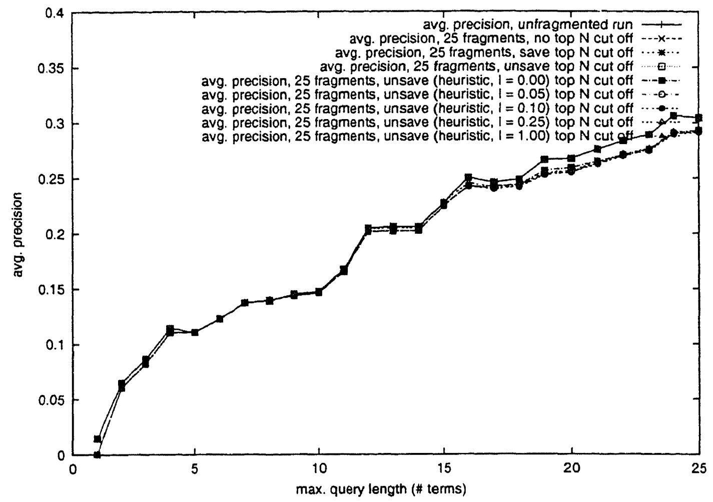

Fig. 11. [Series IVa] *AP* for several max. query lengths, no/safe/unsafe/(heuristic) unsafe top-*N* cut-off

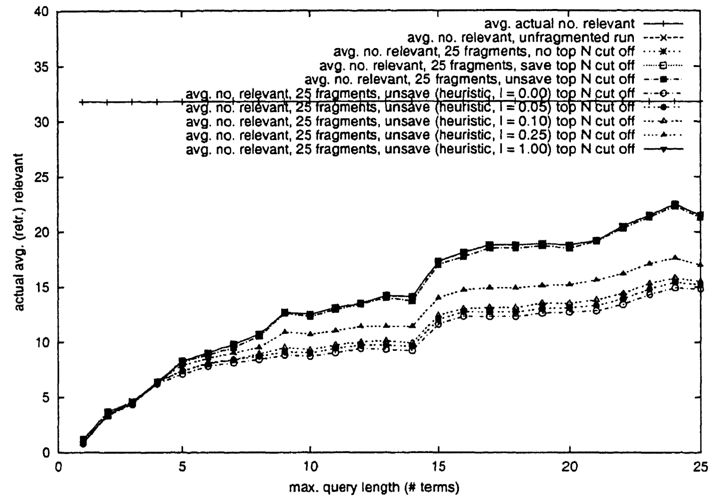

Fig. 12. [Series IVa] *ARR* for several max. query lengths, no/safe/unsafe/(heuristic) unsafe top-*N* cut-off

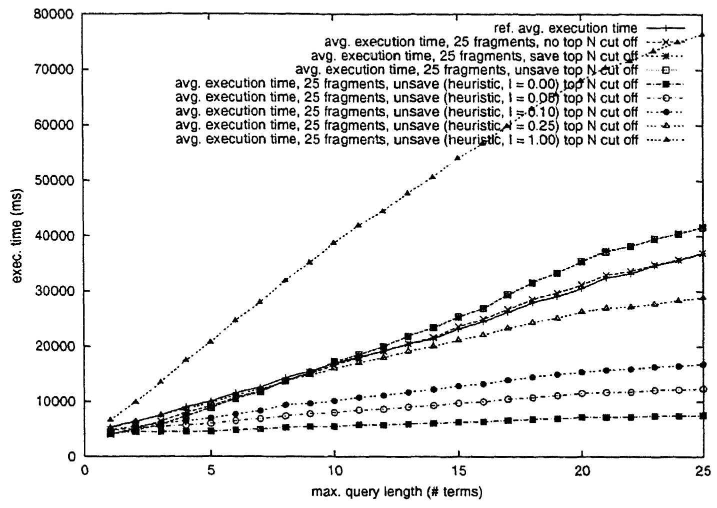

Fig. 13. [Series IVa] AET for several max. query lengths, no/unsafe/unsafe/(heuristic) unsafe top-N cut-off

As expected, the smaller the query length the better the performance. And, although not completely compliant with the usual TREC evaluation standards, we also performed the query result quality evaluation, which not surprisingly, shows a degrade for reducing query length. Again, the normal safe and unsafe techniques do not result in a significant performance gain. The heuristic method, in turn, shows very good performance for the lower l values. For $l = 0.00$ the execution times per query only lightly increases for growing query lengths. However, for growing l the performance collapses quickly.

Series IVb: Influence of Top-N Size. Again, we combined all the results of these series into 3 plots, being the Figures 14, 15, and 16.

We expected the performance to increase for decreasing top-N size. This indeed does happen, but it clearly only happens for really small top-N sizes, and then still only in a minimal form, which is less than we hoped for. Apparently the size of the required top-N does not really affect the computational effort that is required. Probably this has to do with the fact that the top-N cut-off did not really work as we expected and that the main performance gain is obtained from reducing the intermediate results/work. This latter observation is supported by the fact that the heuristic unsafe method always results in significantly lower execution times for lower values of l, independent of the size of the top-N. In general, the differences between the results obtained for the used optimization techniques are clearly visible and resemble the figures we already saw for the (a) variant.

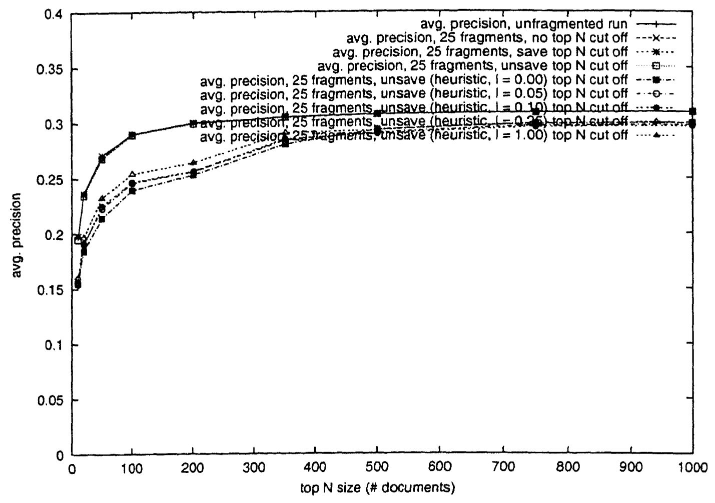

Fig. 14. [Series IVb] *AP* for several top-N sizes, no/safe/unsafe/(heuristic) unsafe top-N cut-off

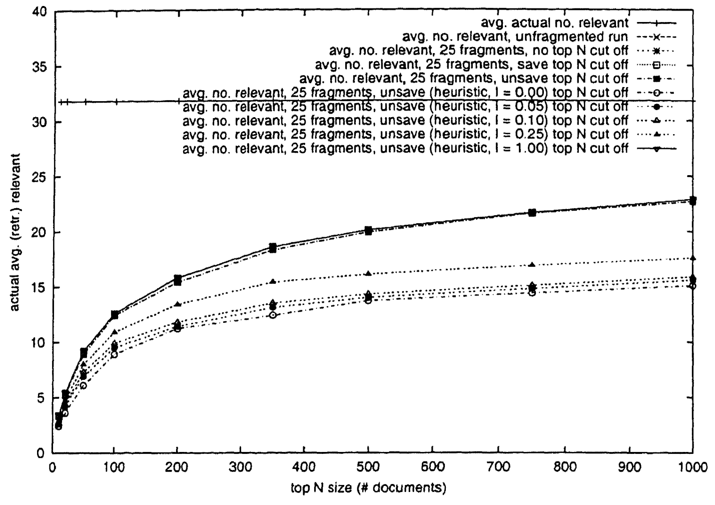

Fig. 15. [Series IVb] *ARR* for several top-N sizes, no/safe/unsafe/(heuristic) unsafe top-N cut-off

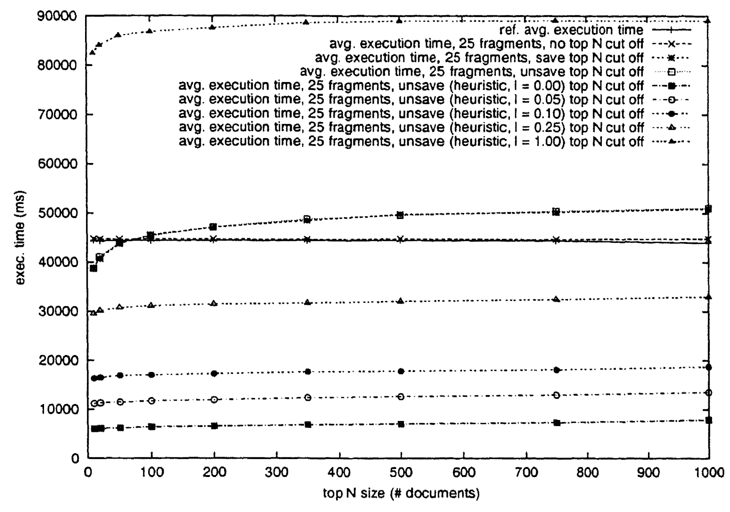

Fig. 16. [Series IVb] AET for several top-N sizes, no/safe/unsafe/(heuristic) unsafe top-N cut-off

Concluding Remarks. The effects we hoped to see for the (a) variant indeed occurred and the heuristic unsafe cut-off technique seems very promising due to its still relatively good quality along with very good performance for low l values. The (b) variant also, in a sense, did show what we expected, but much less significantly than we hoped for. Apparently the size of the top-N is not really an important issue in our case. Future research has to show whether we can improve the top-N cut-off conditions to obtain effective top-N cut-off behavior indeed. Maybe then indeed the size of the top-N will turn out to be of significance.

Fortunately, our heuristic unsafe method does show very interesting performance gain with only minor quality loss.

7 Conclusions and Future Work

This paper presents a convincing case for the suitability of the 'database approach' in the non-standard domain of information retrieval. We first specified the typical IR retrieval process declaratively. This allows the integration of IR techniques in our prototype DBMS, without fixing the physical execution of queries that use these techniques on a predetermined order, which is particularly important for the development of search engines for XML documents, handling queries that refer to a combination of traditional boolean retrieval with retrieval by content.

The experimental validation of our proposed techniques confirm strongly the expected quality versus efficiency trade-off. Series II and III establish the suitability of data fragmentation as an instrument to tailor the physical database design to match the hardware restrictions of the server machines. The final series of experiments demonstrates further evidence in favor of further adaptation of our fragmentation method for top-N optimization techniques.

Summarizing, our results demonstrate convincingly that the smart usage of domain knowledge can significantly improve the retrieval efficiency when operating in a database context. Note that for short queries (i.e. only a couple of terms) the execution times reduce to only a few seconds per query when using our heuristic unsafe top-N cut-off technique. This even outperforms the initial Google of few years ago for uncached short queries [5]. Of course Google then (already) operated on a data collection of about 100 times bigger than the one we used in this paper. Also, such state of the art search engines make use of (query) caching techniques closely related to database optimization techniques like multi query optimization, which we haven't incorporated in our system, yet.

Based on the results of Series II we have already devised a first prototype cost model that seems to predict the execution costs of our fragmented query evaluation approach very accurately (also see [1]). We eventually plan to use this model to optimize the allocation of fragments on a shared nothing parallel system. Next to that we plan to incorporate the cost model in the (physical)optimizer under the logical level of our system (i.e. Moa). This will allow the optimization of the IR part to blend in with the rest of the optimizer, due to the transparent nature of our approach. To evaluate the efficiency gain and opportunities for scalability of this cost based optimizer we are setting up a database with a data collection that is a 100 times larger than the one we used to perform the experiments presented in this paper. We plan to exploit the parallel processing features of our physical (Monet) and logical (Moa) layers to cope with this dataset using a cluster of PCs similar to the one we described in Section 6. Our ultimate goal is to demonstrate that our fragmentation approach indeed does allow seamless integration of multi media information retrieval technology in a DBMS in an efficient, scalable, and flexible manner.

Finally we want to point out that a dedicated IR system most likely always will outperform the best database solution but will lack its flexibility, scalability, and general efficiency. This holds in particular when dealing with both structured and unstructured (like text content) data. Our goal is to find a database solution that at least shows acceptable performance for the unstructured part. We see the results presented in this paper are a first step in the right direction.

References

1. H.E. Blok, S. Choenni, H.M. Blanken, and P.M.G. Apers. A selectivity model for fragmented relations in information retrieval. CTIT Technical Report Series 01-02, CTIT, Enschede, The Netherlands, feb 2001.
2. Peter A. Boncz and Martin L. Kersten. MIL Primitives For Querying A Fragmented World. *VLDB Journal*, 8(2), oct 1999.
3. Peter A. Boncz, Stefan Manegold, and Martin Kersten. Database Architecture Optimized for the new Bottleneck: Memory Access. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *Proceedings of the 25th VLDB Conference*. VLDB, Morgan Kaufmann, sep 1999.

4. Peter A. Boncz, Annita N. Wilschut, and Martin L. Kersten. Flattening an Object Algebra to Provide Performance. In *Proceedings of the 14th International Conference on Data Engineering (ICDE'98)*, IEEE Transactions on Knowledge and Data Engineering. IEEE Computer Society, feb 1998.
5. Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Proceedings of the Seventh International World Wide Web Conference (WWW7)*. WWW7 Consortium, apr 1998.
6. Eric W. Brown. Execution Performance Issues in Full-Text Information Retrieval. Ph.D. Thesis/Technical Report 95-81, University of Massachusetts, Amherst, okt 1995.
7. J.P. Callan, W.B. Croft, and S.M. Harding. The INQUERY Retrieval System. In A. Min Tjoa and Isidro Ramos, editors, *3rd International Conference on Database and Expert Systems Applications (DEXA'92)*, pages 78–83, 1992.
8. Douglass R. Cutting and Jan O. Pedersen. Space Optimizations for Total Ranking. In *Proceedings of RAIO'97, Computer-Assisted Information Searching on Internet*, pages 401–412, jun 1997.
9. A.P. de Vries. *Content and multimedia database management systems*. PhD thesis, University of Twente, Enschede, The Netherlands, December 1999.
10. A.P. de Vries, M.G.L.M. van Doorn, H.M. Blanken, and P.M.G. Apers. The Mirror MMDBMS architecture. In *Proceedings of 25th International Conference on Very Large Databases (VLDB '99)*, pages 758–761, Edinburgh, Scotland, UK, September 1999. Technical demo.
11. A.P. de Vries and A.N. Wilschut. On the integration of IR and databases. In *Database issues in multimedia; short paper proceedings, international conference on database semantics (DS-8)*, pages 16–31, Rotorua, New Zealand, January 1999.
12. Ronald Fagin. Fuzzy Queries in Multimedia Database Systems. In *Proceedings of the 1998 ACM SIGMOD International Conference on Principles of Database Systems (PODS'98)*, pages 1–10. ACM Press, 1998.
13. Ronald Fagin. Combining fuzzy information from multiple systems. *Journal on Computer and System Sciences*, 58(1):83–99, feb 1999. Special issue for selected papers from the 1996 ACM SIGMOD PODS Conference.
14. Ronald Fagin and Yoëlle S. Maarek. Allowing users to weight search terms. Retrieved from authors website.
15. David A. Grossman and Ophir Frieder. *Information retrieval: algorithms and heuristics*. The Kluwer international series in engineering and computer science. Kluwer Academic, Boston, 1998.
16. D. Hiemstra and W. Kraaij. Twenty-One at TREC-7: Ad-hoc and cross-language track. In Voorhees and Harman [18].
17. C.J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 2nd. edition, 1979.
18. E.M. Voorhees and D.K. Harman, editors. *Proceedings of the Seventh Text Retrieval Conference (TREC-7)*, NIST Special publications, Gaithersburg, Maryland, nov 1999.
19. A.P. de Vries and D. Hiemstra. The Mirror DBMS at TREC. In Voorhees and Harman [18], pages 725–734.
20. G.K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, Reading, MA, USA, 1949.