DATABASE OPTIMIZATION ASPECTS FOR INFORMATION RETRIEVAL

Dissertation committee:

Prof.dr. P.M.G. Apers, promotor
Dr. H.M. Blanken, assistant-promotor
Prof.dr. H.-J. Schek, ETH, Zürich, Switzerland
Prof.dr. M.L. Kersten, Universiteit van Amsterdam
Prof.dr. F.M.G. de Jong
Prof.dr. W. Jonker
Dr.ir. R.S. Choenni, Universiteit Nyenrode
Prof.dr. H. Wallinga, chairman/secretary

The research reported in this thesis is part of the national research project Advanced Multimedia Indexing and Searching (AMIS), funded by NWO, the Netherlands Organization for Scientific Research, under grant no. 612-21-201.

CTIT Ph.D.-thesis Series No. 02-41

Centre for Telematics and Information Technology (CTIT)
P.O. Box 217, 7500 AE Enschede, The Netherlands

SIKS Dissertation Series No. 2002-03

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Graduate School for Information and Knowledge Systems.

Cover: Partial df plot and vocabulary (stemmed and stopped) of the intra web of the Faculty of Computer Science of the University of Twente, as indexed during Christmas 2001

ISBN: 903651732X
ISSN: 1381-3617 (CTIT Ph.D.-thesis Series No. 02-41)

Publisher: Twente University Press
P.O. Box 217, 7500 AE Enschede, The Netherlands
http://www.tup.utwente.nl/

Print: Océ Facility Services, Enschede, The Netherlands

DATABASE OPTIMIZATION ASPECTS FOR INFORMATION RETRIEVAL

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof.dr. F.A. van Vught,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op vrijdag 12 april 2002 te 15:00 uur

door

Henk Ernst Blok

geboren op 23 mei 1974
te Hoofddorp

Dit proefschrift is goedgekeurd door:

Prof.dr. P.M.G. Apers, promotor
Dr. H.M. Blanken, assistent-promotor

In loving memory of my grandparents.

Acknowledgments

When I started in 1992 as a math student I quickly started wondering what to do after that. I did consider a job in academic research but was not convinced of whether that would be something for me until, in May 1997, prof.dr. C. Hoede suggested me to do a Ph.D.

A couple of weeks before I finished my Master's assignment he referred me to a vacancy at the Faculty of Computer Science. The official newspaper ad learned that the official deadline for applications had already passed. A math position on a subject I liked seemed to be very difficult to find, and I always had an interest in computer science. So this vacancy immediately gained my interest when prof. Hoede mentioned it. A phone call learned that I could still send in my application. I was invited for an interview and a day later, the day of my graduation, I got the job. I am grateful to prof. Hoede for his trust and initiative.

At the moment of writing four and a half years have passed. In front of you is my Ph.D. thesis counting 217 pages. I found it far from easy to get it all done. I learned a lot about databases, about research, and, most of all, about myself. I have to thank a lot of people for supporting me and making this result possible. I thank you all. I would like to mention some people in particular.

First of all I thank Peter Apers for his useful comments and support. It was sometimes hard to get in touch with him due to his busy schedule but at the crucial moment he was always there. I also thank Henk Blanken for his comments and supervision. He was almost always present to help out whenever I needed it. His experience in database research often served as a valuable source of inspiration.

I express my gratitude to prof.dr. H.-J. Schek for participating in my dissertation committee.

I thank Martin Kersten, Franciska de Jong, and Wim Jonker for participating in my dissertation committee, and for their support and useful comments on my work. I thank Sunil Choenni for all the discussions we had and the resulting

publications. I hope we can continue our cooperation in the future. Also I thank him for joining my dissertation committee.

I thank Erwin for reading and commenting on a draft version of this thesis and letting me help out with his dissertation. The latter experience saved me a lot of problems when formatting this work. I also thank him, Hana, and Wim for the times we went to the cinema or had diner.

I thank Erjen and Djoerd for putting a lot of their time in reading a draft version of this dissertation and providing many useful suggestions, ranging from typos to improvements in the math. I thank Maarten Fokkinga for his TEX assistance. This saved me a lot of time.

I thank Maurice for all the times we had diner at 'De Stek' and the various discussions we had during those diners.

Furthermore, I thank all those people who provided the needed technical and administrative support. In particular I thank all people at CWI for helping me out when I encountered yet another problem with Monet. I thank all the lab people for their support and the freedom and trust to allow me to maintain a cluster of high performance PCs in their 'backyard'. In particular I thank the following people: Lynn Packwood, Jan Flokstra, Henk v.d. Zandschulp, Tonnie Tibben, Ronald Hello, Jan Veninga, Fred Gansevles, Tim Wassink, René Beltman, and Irving Melfor. I also thank Sandra, Els, Suse, and Jetje for taking care of much administrative work ranging from making a simple appointment with one of the busy profs. in my committee to making travel arrangements.

Almost last, but not least, I thank Arjen for his support, comments, and being one of my paranymphs.

Finally, I thank my brother, Jaap, for being one of my paranymphs, and my parents for their comments and especially for being there whenever I needed them.

H.E.
Enschede, April 2002

Contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Introduction

The amount of data that is being stored is growing more and more, both in size and diversity. A typical example is the growth of the world-wide web. Historically several research areas have been working on problems of how to store the huge amounts of data, and how to find relevant data back, once stored.

Several research fields like natural language processing, machine vision, storage media design, library science, databases, and information retrieval (IR) have been developing technologies to store and retrieve complex data collections. In this dissertation we focus on database technology in combination with IR. The research that forms the background for this dissertation is situated in the area where both these fields join. In this work we are particularly interested in certain performance issues in this context.

The remainder of this first chapter is dedicated to sketch the global line of reasoning that forms the backbone of this dissertation. But before we continue, we give an impression of some real-world problems for which this dissertation tries to contribute to a solution.

1.1.1 Scenarios

In this section we sketch a couple of real-world scenarios to provide a context in which this work can be placed. These scenarios are just examples, many more can be thought of. The common factor of all these scenarios is that they require, or would greatly benefit from, an efficient means of combined querying

of highly structured data and what is often called content data. The particular content data type we are interested in, is plain text.

World-wide web

The web clearly has become one of, or maybe even the, largest data collection we know. It contains huge amounts of HTML documents with multimedia objects such as text, images, audio, and video, and many more data types. Current search engines, portals, and directory systems only provide limited capabilities to find the information one might be looking for. But a system with the ability to provide capabilities to combine several means of search information would be very nice. Often not only the content of the web site is of interest but also structured data like, when it was last updated, or who is the author of the page.

Digital library

Digital libraries become more and more common good and often even partially replace the 'old-fashioned' library with books and journals printed on paper and stacked on shelves. Many normal libraries already switched from paper cards to a computerized index system many years ago. This new index provided the easy means to search for words in the title, instead of just via some simple classification means, like publication date and some keywords attached to the work by a librarian. Some libraries nowadays are completely digital, hence the word digital library. A typical example of a popular digital library in the computer science community is the ACM Digital Library [ACM]. In such a library a user typically wants to search not only on title, author, or publication date, but also on the entire content of the documents.

Office information system

As already argued by the authors of AIM [DKA+86] and MULTOS [BRG88], large collections of documents with varying content need to be managed in an office environment. A user typically wants to search on the type of document (whether it is a report or a letter, for instance) but also on the contents.

Newspaper archive

A news paper archive contains a lot of historical information, both in photographs as well as in text. For all this data several properties like the issue they appeared in, the page number on which they were printed, are known.

A user might be interested in all non-frontpage articles about the Euro introduction with a picture, published before 1999.

1.1.2 Research areas

All scenarios we sketched above involve the combined querying of structured data and content data, in particular text. Several means to realize this are known in literature. One of the possibilities is the integration of IR functionality in a database management system (DBMS). We think that this option has some advantages over the other means for combined querying. In particular, we think that some of the key characteristics of a DBMS would be welcome in this context of querying content as well. Which characteristics becomes clear in the remainder of this chapter.

However, integration of IR functionality in a DBMS is not without problems. It has been tried in the past, but either with only a very rigid search technique or without properly enhancing the database system to cope with the accompanying efficiency problems. The contribution of this dissertation is situated in solving some of those efficiency problems.

1.1.3 Outline

In the next two sections, Sections 1.2 and 1.3, we give a brief introduction to both the database and information retrieval areas. Experts in the database field or the IR field might consider to skip either one of these sections or both and continue directly with the next section.

After these brief introductions we sketch an integrated approach, combining both areas in Section 1.4. We think that such a combined approach covers the mentioned scenarios much better than the two separate fields on their own. We conclude this chapter with Section 1.5 where we present the main research questions this dissertation concentrates on.

1.2 Databases

This section provides a basic introduction to some key principles in the database field. An experienced database researcher might choose to skip this section and proceed directly to the next one.

This section is structured as follows. First, in Section 1.2.1, we sketch the typical application domain where database technology is being used historically.

Next, in Section 1.2.2, we introduce some basic principles, the so-called 'goodies'. Third, we briefly describe how a database typically deals with queries from a user, in Section 1.2.3. Fourth, and finally, we mention a crucial DBMS component responsible for the efficiency of database query processing in Section 1.2.4.

1.2.1 'Standard' database applications

The database field mainly focussed on the small area of highly structured data. Common examples are the typical employee administration, or the product-part administration. The relational system have become the most well-known and most popular database systems, and store all data in table format. For example:

EMPNO	FNAME	LNAME	DEPTNO	SALARY
⋮	⋮	⋮	⋮	⋮

describing employees, their department number, and their salary, and

DEPTNO	DEPTNAME	DEPTADDRESS
⋮	⋮	⋮

listing the departments and their addresses.

1.2.2 'Standard' database 'goodies'

To facilitate the management of the data the DBMS provides certain 'goodies'.

The most typical feature of the database approach is that users have to specify their query in a declarative manner. This leaves the DBMS the opportunity to process the query the way it thinks is most efficient. This property is called *data independence*. A typical database query could be to ask for all employees who get paid over € 100,000.00 a year and their department addresses, for instance because you want to send them a letter about something. The DBMS now has several options to evaluate this query. The system might first select all the employees from the employee table that match the salary criterion and then find the corresponding department addresses. But it could also first list the department address for each employee and then select all those entries for which the employee earns over € 100,000.00 a year. Internally in the DBMS these different options to execute a query correspond to a different order in which certain operators are called.

Many other features exist, like transaction management, concurrency control, and distribution, which we do not elaborate on any further.

These features allow the database!administrator and the database!applications, i.e., the software that uses the DBMS, to work with the database in a much more flexible manner instead of when data would have been stored on just a standard file system as provided by the operating system (OS). A file system usually only provides just storage facilities on some storage medium like a hard drive or compact disk with only primitive read and write methods and limited meta information, often not more than owner, size, and modification date. Furthermore, when data is stored on a file system each party would be required to take care of all the facilities typical for a DBMS on its own or in explicit cooperation with the other applications and the OS.

1.2.3 Query processing

To guarantee the data independence property, the architecture of a DBMS is typically divided in three layers: *conceptual*, *logical*, and *physical*. Each query is formulated in some calculus like language at the conceptual level. In the commonly used relational DBMSs this language is usually SQL. Next this high-level query expression is translated into a logical expression, which usually is some kind of relational algebra. Finally, this algebra expression is translated into a physical expression.

1.2.4 Query optimization

Recall our two options to evaluate the query in Section 1.2.2. Every database system has a component called *query optimizer* that is responsible for taking the decisions which way to process a query.

One can imagine that the number of employees and departments in total and the number of both that match the query are important factors in deciding which of both processing alternatives is the most efficient. The query optimizer therefore is supported by a *cost model*. This model is able to estimate properties like cardinalities, i.e., the number of tuples in a table, and tries to estimate how much effort it will take the system to process a given evaluation variant as suggested by the query optimizer. To be able to properly determine the cardinalities of intermediate results, the cost model needs a notion of how many tuples are selected by the operator that produces that particular intermediate result. This notion of how much an operator selects is called *selectivity*. Often this selectivity can not be computed exactly, it can only be guessed. It is the job of the selectivity model to do this as accurately as possible.

The optimizer operates on all three architecture levels. On the conceptual level its task is typically limited to assisting in translating the query expression into some standardized form. At the logical level the typical operator reordering problems have to be solved, like the one in our salary example. When the logical expression is translated into the physical form the optimizer for example has to take into account whether index structures, i.e., special data structures designed to speed-up access to certain parts of the data, are present or not, or whether the data is stored in some ordered format.

1.3 Information retrieval

Similar to our introduction to the database field in the previous section this section provides a very basic introduction to some key principles in the information retrieval field. An experienced information retrieval researcher might choose to skip this section and go directly to the next one.

This section is structured as follows. First, we sketch the typical application domain where information retrieval technology is being used, in Section 1.3.1. Next, in Section 1.3.2, we introduce some basics concerning the ranking of documents.

1.3.1 Information retrieval applications

Information retrieval systems, or short IR systems, mainly concentrate on finding text documents, either just based on its title or on the entire text, often called *full text retrieval*. Usually a user types in a couple of keywords for which the system then returns those documents that it 'thinks' are most interesting given those search terms. Since, usually a huge amount of documents have some relationship to the search terms, only those ones considered most relevant by the system are returned.

Typical application examples of such IR systems are web search engines, (digital) library search systems, and archives. Also applications exist where more than just a few keywords are used to search documents, for instance when searching for patents where it is important to have a more 'precise' answer. And in cases of automatic query expansion queries typically grow longer than just a couple of keywords.

1.3.2 Ranking documents

Many methods exist to search for text documents by using keywords. Boolean search as used by many IR systems in the past, is not considered state-of-the-art anymore, for being too rigid. Modern day IR systems *rank* documents by computing a score for each document containing at least one of the query terms based on a more or less sophisticated statistical or probabilistic model.

Since these models are only a limited view on reality, they are rather imprecise. To overcome this deficiency IR query processing is often not considered as just processing a single query but a sequence of queries plus so-called *relevance feedback*, where the user has a chance to tell the system which results are right and which are wrong. This feedback is then used by the IR system to improve the next results. To allow comparison of different IR systems on their *retrieval effectiveness*, i.e., how good they are in finding the right answers and leaving out the wrong ones, several *quality measures* have been introduced. In the next chapter we describe some of these quality measures, such as *precision* and *recall*, in more detail.

What has stayed the same for all the different retrieval models from the beginning, is the fact that IR systems are mainly focussed on retrieval and not on general data management. Typically IR systems basically incorporate one retrieval algorithm that has been tuned to the data on disk underneath and vice versa: the algorithm is data bound, i.e., not data independent.

1.4 Integrating IR in a DBMS

Recall the scenarios of Section 1.1.1. Clearly, those scenarios would benefit from systems providing both the structured search capabilities typical to the database approach and the content search capabilities typical to the IR approach. This section briefly sketches our view on how such combined querying capabilities might be realized, but also what problems might come along.

In Section 1.4.1 we elaborate on the means to provide combined querying facilities. We also present our choice how to approach this. In Section 1.4.2 we present the problem areas we perceive in integrating IR technology in a DBMS in general. Section 1.4.3 elaborates on the common approaches taken to integration of text retrieval in a DBMS. Finally, in Section 1.4.4 we explain our approach.

1.4.1 Combined querying approaches

Several means exist to provide facilities for combined querying of structured and content data, i.e., text in our case. One could link a DBMS and a dedicated IR system at the application level, leaving the application in charge of combining, i.e., intersecting or unioning, results from both systems. Another option would be to implement the entire IR functionality in terms of the query language used by the DBMS. A third option would be to use the extension mechanism of modern DBMS to add IR functionality to the DBMS, either by incorporating the entire IR system in the DBMS or as a backdoor to a separate IR system.

We choose to use a fourth approach as proposed in [Vri99], by integrating the IR functionality in terms of atomic operators at all three typical levels of the DBMS architecture. In the next sections we elaborate more on the reasons why we think this is a promising direction.

1.4.2 Problem areas

The seamless integration of IR functionality throughout a DBMS is not trivial in our opinion. We distinguish several major differences in approach between IR and DBMS technology. Each of these differences can cause problems.

First of all, database technology historically assumes a closed world in which query results can be proven to be correct, query processing is said to be exact. IR queries in contrast are typically just a keyword representation of the user's information need. Whether the returned answers are correct or not is a subjective matter, to be decided by the user. Often this is called imprecise or inexact query processing. When integrating IR query processing into a DBMS the DBMS needs to be told how to deal with this imprecise behavior. Many approaches have been proposed to solve this. But most approaches where IR functionality was integrated in a DBMS used just boolean search, which nowadays is often considered as too crude to capture this impreciseness.

Secondly, as mentioned before, IR algorithms are often data bound, meaning the algorithm was designed to operate on data stored in a specific way. This obviously conflicts with the data independence property of the typical DBMS approach. Of course, implementing the entire IR algorithm as a black box physical operator would solve this since in a DBMS only at the physical level, being the lowest layer in the architecture, data bound operators can exist. However, such a black box often introduces optimization bottle necks with respect to query optimization. Such an approach therefore clearly is not beneficial to a better performance. So, the only option is to transform the IR algorithm into a data independent variant. This problem also has been

solved before, but only from a query processing point of view. Not much has been done on enhancing the query optimization facilities of the DBMS once IR query processing facilities have been integrated.

Third, recall that query optimization is a typical part of any DBMS. It comprises the runtime reordering of operators using database statistics and a cost model to exploit the data independence property to improve ad hoc query performance. In an IR system this does not really exist. The work on optimizing query processing in the IR field is of a completely different nature. It comprises the development of better and faster retrieval algorithms, or better tuning of the data layout on disc given a basic algorithm. Once the setting has been improved, the old situation is transfered into the new setup. The typical data bound setup is crucial to this approach.

Fourth and finally, recall that IR query processing actually is an alternating sequence of queries and feedback, in contrast with database query processing which typically assumes ad hoc queries. Even in the area of multi-query optimization in the database field, where sequences of queries in time or in parallel are considered, queries are typically not related to each other in such a natural manner, but strictly on arbitrary similarities. So, in the IR case the query optimizer ideally should be aware of this special relationship between queries in time.

1.4.3 Traditional integrated approaches: Boolean search or non-enhanced optimizer

Integrating IR functionality in a DBMS is not new, as it was already implemented in systems like AIM and related systems [SP82], and MULTOS. We think that such an integrated approach is interesting from the point of view that data independence has brought many advantages to the use of database technology. We think those advantages are also welcome in other areas than just those dealing with highly structured data.

As already shown by the MULTOS system this also provides the basic means to integrate the IR query processing into the normal way of database query optimization. We think that such an integrated manner of optimizing both the structured and the content part of the query should be the case to avoid bottlenecks in the means for optimization and as a consequence performance problems.

The disadvantage of the MULTOS approach is that it uses a boolean search technique. As pointed out in the previous section, this is not considered state-of-the-art anymore.

Many more approaches have been proposed in past years to integrate IR in

a DBMS, also partial match and probabilistic retrieval methods. However, to our knowledge, for none of those systems the query optimizer has been reported to be properly enhanced to deal with this new domain. In our view this will sooner or later cause serious performance bottlenecks, in particular when the data collections grow to a nowadays reasonable size.

1.4.4 Our integrated approach: Probabilistic IR with optimizer

Since, the concept of plain boolean retrieval as supported by systems like MULTOS is not considered state-of-the-art anymore, we want to support modern day statistical and probabilistic models. And, in addition, we want to provide enhancements to the query optimizer as well.

In this dissertation we use a variant of the state-of-the-art retrieval model presented in [Hie01]. We integrate this technique in a test DBMS according to the ideas described in [Vri99]. First of all, we are interested in porting existing IR optimization techniques to the database way of query processing and optimization. Due to the data bound nature of most of these techniques, as mentioned in Section 1.4.2, we expect that this is not a trivial matter.

Furthermore, we want to look at means to generate choices for the query optimizer. Recall from Section 1.4.2 that IR optimization research mainly concerned the replacement of an entire algorithm with a more efficient algorithm. A query optimizer in a DBMS is used to reorder operators in the query. A single monolithic algorithm to process a query leaves not much options for such reordering. So somehow we have to split up this algorithm in parts that do allow changes in how the queries, or parts thereof, are processed.

Next, as explained in Section 1.4.2, we assume the queries to be part of an alternating sequence of queries and feedback. The typical IR cut off techniques require some extra administrative work to be able to determine when processing can be cut off. We think it might be interesting if we could throw away this extra administration, saving us the costs, and instead determine in advance which portion of the data we want to use for a given query. But throwing away data in advance might result in ignoring data that would have been important for evaluating the query. So, the results might contain more 'wrong answers', i.e., the result are of a lower quality.

In the first iterations of the querying process a user might not mind a few extra 'wrong answers' popping up, if it goes significantly faster to get those answers. Even more, those extra 'wrong answers' might be useful if negative feedback were allowed as well, thus reducing the number of required iterations maybe even more than when these answers wouldn't have been there.

But a too great loss of quality would certainly have a negative impact on the overall search process. So, we want to predict the penalty on the quality when we willingly ignore data that otherwise might have been used. Only if the quality reduction is small enough compared to the speedup, this option is of practical use. So, in other words, we are also interested in opportunities to trade quality for speed.

1.5 Research objectives

In the previous sections we have sketched the database and IR basics. We also described why the integration of IR technology in a DBMS might be useful. We argued that the approaches in the past lack either good retrieval functionality or a good query optimizer. In line with the last part of the previous section where we briefly presented our approach we now present the main objective in Section 1.5.1. We close the section by presenting the concrete research questions we address in this dissertation, in Section 1.5.2.

1.5.1 Main objective: Efficient set-based IR query processing

Given all we discussed above we can capture our main research objective by asking the following question.

Q *Can IR top-N queries be optimized in a database manner?*

We do not expect that databases will outperform custom built IR systems. But when databases do perform well enough, the additional benefits of having the typical 'goodies' may make databases a competitive means to realize IR systems in the future.

1.5.2 Derived objectives

In more detail this dissertation tries to provide answers to the following research questions. A positive answer to these questions would bring us a step closer to reaching the main objective.

IR algorithms often exploit the fact they are highly data bound by cutting off query processing when some stop criterion has been met. Due to the typical set-based way of query processing in a DBMS this cut off is not trivial. The

only exception to this is when the whole IR algorithm is implemented as a single physical operator. But we decided not to do this for other performance reasons.

Suppose we cut off query processing in advance by pre-selecting a portion of the data that is expected to be sufficient for computing the results. The first thing to know is how much this would speed up query processing. So, we need an accurate cost model to estimate the cost aspects of using a certain portion of the data during query evaluation. The accuracy of such a cost model depends greatly on the accuracy of the underlying selectivity model. This leads to the following research question.

Q1 *Can we estimate selectivity as a function of the used portion of the data?*

Besides this selectivity problem, the quality behavior might be very interesting as well. As argued before, this would allow us to provide the user with a means to control a trade-off between query evaluation speed and answer quality. This leads us to the following question.

Q2 *Can we estimate the quality behavior as a function of the used portion of the data?*

Answering these two questions is the actual contribution of this dissertation, see Chapter 5 and Chapter 6, respectively.

Since, IR data tends to grow huge, the hot set, i.e., the data involved in query processing at a certain moment in time, will not fit in main-memory. So, we somehow have to process the data in batches anyway. The idea is that these batches may be of practical use to serve as cut off points as-well, if properly chosen. One might even decide to determine these batches at indexing time since this can be very time consuming. The process of dividing a table in batches, which form a partitioning of the tuples in the table, is called *horizontal fragmentation* of the table. This observations leads to the following research question.

Q3 *Can horizontal fragmentation facilitate the trading of quality for speed to support set-based IR top-N query optimization?*

Instead of just talking about using arbitrarily sized portions of the data, fragmentation might be used instead, to reduce the number of possible options to divide the data. This might simplify the process at query optimization time,

reducing the time needed to optimize. In Chapter 4 we investigate several aspects of the division of a test data in fragments.

In Chapter 3 we review and adjust these research questions in the light of the related work presented in the next chapter.

Since this third research question concerns a better understanding of horizontal fragmentation in the context of this dissertation, we handle this question, from here on, before the other two.

1.6 Outline of this dissertation

The remainder of this dissertation is organized as follows. In Chapter 2 the related research done in the database and information retrieval field, and the integration of IR technology in databases is described. Chapter 3 concerns a more thorough analysis of the problem area and the followed approach. Chapter 4 reports on the experiences with horizontally fragmenting IR databases, including several results on the porting of IR top-N optimization algorithms to the database environment. Chapters 5 and 6 describe the developed selectivity and quality models, including experimental verification. Finally, Chapter 7 concludes this dissertation.

References

[ACM] *ACM Digital Library*, URL: http://www.acm.org/dl/.

[BRG88] E. Bertino, F. Rabitti, and S. Gibbs, *Query Processing in a Multimedia Document System*, ACM Transactions on Office Information Systems **6** (1988), no. 1, 1–41.

[DKA+86] P. Dadam, K. Kuespert, F. Andersen, H.M. Blanken, R. Erbe, J. Guenauer, V. Lum, P. Pistor, and G. Walch, *A DBMS Prototype to Support NF² Relations: An Integrated View on Flat Tables and Hierarchies*, Proceedings of the 1986 ACM SIGMOD International Conference on the Management of Data, ACM Press, June 1986, pp. 356–367.

[Hie01] D. Hiemstra, *Using language models for information retrieval*, Ph.D. thesis, University of Twente, Enschede, The Netherlands, January 2001.

[SP82] H.-J. Schek and P. Pistor, *Data Structures for an Integrated Data Base Management and Information Retrieval System*, Proceedings of 8th VLDB Conference, September 1982, pp. 197–207.

[Vri99] A.P. de Vries, *Content and Multimedia Database Management Systems*, Ph.D. thesis, University of Twente, Enschede, The Netherlands, December 1999.

Chapter 2

Related work

2.1 Introduction

This dissertation is about optimization aspects for information retrieval. In particular we are interested in the situation where IR functionality is integrated in a DBMS. In this chapter we give an overview of related research. As in the previous chapter we distinguish three main subject areas: databases, IR, and IR integrated in databases. This is reflected by three sections, each about one of these. In Section 2.2 we give an overview of relevant database research issues. In particularly we focus on query optimization and related issues such as cost models and selectivity. Database researchers might consider to skip this section. Next, we describe some basic principles from the IR field in Section 2.3. In this section we focus not only on optimization techniques but also on quality related issues. As for the database related work, experts in this area might consider continuing to the next section directly. In Section 2.4 we elaborate on the state-of-the-art on integration of database and information retrieval technology. Here we do not only give examples of integrated systems but we also discuss to what point these systems provide extra query optimization support for this integrated situation. We conclude this chapter with Section 2.5.

2.2 Database query processing

Relational database technology is several decades old by now and much research has been done in that area since. In this section we focus on database query processing technology. In particular we describe several important is-

sues concerning query optimization. Furthermore, we address the topic of extensibility, which poses some additional problems when dealing with query optimization. We also briefly address the issue of data fragmentation. Next, we give a brief description of the Monet main-memory DBMS, a special kind of extensible DBMS. We conclude this section with a discussion of the top-N query optimization problem, which plays an important role in the remainder of this work.

2.2.1 Query optimization

Like database query processing in general, query optimization has been subject to extensive research since several decades. Good overviews can be found in papers like [Yao79, JK84, Gra93, Ioa96] and several well-known text books concerning (relational) databases in general, like [Ull88] and [Dat95].

In this section we give a brief overview of basic query optimization concepts. In the next sections we elaborate in more detail on those concepts that are more closely related to the work presented in this dissertation.

As we already mentioned, query optimization can take place at and in between any level of the three architecture layers: conceptual layer1, logical layer, and physical layer.

Figure 2.1 gives a schematic overview of a typical query optimizer architecture.

In [Ioa96] a slightly different distinction is made, dividing the processing path in two stages: the declarative, rewriting stage and the procedural, planning stage.

In the rewriting stage, a query is transformed into an equivalent query using general rules. These rules do not make use of any cost information. If the new query is assumed to always be beneficial, the original query is discarded. Otherwise, the new query is sent to the planning stage as a possible alternative, next to the original query expression. In the planning stage all the possible execution plans for the queries it receives from the rewriting stage are produced. Possible execution plans are determined by searching both, what is called the *algebraic space* and the *method-structure space* in [Ioa96]. In other papers like [Cha98] these two spaces together are called the *search space*.

The algebraic space is the space spanned by all algebraic transformation rules. The transformation rules are based on the commutativity and associativity properties of the relational algebra used at the logical level in the DBMS. In the next section we go into the issue of this *logical optimization* a bit more.

The method-structure space is the collection of all possible physical implementation for each logical operator. For a logical join operator several physical

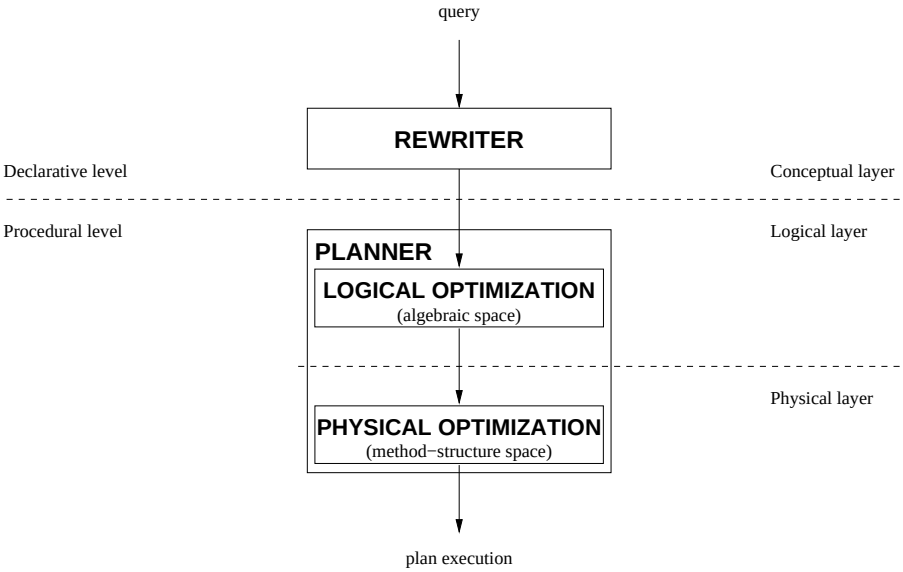

Figure 2.1: Query optimizer architecture

implementations exist, like the *nested-loop* implementation or the *merge scan* implementation. We elaborate a bit more on this issue of *physical optimization* below.

Next to this classification of optimizer tasks based on architectural principles, another classification of query optimization can be made based on the moment in time when it is done. Often, queries are embedded in some application program. These queries are rather fixed and can be optimized at *compile time*. Other queries, such as ad-hoc queries directly entered by a user, are not known in advance and have to be optimized at *execution time*.

The biggest problem of execution time query optimization is how to search the execution plan search space efficiently for the optimal plan. In general the size of the search space is exponential in the size of the query, i.e., the number of operators in the query expression. The main goal is to find a faster query plan. But this search for faster query plans should be very efficient, since longer searching also increases overall execution time. If the loss of time caused by optimizing is bigger than the profit of having a faster query plan, the query optimizer would only slow down overall query processing. We do not elaborate on the possible means to achieve this goal any further since we find it beyond the scope of this dissertation.

Compile time query optimization does not suffer this much from time restric-

tions on the optimization process. Here it may be possible to do an exhaustive search for the actual optimal plan. However, the big issue here is that the queries are fixed in structure but the value of free parameters and the state of the database is typically unknown at compile time.

As explained before, we are dealing with a rather fixed query expression which is typically known in advance. Therefore, the costs of the query transformation process are not really our concern. Though, as becomes clear below, we do have some interest in some parts of the query optimization process to be fast.

In the remainder of this section we describe four parts involved in the query optimization process in more detail. First, we elaborate a bit more on logical query optimization. Next, we take a closer look at physical query optimization. Since a major part of this dissertation is about selectivity and the role it plays in cost models, the last part of this section concerns the related work about cost models and selectivity models.

Logical optimization

At the logical level the optimizer tries to reorder the operators in the query graph by using rules based on algebra properties like commutativity and associativity (i.e., exploring the algebraic space).

These properties are used in transformation rules that are specializations of the common rule to limit the intermediate results as soon and as much as possible. Typical examples are rules to push selections and the most restrictive joins down the operator tree. In [HS93, LMS94] predicate reordering techniques have been proposed that are even more sophisticated.

A typical example are the properties for an algebraic join operator, which can be used to change the order of joins in the query. Given three relations R_1, R_2, and R_3, the commutativity of the join ($\bowtie$) means:

$$R_1 \bowtie R_2 \equiv R_2 \bowtie R_1 \tag{2.1}$$

and the associativity means:

$$(R_1 \bowtie R_2) \bowtie R_3 \equiv R_1 \bowtie (R_2 \bowtie R_3). \tag{2.2}$$

The push-select-down rule does not seem to use cost information. This might suggest that it does not belong to the planning phase as described in [Ioa96]. However, it does utilize the implicit notion of costs that by pushing the selection down the operator tree, the intermediate results are limited in an earlier stage of the query processing. Smaller intermediate results means lower costs.

For the join-order rule something similar holds. The join-order rule needs a notion of *selectivity* to allow proper estimation of the intermediate result

sizes, required to determine which join is the most restrictive one. These intermediate result sizes can again be interpreted as an abstract, implicit cost model.

Given the selectivity of the operators, dumb enumeration through all possible options to find the best solution (i.e., operator order) still is usually too expensive. Therefore, special techniques are used to direct the search in the most promising direction. Note that rules like push-select-down are heuristics, since they usually produce a better plan, but cases do exist in which they make things worse. For instance, a selection that uses an index on one attribute might distort a sort order on another attribute, thus making a subsequent join slower. Whether to use an index, which unintendedly might cause this effect, is to be decided during physical query optimization. So at the moment the selection is pushed down, during logical optimization, this is yet unknown.

At the end of this section we elaborate more on selectivity models. In the next section we describe the optimization at the physical level, the level below the logical level.

Physical optimization

The physical optimization, or optimization in the method-structure space as [Ioa96] calls it, tries to choose the most efficient physical operators for the logical operators in the logical algebra expression. The choice for the right operator implementation depends on the (expected) execution costs of the operator in that particular case, given the query and its place in that query, the (presumed) state of the database, and possible intermediate results at the moment that the operator is going to be executed.

For instance, a merge scan join needs the operands to be sorted. If the operands are already produced in a sorted manner earlier in the query processing this kind of join might be very interesting. However, in some cases, when the operands have not been sorted yet, performing the extra sort may take too much extra time making a nested-loop join, instead of a sort with a merge scan join, much faster.

Another important factor in the choice of the best physical operator is the availability of an index. The optimizer might also decide to build the index on the fly, when it expects this will be a profitable investment that will speed up certain computations.

The cost model is responsible for taking all such relevant factors into consideration when estimating the execution costs. In the next section we elaborate a bit more on this issue.

Cost model

The cost model plays a crucial role, allowing the physical optimizer to make the right choices. Given a query graph, the cost model has to compute the costs for each operator, being a node in that graph. Also it needs to compute the costs of retrieving, storing, and transporting the data. In other words, important factors for the cost model to take into account are the complexity, and related CPU load, of each operator implementation given the properties of its in- and output operands, IO costs, both to and from disk and communication delays such as network transfer costs.

Important in- and output operand properties are statistics like cardinality, sort order, availability of an index, its memory requirements, and many more [Yao77a]. In case of cardinality statistics, the information usually is directly available when dealing with data that is available without any computations. For intermediate results on the other hand the system has to estimate these statistics. Selectivity models, as we describe in the next section, are then needed to take care of this.

Disk IO estimation also requires a notion of selectivity in the sense that disks are usually accessed as a block device [Yao77b, IB86]. In the next section we go deeper into the subject of selectivity models. However, since we are mainly interested in a main-memory DBMS, we do not elaborate on this type of costs any further. But we do have to stress that most traditional commercial DBMSs work disk-based and since disk IO is usually much more costly than memory IO, many costs models are dominated by the disk factors.

One of the big problems for a cost model to do its job is that it has to do a prediction in advance of actual query execution. However, during query execution the state of the whole system changes constantly. Figures like network load, memory load, CPU load, and such are hard to predict and therefore often taken as constant [CHS99]. Also, a cost model usually tries to present the expected costs. This seems reasonable, but due to the large errors that are usually inflicted by the previous assumption and the fact that selectivity estimation is usually not very accurate [IC91], it often makes a huge error. Choosing the right plan based on this prediction is therefore prone to errors. In parametric cost models [Gan98, CHS99] this problem is attacked by leaving the uncertain runtime parameters free, often just restricted within a certain domain. The query optimizer then generates a query plan with alternative branches or a plan that is optimal for the general class of queries generated by the free parameters. At runtime the cost model fills in any open parameters when a choice between branches has to be made. This results in the use of the most accurate status information available which often leads to more efficient query execution.

Selectivity

Both, during logical and physical query optimization, a good notion of the selectivity of the operators is crucial. Each good DBMS typically maintains a wealth of easy maintainable statistics about the base relations. Usually these statistics are cheap to get, since they are usually relatively small in size compared to the relations they are related to and already available in the system without the need to compute them on demand.

To be able to estimate the intermediate result sizes a notion of selectivity of each operator is required.

In the literature, a large number of efforts has been reported on the prediction of selectivity factors in different contexts and under different assumptions [Car75, Yao77b, Mut85, IB86, LNS90, CR94, IP95, GGMS96, PIHS96, CMN98, CMN99]. Roughly two directions can be distinguished in the prediction of selectivity factors.

Research in the first direction has been focussed to the prediction of the number of page or block accesses, to retrieve τ tuples from R tuples which are randomly distributed on B blocks. This problem has been extensively investigated leading from open [Car75] to closed mathematical formulae [Yao77b] for predicting the selectivity. To accelerate these formulae, researchers came up with approximations of these formulae [IB86].

The second research direction mainly focuses on the prediction of intermediate join or selection result sizes. This area has also been subject to extensive research and can be divided in four categories:

non-parametric ad-hoc techniques to administer the attribute value distribution (example: histograms),

parametric the data distribution is approximated by a mathematical distribution function with one or more parameters (examples: uniform, normal, Pearson family, and Zipf [Zip49] distribution),

curve fitting a polynomial (or other function) is approximated using a least squared error estimation method,

sampling queries are run against a sample taken from the real data to compute the selectivity for that sample, followed by extrapolating the selectivity on the sample to the whole data set.

We refer to [CR94] for a more detailed description of each of these.

Many commercial systems use the histogram method, and some still even use the trivial histogram, i.e., they assume the data to be uniformly distributed [Ioa96].

In [Lyn88] explicitly the problems are addressed that arise in selectivity estimation when dealing with highly skewed data, like for instance Zipfian distributed data in textual databases. A method to allow user-defined selectivity estimators is proposed. For a special distribution like Zipf, this would allow the user to provide a specialized model that is able to do the job much more accurately than the general purpose selectivity estimation techniques used otherwise. Furthermore, it analyses several selectivity estimators with respect to a typical text database.

2.2.2 Extensibility

To meet the growing need for flexibility demanded by the growing number of new applications that are being supported, many DBMSs nowadays are extensible. These systems allow additional modules to be added with extra data structures, new operators, or complete ADTs. Three well-known examples of extensible DBMSs are Informix [Inf], Starburst [HCL+90], and Postgres [SR86]. An extension module is often called *datablade*, a name that originates from Informix, or *cartridge*.

The ADTs provided by extension modules may contain very expensive methods, which pose special problems to the query optimizer [Hel98]. Typically, the query optimizer has no built-in knowledge to deal with the operators and data structures in the ADTs. In the Predator DBMS [SLR96, SLR97, SP97, Ses98] this problem has been attacked by introducing the notion of *enhanced ADT* or *E-ADT*. By adding an optimizer extension, the enhanced part, along with the normal ADT, the central optimizer now can delegate optimization concerning the ADT in question to that optimizer part that is dedicated to that ADT.

A typical example (also see [BVB99]), similar to the ones used in the Predator publications, is about an image ADT with two operators, *CLIP* and *ROTATE*:

Example 2.1
The CLIP operator takes as operands an image and the x and y coordinate of the upper left and lower right corner of a rectangle within the bounds of the given image.

Suppose we have an image myimage of 40 *pixels wide and* 60 *pixels high.*

An example use of the CLIP operator would be:

$$CLIP(myimage, 10, 20, 19, 39) \tag{2.3}$$

which produces a sub image of 10 *pixels wide and* 20 *pixels high of myimage.*

The ROTATE operator takes as operands an image and the angle to rotate over in degrees, counter clockwise.

An example use of the ROTATE operator would be:

$$ROTATE(myimage, 90) \tag{2.4}$$

which produces a copy of myimage placed on its left side.

The following query, combining these two operators, is also valid:

$$CLIP(ROTATE(myimage, 90), 10, 20, 19, 39) \tag{2.5}$$

which produces the lower left corner area of 10 *pixels wide by* 20 *pixels high of the rotated version of the image.*

An alternative expression with exactly the same answer is:

$$ROTATE(CLIP(myimage, 20, 40, 39, 49), 90) \tag{2.6}$$

Since the ROTATE in the second expression has to rotate a smaller image than in the first expression, whereas the region for the CLIP stays of the same size, the second expression is executed faster.

The E-ADT system provides the means to the optimizer to translate the first expression in the example above, when encountered, into the second one. This operator reordering is comparable to the push-select-down principle mentioned before. It follows the general rule to limit the intermediate results as much and as soon as possible.

2.2.3 Data fragmentation

Fragmentation of relations is a commonly accepted means to divide data in a database in predefined portions. The basic principles can be found in almost any standard database textbook. We use the description given in [CP86].

Two basic forms of fragmentation can be distinguished: *horizontal fragmentation* and *vertical fragmentation*. A horizontal fragmentation of a relation is a set of relations that form a partitioning of the tuples in the original relation. The elements of the fragmentation are called *fragments*. Being a partitioning, means that the fragments of a horizontal fragmentation do not overlap, and together contain all the tuples of the original relation. A vertical fragmentation is the dual counterpart of the horizontal fragmentation: the columns are partitioned instead of the tuples.

The way a relation is fragmented is supposed to be dependent on properties of the relation's own attributes. In some cases this requirement cannot be met. A special horizontal fragmentation exists, where the horizontal fragmentation

of another relation can be used to fragment the relation in question. This is called *derived horizontal fragmentation*.

Note that fragments are relations themselves, and therefore can also be fragmented, either horizontally of vertically. So, fragmentations can be nested.

2.2.4 Monet

Monet [BK95, BH96, BK99, BMK99] is an extensible—textrm, low-level main-memory DBMS intended to support 'new application domains'. Monet uses a binary relational model, which means that all tables have exactly two columns. Such a two-column table is called a *BAT* in Monet. BAT is an acronym for *binary association table*.

Any relation that is wider than two attributes has to be stored in a fully vertically fragmented manner. Usually each original column is placed in the tail of a BAT with an artificial key head column. By joining on the head column the original wide relation can be reconstructed. The reasons to work with BATs only, is that in the new applications often only a few attributes are used in the actual query processing. Dragging along the rest of each tuple would only take up memory. Of course, in the end, some extra (semi)joins might be needed, compared to traditional disk-based DBMSs, to re-attach any required other attributes.

Being a main-memory DBMS means that the *hot set*, the data needed to perform an operation plus the results of that operation, has to fit in main memory at all times. The advantage of working in main-memory is that the system is relatively simple compared to a disk-based DBMS. This in particular has the advantage that it is very fast and predictable. When the data does not fit in main-memory it has to be horizontally fragmented by the database administrator or the user.

MIL, the *Monet Interface Language*, offers a set of low level algebra operators that are close to physical operators and is not really intended as an end-user language. The Monet philosophy distinguishes two kinds of query optimization:

Tactical optimization This optimization takes place at run-time. It takes the actual system state and exactly known properties of the real input into account. So, it does not suffer from estimated or lacking knowledge of the system state and estimated values of the properties, which is usually the case for classical optimizers.

Strategical optimization This optimization is seen as a task of the front end (for instance an SQL to MIL translator layer) and should operate

> without any knowledge of physical properties. It is assumed that the best plan is the one with the smallest intermediate results, and that this should be the main goal of this type/phase of optimization. Since it is seen as a task for the front end, not much more is said about it in [BK99].

[BK99] stresses explicitly that these two levels of optimization should not be confused with the classical notion of logical and physical query optimization. However, strategical optimization is certainly a part of the logical optimization as we view it.

2.2.5 Top-N query optimization

In many modern DBMSs some top-N optimization features have been incorporated by now in the form of a `STOP AFTER`, `BREAK`, `LIMIT`, or similar directive. The current version of Monet does not support such top-N optimizations, yet. But, work is being done in that direction [KN98].

Top-N queries [CK97a, CK97b, CK98, CG99], and the closely related iceberg queries [FSGM+98], form an important class of queries in the new application domains.

Top-N queries can be defined as queries:

> (...) that request a certain number of answers (N) having the highest (...) values for some attribute, expression, or function.
> [CK97b]

The naive approach to handling a top-N query is to retrieve attributes, or compute a function thereof, for all tuples, sort the result in descending order and present it to the requesting application. Next, leave it up to the application or end-user to use the typical cursor API to retrieve the first N results and just discard the rest. Obviously, this causes a lot of unnecessary work done by the DBMS. Top-N query optimization is the class of optimization techniques to reduce this waste of work and ideally have the DBMS produce only the required N best tuples as end result.

Several techniques have been proposed to push the top-N work down the operator graph, similar to pushing selections down the operator sequence. Basically a top-N operator can be seen as a sort followed by a positional selection. In [SSM96, RDR+98] dealing with the ordering is addressed in particular. In [CK97a, CK97b, CK98, CG99] similar techniques are exploited to sort and limit the intermediate results as soon as possible. However, sorting earlier on, usually means sorting bigger intermediate results, which we do not want, since sorts are relatively expensive. Since the sort needs to be done before the

selection, which in turn preferably is performed as soon as possible, this poses a problem.

The probabilistic top-N optimization approach proposed in [DR99a, DR99b], attemps to overcome this problem. This approach uses the distribution of the attribute, expression/function outcome to guess a value range that covers the desired top. Using this range a normal select on this range can be inserted before the sort to limit its input operand size. This normal select can then be pushed down by using normal push-select-down rules. It is demonstrated that thus intermediate results can be limited fairly good early on in the query plan, without the drawback of the early expensive sort to do the positional select. But, this method also has a disadvantage. If the guess of the range to pre-select is too restrictive, a, what in [DR99a, DR99b] is called, restart is required. This restart has a certain overhead penalty attached to it. Structurally overestimating the range can of course prevent this restart from happening but would introduce a structural extra work load. The trick is to provide the right probability distribution to the model to make the most optimal guess.

Most of the required statistics are already available in the DBMS for other purposes like selectivity estimation (see Section 2.2.1). However, also the same statistical accuracy problems do apply here.

2.3 IR query processing

Like DB research, IR has been subject to investigation for several decades. In this section we give a brief overview of the IR topics relevant to this dissertation. First, we describe the relevant methods and concepts to explain the basics of full text information retrieval. Next, we give an overview of techniques to optimize IR top-N queries. In the third and final section we describe the most commonly used retrieval quality metrics and the results in literature concerning the trade-off between retrieval effectiveness and efficiency. For an overview of the IR field see [Rij79, RH96].

2.3.1 Methods and concepts

Since browsing all documents known to the system for each query is too expensive, they typically are indexed before they can be queried. How this is done depends on the used retrieval model. Usually several kinds of information are extracted, such as which word occurs in which documents. The resulting *metadata* has a similar purpose as an index in a database: speed up query evaluation. Often, stop words, words that occur highly frequently and are almost none discriminative, such as articles, are removed from the documents during

indexing and from the query before actual query processing. This process is called *stopping*. Also, to reduce the diversity of terms, all words found in the documents and queries are reduced to their stem, i.e., *stemmed*, before being passed for further processing.

An IR query typically is a list of keywords, which, depending on the retrieval model, are linked together by certain operators.

IR strategies can be divided into four categories [RH96]:

boolean keyword/exact match systems In a boolean system the query is a boolean expression formed of keywords and boolean operators. A document matches the query when it contains query terms as specified by the boolean expression. The exact match variant is a special case where only the keywords are specified by the user. The actual query is assumed to be a boolean query with the same keywords linked by `or` operators. Documents are usually not ranked. If ranking is required properties like the number of query terms matching the document are taken into account.

vector-space model Both query and documents are represented as a vector in the term space. Each coordinate in the vector corresponds to a term being present (value is 1) or not (value is 0) in the query or document, respectively. Each document is then ranked by the distance of its vector to the vector of the query. The closer the document vector is to the query vector the higher the rank of the document.

probabilistic models The query and the documents are matched via a probability mechanism. A popular approach, based on Bayesian networks, is used in the well-known INQUERY system [TC90, CCH92, BCC94]. In this particular probabilistic approach documents are checked for how much 'proof' they provide for the query. Documents that provide better support are ranked higher than documents that provide less 'evidence'.

AI approaches The document collection is seen as a kind of knowledge base. A query is then seen as a predicate over which one can reason with respect to the domain provided by the knowledge base. The advantage of the answers provided by such a retrieval system is the explanation of why the answer was returned, which is typical to knowledge base reasoning systems.

An interesting state-of-the-art probabilistic model has been proposed by [Hie98, Hie00, Hie01], which has proven to work quite well. It is based on a language modeling approach [PC98]. The well-known $tf \cdot idf$ retrieval model, which is used in many approaches to ranked retrieval [SB88], can be seen as simplified variant of this well-founded retrieval model.

The key parameters in the $tf \cdot idf$ model are:

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

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

inverse document frequency: For each term, idf is the inverse number of the df.

The score of a document is then computed as the sum of the product of tf with idf for each query term that occurs in the document. The higher the tf the more relevant the document is assumed to be with respect to that query term. However, terms that occur in many documents are less discriminative than terms that occur in fewer documents. The idf factor expresses this relation between the document frequency and the 'relevance' of the query term in question.

After all (or enough, as is explained below) documents have been evaluated, the documents are ordered on descending score and the top-N is returned.

Note that to rule out negative biases inflicted by the length of the documents and such the tf and idf are often normalized in some way. Many variants exist within the class of $tf \cdot idf$ retrieval methods.

Due to the mismatch between the user's information need and the query language semantics of the IR system, both at the query formulation and the returned answers a gap between what the users want and what the system 'thinks' the user wants occurs. To reduce this gap IR querying is often setup as an iterative procedure. The user formulates his/her information need in the form of a query. This query usually just consists of a set of search terms. The IR system then tries to do its best to return the right answers. The user then gets a chance to tell the system which answers are good and which are bad. This is called *relevance feedback* [Roc71]. The system then tries to exploit this extra information to improve the results, after which the user again can provide feedback. This process continues until the user stops, either being satisfied with the final results or, in the unfortunate case, being disappointed in still not having received what he/she is looking for.

2.3.2 Top-N algorithm optimization

In this subsection we provide a brief overview of the most important optimization techniques in the IR field.

This section is divided in three paragraphs. The first paragraph describes the basic algorithm on which most of the optimization techniques are based. The second paragraph shows the possible consequences for the quality of the returned answers when using certain versions of this algorithm. The last paragraph describes some issues for physical design of the data storage which might help increase the profits of the algorithm in the first paragraph.

Effects of a smarter algorithm

In IR-research a lot of effort has been done on optimizing query processing. Most of the work done in this area is based on some ideas from [Fag98, Fag99, FM00]. In these papers several algorithms are proposed that stop executing as soon as the required answers are computed. Most important is that these algorithms, and especially the stop criteria used, are proven to be correct. This means that using these algorithms under the given constraints results for sure in the required set of answers or a superset of those.

The basic principle of the algorithms goes as follows (loosely formulated):

Given two sequences of equal length, say $A = [a_i]$ and $B = [b_i]$, of numbers, with $|A| = n$ and $|B| = n$.

We want to compute a subsequence D of C, with $C = [c_i | c_i = a_i \cdot b_i]$, where $|D| = N$ and $min(D) \geq max(C - D)$, meaning D contains the N highest elements of the product of A and B.

The steps of the algorithm are then (roughly speaking):

1. Suppose we order one of the sequences, say A, descending. The resulting sequence A' is therefore a permutation of the original sequence A. Let's call this permutation π. Then:

   $$A' = [a'_i | a'_i = a_{\pi(i)}]$$

   such that:

   $$\forall i \in \{2, \ldots, n-1\} : a'_{i-1} \geq a'_i \geq a'_{i+1}$$

2. Create a sequence B', that has the same permutation π as A':

   $$B' = [b'_i | b'_i = b_{\pi(i)}]$$

3. Create an empty set E.

4. For each $i \in [1, \ldots, n]$ (in ascending order) do:

   (a) Compute $v = a'_i \times b'_i$.

(b) IF $|E| < N$ OR $v > min(E)$ THEN $E := E \cup \{v\}$.

(c) IF $|E| > N$ THEN $E := E - \{min(E)\}$.

(d) IF $|E| = N$ AND $max[b'_{i+1}, \ldots, b'_n] \times a'_i < min(E)$ THEN stop looping over i.

5. Create sequence D from set E by ordering the elements ascending. D is the requested top-N of the product of A and B.

Both [Bro95] and [CP97] use (modified versions of) this idea to calculate the document score-contribution per term.

When using this algorithm for computing the $tf \cdot idf$, tf can be taken as the A and idf as B, or vice versa. In fact, this means that one computes the $tf \cdot idf$ of the most promising tf (or idf) before the $tf \cdot idf$ of the lesser promising tf (or idf), hoping that not all tf (or idf) values have to be taken into consideration but only a certain, hopefully small, set of 'best' ones. Of course this kind of 'smarter' algorithm takes some extra administrative overhead that makes the algorithm only profitable in cases where the considered part of the ordered sequence A is small enough to compensate this overhead.

Since words in natural language are typically Zipfian distributed [Zip49] many terms exist in the system with high idf and only few with a very low idf. Due to the Zipfian distribution these many high idf terms take only very little storage space compared to the few low idf terms. As said, the high idf terms are the a-priori most interesting ones. So, in the case of taking the idf as A the Zipfian behavior of the data implies that chances are high that indeed only a very small portion of the data needs to be evaluated to compute the top-N. Therefore it is very likely that such a 'smarter' algorithm is profitable.

Note that tf and idf are not of equal length, so formally these two sequences cannot be used in the mentioned algorithm. By repeating each idf value for all tf for that term this little problem can be fixed. Of course, with a slight, more or less trivial, modification of the algorithm, the desired effect can be achieved without repeating values.

Other publications on IR top-N optimization are [Per84, BL85].

Safe and unsafe methods

An extra option is to skip the last step of ordering the final results. In the IR context this might be very well acceptable for several reasons. Two of these reasons are:

- The top-N is the most important issue, the ordering is often less important.

- If the elements in E do not differ much, the ordering of E is disputable, since these figures result from statistically based computations of which the error margin might be much greater than that inter-element margin.

This point also brings us to another point in general, noted by [Bro95] as the difference between so-called *safe methods* and *unsafe methods* of optimization:

safe methods This class of methods still returns (at least) the required answers in the desired form (i.e., ordering and such).

unsafe methods This class of methods promises to return good answers, but does not guarantee to return all the required answers or in the desired form. Correct answers might be replaced by other, usually just a little less optimal, answers. And, as already addressed above, the answers might not be ordered correctly.

Effects of physical design

Though we are mainly interested in a main-memory environment we do describe some issues since they provide a good impression of what proper data alignment might do. In a main-memory environment where not all data fits into memory and therefore has to be fragmented, these techniques can be of interest.

By placing the data on disk in the right order, with the right clustering and appended by some efficient access accelerators, a lot of time can be saved, just streamlining disk read IO. [Bro95] uses properties of the IR level to place the document-term statistics efficiently ordered in an inverted file structure. The statistics are stored in descending *idf*. This way two effects are achieved in one:

1. using an algorithm as described above,

2. considering a lot of different terms per block read operation. The reason for this is that a large *idf* means that a term occurs in only a few documents. This results in a small data segment in the inverted file for a certain term (per term the list of documents is small). In turn, this means that a lot of those small document-per-term lists can be stacked in one block.

[Bro95] implemented and tested his ideas using INQUERY, and claims interesting performance improvements, and only a minor decrease in quality.

2.3.3 Quality metrics and trade-off

Since IR is by no means exact retrieval, in contradiction to querying a DBMS containing only alpha numerical data, *the* good, unique answer does not exist, or cannot be determined. Only good answers and better answers exist. And even the question which answer is better than the other one is hard to answer solidly.

To measure answer quality, or effectiveness, several metrics have been developed. Here we describe the most commonly used ones [Har94]. The two most well-known basic quality metrics are:

precision The fraction of the returned top-N documents that is relevant according to the user.

recall The number of relevant documents, according to the user, in the returned top-N relative to the total number of documents known to the system that should have been returned, according to the user.

Actually, precision and recall are related. The higher the one, the lower the other. Both measures can be combined in several ways to get a single quality figure. One of the most commonly used ones is the *average precision*. This measure corresponds with a user who walks down a ranked list of documents and will only stop after the user has found a certain number of relevant documents. The measure is the average of the precision calculated at the rank of each relevant document retrieved. Relevant documents that are not retrieved are assigned a precision value of zero. For example, if three relevant documents exist in the collection and they are retrieved at rank 4, 9, and 20, the average precision would be computed as $\frac{\frac{1}{4}+\frac{2}{9}+\frac{3}{20}}{3} = 0.21$.

Furthermore, in systems where the querying process is an interactive one with relevance feedback from the user, even the worse elements in an answer set might serve a good purpose of providing the user with an option to tell the system what he/she explicitly does not want.

Therefore a great reduction in query execution time might very well be preferable over a 'better' answer of which the computation takes much more time. This has lead to the idea to trade quality for speed [BMN94, JFS98, ÇJF+98, WB00].

The VODAK object-oriented DBMS [BMN94] incorporates retrieval effectiveness as additional parameter in its cost model to allow a trade-off between speed and quality for the retrieval of SGML documents. Unfortunately, not much is said how this parameter interacts with the rest of the cost model.

In [JFS98] two strategies are proposed: a ranking algorithm that adapts its behavior depending on what is already available in the buffers maintained

by the retrieval system, and a buffer replacement algorithm that is ranking-aware. This allows for much more efficient use of the data already available in the buffers, saving a lot on disk IO. In the paper experimental results are presented for a series of query refinement cases. Query refinement is the process of adjusting the query based on its results and is one of the ways to implement relevance feedback.

In [ÇJF+98] a brief overview is given of several means and applications of trading quality for speed. Most of the techniques mentioned are unsafe optimizations similar to the ones we mentioned earlier. Next to the application of these techniques to retrieval systems where the user is waiting for an immediate answer, also the passive task of information filtering is mentioned. In that case the user pushes a query to the system and periodically receives batches of data that the system renders as relevant. User profiles are used to configure the trade-off between speed and quality. Since, the user is not waiting for an immediate answer more time can be spend, if desired, to sift out unwanted documents. If the user wants information on a more frequent basis, the system has less time to do the sifting and thus produces more garbage.

2.4 Integration of IR and databases

In this section we give an overview of related work concerning the integration of IR technology in a DBMS. First, in Section 2.4.1 we give some examples of systems that provide integration on the functional level. In other words, we give examples of systems that, to some level, provide the means to query text, or content in general, in combination with typical structured attributes. In Section 2.4.2 we describe which retrieval technologies are supported by the integrated systems. Section 2.4.3 describes how the integrated systems provide the retrieval functionality. As we already mentioned in the previous chapter several means exist, ranging from coupling an off-the-shelf IR system with a DBMS to fully integrating on all levels of the DBMS architecture. Finally, in Section 2.4.4 we discuss the rather few proposals to really enhance the query optimization process to deal with IR processing properly.

2.4.1 Examples of integrated support

Many approaches to integrate content retrieval, and IR in particular, in a DBMS, have been proposed or reported in literature. In Section 1.1.1 we already mentioned AIM [DKA+86] and MULTOS [BRG88]. In this section we give some examples of proposals and systems that provide specialized functional support for multimedia documents.

Not much work has been done on the area of query languages. Structured data is usually queried using a typical database query language like SQL. The IR part is expressed in the form of keywords, sometimes connected by boolean operators. Combining these two forms of expressing queries is quite straightforward. The keywords are embedded as parameters, any implicit request to rank documents is made explicit in the form of one or more operators, and any boolean operators are either passed as parameters with the keywords, or represented by the boolean elements provided by the database query language. In some cases the query language was defined with combined querying of structured and content data in mind, such as in the case of MULTOS. However, due to the boolean search method used, such query languages usually still look like a typical database query language.

The way these combined queries are processed and how the data is stored is of more interest, so we focus on that in the remainder of this section.

One of the earliest proposals for the integration of text search in a DBMS is [SP82]. It forms the basis for the AIM system [DKA+86] and the Darmstadt Database Kernel System [PSS+87]. In [SP82, DKA+86] the NF^2 data model is presented as a more natural means to model complex data, and in particular text collections. NF^2 is an acronym for *Non First Normal Form*. In this model the first normal form required by the standard relational model is dropped, thus allowing relation valued attributes. In other words, it allows sets to be nested. This allows a text document to be modeled as a set of words. A collection, or set, of documents then becomes a set of sets of words. For example, structures like the following can be constructed (using our own notation) as:

```
DOCCOLL: SET<
           TUPLE<
             DOCID:  STRING,
             AUTHOR: STRING,
             TEXT:   SET<
                       STRING
                     >
           >
         >
```

In this example, a document collection `DOCCOLL` is described, containing documents with an author name `AUTHOR`, an id `DOCID`, and containing the text `TEXT`. The `TEXT` attribute is a set itself containing words in the form of strings.

Over the years, researchers with more or less close ties to the authors of [SP82] have been developing several DBMSs with text retrieval support [Sch93, BA94, BMN94, FR97, GBS99].

In [Bur92] an architecture is proposed for a retrieval system based on ideas

borrowed from the relational model. Also, data independence is seen as a crucial property for such a system. However, the main focus is on the retrieval part. It just borrows some basic architecture ideas from the database field.

Similar to the method already proposed in [SP82], an NF^2 approach is advocated in [JN95]. However, the authors reported to have built only a prototype in Prolog which, as they already mention themselves, does not scale to more realistic collection sizes.

Another interesting approach, closer to home, is the miЯRor system [VB98a, VB98b, Vri98, VDBA99, Vri99, VW99]. Though, the miЯRor architecture concerns a distributed multimedia DBMS in general, experimental work has been done in the area of information retrieval. The miЯRor metadata database component is a crucial part in the architecture. A major part of [Vri99] concerns this component. The presented prototype is running a version of Moa. Moa [BWK98] is an extensible NF^2, or XNF^2, logical algebra. Moa is partly based on the work presented in [Ste95]. One of the crucial characteristics of Moa besides its extensibility, is its special handling of nested structures.

Typically, querying nested relations requires a specialized query language that either does one of the following to handle operations on the nested data:

- unnest the structure, perform the operation and nest the result again,

or:

- push the operation through the nesting and execute it at the level of the nested data.

The first methods appears to be very difficult, since nest and unnest operations are not each others inverse [Ste95]. This problem manifests itself when dealing with empty sets. When a set containing sets is unnested, any nested empty sets cannot be distinguished anymore. Nesting this data again then results in loss of these originally nested empty sets.

The second method requires special versions of each operator that has to deal with nested data. In most cases this results in depth-first iteration over all separate nested items. This is called nested-loop query processing, which is known to be relatively slow.

Moa takes a hybrid approach. It keeps explicit structure information stored, next to the data, which is stored in a flat manner. The explicit structure administration solves the unnest-nest problem since empty sets cannot disappear. The flat storage of the actual data allows set-based processing, thus avoiding inefficient nested-loop processing.

Each structure in Moa, such as a `SET`, `TUPLE`, or `LIST`, is implemented as a

separate extension module—textrm to the kernel which only has a notion of an abstract structure. Each structure extension defines some basic concepts: logical structure, logical operators, a mapping from logical structure on physical tables, and an implementation of each logical operator in terms of physical operators.

A query is defined in terms of logical operators on logical structures. For this query a physical plan is generated by using the physical implementation of each involved logical operator as defined in the extension module. This physical plan acts on the physical tables. In the current working prototype the Monet main-memory DBMS is used to implement this physical part. Next, the logical operators are interpreted to transform the input logical structure into the logical structure of the result. The physical plan is executed in Monet, which delivers the resulting data. Due to the special nature of the mapping from logical structure on physical tables and the implementation of each logical operator in terms of physical operators the resulting flat data that is produced in Monet precisely corresponds to the structure resulting at the logical level.

The miЯRor, Moa, and Monet research initiated many other research initiatives related to the integration of content management in a DBMS context. Examples are [WSK99, ZA00, BWZ+01] and also this dissertation.

2.4.2 Retrieval model support in databases

In this section we give an overview of approaches known in literature concerning the integration of IR technology in a DBMS. As we already mentioned in Section 2.3, IR techniques can be classified as boolean/exact, vector-space, probabilistic, or AI. We structured our overview accordingly. Since, we did not find any literature about integration of the AI approach in a DBMS that was of any interest to this dissertation we left it out of our summarization of approaches below.

From the systems we described in the previous section, [SP82, DKA+86, PSS+87, BRG88, JN95, GBS99] implement either boolean search or exact match retrieval. For the commercial DBMS Informix [Inf] a boolean search datablade is available, called Excalibur [Exc99]. Also, many modern day web search engines still use boolean techniques.

We found only one publication where explicitly the integration of the vector space retrieval model in a DBMS was mentioned [CG96a] though it is likely that some more work on the integration of this model in a database context exists.

Like for boolean retrieval, many work has been done on the integration of the probabilistic retrieval model in a DBMS [Sch93, GHF94, Fuh96, Son96,

VCC96, FR97, GFHR97, GF98, VW99, VH99, FGCF00]. The system described in [VCC96] is based on the INQUERY retrieval system we mentioned before. Note that the miRRor system [VH99, FGCF00] actually supports any retrieval model, but for the prototype the state-of-the-art probabilistic retrieval model of [Hie01] was used.

2.4.3 Integration methods

In the previous section we have seen a lot of approaches classified by the retrieval models they address or support. This support can be embedded in an actual system in several ways. In this section we look at the approaches in literature from the perspective of how the IR technology was integrated in the database context. Where possible we refer to the systems in the previous section.

IR functionality can be combined with database functionality in several ways.

Coupled system

In a coupled system a dedicated IR system and a DBMS are linked at the application level. This is one of the options proposed in [VCC96], where the authors used an off-the-shelf DBMS and the INQUERY retrieval system. This is an easily realizable approach, but leaves the application with the task of combining results.

Exploit query language

In this case the whole retrieval algorithm is implemented in terms of the database query language, for instance SQL. This means that existing elements of a query language are used to express the entire ranking algorithm. This, of course, results in huge query expressions which are difficult to handle. Typically, this approach is only sensible when queries are embedded in an application. The query expressions are much too complex to be typed in directly by a user sitting behind a terminal. Like the previous approach, this one is easy to realize. Examples in literature can be found in [GHF94, GFHR97, GF98, FGCF00].

Entire IR system via single operator

The IR system is re-implemented as extension module—(, or a wrapper operator is implemented in an extension module that transfers its work to a

dedicated external IR system. The latter approach can be seen as a variant of the first approach of coupling an IR system with a DBMS. Both variants are mentioned in [VCC96]. The commercial Excalibur text datablade [Exc99] for the Informix DBMS [Inf] is a clear example of a complete system implemented in an extension module.

Extension of query language with IR operators

This case, modifying the database query language, is comparable to the previous case in some sense. However, the distinction with the previous approach is, that here not an entire IR system is captured behind one operator. Instead several operators are added, for instance via an extension mechanism. The operators perform, what could be seen as, an atomic operation in the retrieval process. In most cases they form a set of probabilistic operators that together provide the required ranking functionality of a probabilistic IR system.

In [Fuh96] and [FR97] a probabilistic extension of the relational model is proposed. This would allow the DBMS to deal with uncertainty in a natural manner. Probabilistic IR is one of the application areas demonstrated by the authors. In [VCC96], besides the other approaches also covered in that paper, an extension is proposed comprising several operators. Each operator has a corresponding counterpart in the inference network model on which INQUERY is based. The inference network is in turn based on Bayesian networks which are a special class of probabilistic reasoning systems.

A special case is the work presented in [Son96]. There an object-oriented system is proposed to store and retrieve documents resembling the NF^2 approach used by AIM, as we mentioned before. Here the typical extensible nature of the OO system is used to model both the structure of the documents in the collection as well as the operators to retrieve the information.

Full integration and other approaches

Next to the approaches sketched above, many others exist. Here we list some of these. Most of these approaches differ from the previous ones because these were designed from the start with multimedia or text retrieval as application domain in mind. The previous four cases all proposed the use of existing systems in some sense, either by using them without any modification or extended in some way with additional functionality. The approaches we describe below often involve a completely new system.

The NF^2 model was already there before [SP82], [DKA$^+$86], and [PSS$^+$87]. But AIM and the Darmstadt Database Kernel System were developed with the idea of mixing content data and structured data in mind. Earlier on in

this section, we gave an example of how the NF^2 capabilities can be exploited to model text collections in quite natural manner as nested sets. Queries are posed in a special query language that is able to deal with the nesting. Boolean retrieval boils down to selecting all those documents, i.e., sets of words, for which the semi join with the query, a unary keyword relation, is not empty.

Though based on a different data model, MULTOS also was designed with this new application area in mind.

In [Bur92] basic principles from database technology, like the relational model and the requirement of data-independence, are borrowed and combined with IR technology to present a new approach. Though this approach does not really fit in the profile of a DBMS with extended capabilities for IR, it does present an interesting setup, constructed from scratch.

The SPIDER system [Sch93] and the work on what is called proximal nodes in [NBY97] distinguish themselves by their own query language. The higher order database concept presented in [GBS99] and related work, though using off-the-shelf DBMS technology, differs from the rest by its special distribution and transaction management ideas. They use additional management tools to take care of this, which could be seen as special middleware.

Finally, the miЯRor approach [VW99, Vri99] proposes integration of the IR functionality at all three layers of the typical DBMS architecture: conceptual, logical, and physical. The argument is that by doing so, many performance bottlenecks are avoided. To support this, the miЯRor approach requires all three layers to be extensible, adding those parts of the IR support to the level where it performs best while retaining data independence at the logical and higher levels. This division of IR processing parts over the three architectural layers to improve efficiency is called the *multi-model* approach in [Vri99].

To support IR, the DOCREP structure, which models a text document, has been added to Moa as an extension module—). This allows for a semantically richer modeling of document collections, which has several advantages. First of all, it makes the model more intuitive and readable for a human. Secondly, it allows more efficient query processing, since it provides more information which can be exploited during query optimization. For instance, the example NF^2 model of a document collection we gave above can now be modeled as follows:

```
DOCCOLL: SET<
           DOCREP<
             DOCID:  STRING,
             AUTHOR: STRING,
             TEXT:   SET<
                       STRING
                     >
```

```
        >
      >
```

The DOCREP structure has a special score operator that can compute the $tf \cdot idf$ score of a particular document for a given query consisting of keywords. The DOCREP maintains internally, at the Monet level, tf and idf statistics to allow efficient computation of this score. A typical IR ranking query in Moa, with the set of search terms modeled as a unary relation Q, would look like the following:

```
TOP10 := sublist[1,10](
           sortdesc[THIS.S](
             set2list(
               map[TUPLE<THIS.DOCID, S: score(THIS,Q)>](
                 DOCCOLL
               )
             )
           )
         )
```

This query does the following. The map operator generates a TUPLE for each element of the set DOCCOLL. Each of these new tuples contains the original DOCID of the DOCREP element, referred to by the special variable THIS, and a score S computed with the score function. The set2list operator converts this new set to a LIST. On LIST two operators are defined, among others: sortdesc and sublist. The sortdesc operator orders the LIST descending on the attribute S of the contained tuples. The THIS variable again acts as a special variable denoting the elements of the LIST. Finally, the sublist operator performs positional selection of the first 10 elements.

In line with the multi-model approach, also on the Monet level extra operators were defined to support efficient computation of the actual $tf \cdot idf$ scores.

During query execution, the score operator of DOCREP in Moa is translated into these physical operators at the MIL level. Then these operators, in combination with existing operators in MIL produce a BAT with the scores. The sortdesc operator in Moa translates to a sort operation on this BAT at the MIL level. The sublist in Moa results in a slice operator, which is the positional select operator in MIL, producing the BAT corresponding to the TOP10 result structure, which represents a LIST in Moa.

2.4.4 IR optimization support in databases

Compared to the large amount of work done on the integration of IR in databases only little has been done in the field of providing enhanced query optimization techniques to support these new query processing facilities. In [Son96] the authors even explicitly mention that, now that they have integrated the IR functionality in their DBMS, the built-in query optimizer takes care of the query optimization, free of any additional effort.

Actually, only in [BRG88], the paper about MULTOS, and in [CG96a] query optimization in light of queries which combine structured and content data is really addressed. Both use a costs model to decide how to execute the query. Additionally, in [CG96a] optimization techniques proposed in [Fag98] to optimize the combination of several rankings is exploited.

In [FR97] the authors suggest in their conclusions that the computations of the probabilistic extensions to the relation model as proposed by them, might require to take CPU costs into account during query optimization. Note that in many DBMSs the cost model only looks at disk IO costs since those costs typically dominate CPU costs in the case of disk-based processing.

Finally, incorporation of retrieval effectiveness, or answer quality, such as recall and precision, in the optimization process is very rare. In the VODAK object-oriented DBMS [BMN94] the retrieval effectiveness is added as extra parameter to the cost model for the retrieval of SGML documents. This allows the optimizer to handle the trade-off between speed and quality. In [WB00] degrading the quality of the answers was also considered as possible option to gain speed. However, this idea concerned nearest-neighbor search and was evaluated for image retrieval only with a bad quality metric.

2.5 Concluding remarks

In this chapter we have discussed three areas: database technology, IR technology, and the area where these two come together. We mainly focussed on issues concerning the integration of IR functionality and optimization techniques in a DBMS, which we chose as our context. In the remaining chapters we refer to the work described here were necessary.

References

[ACM86] *Proceedings of the 1986 ACM SIGMOD International Conference on the Management of Data*, ACM Press, June 1986.

[ACM96] *Proceedings of the 1996 ACM SIGMOD International Conference on the Management of Data*, ACM Press, 1996.

[ACM97] *Proceedings of the 1997 ACM SIGMOD International Conference on the Management of Data*, ACM Press, 1997.

[ACM98a] *Proceedings of the 1998 ACM SIGMOD International Conference on the Management of Data*, ACM Press, 1998.

[ACM98b] *Proceedings of the 1998 ACM SIGMOD International Conference on Principles of Database Systems (PODS'98)*, ACM Press, 1998.

[AOV+99] M.P. Atkinson, M.E. Orlowska, P. Valduriez, S.B. Zdonik, and M.L. Brodie (eds.), *Proceedings of the 25th VLDB Conference*, VLDB, Morgan Kaufmann, September 1999.

[BA94] K. Böhm and K. Aberer, *Storing HyTime Documents in an Object-Oriented Database*, In Nicholas and Mayfield [NM94], pp. 26–33.

[BCC94] J. Broglio, J.P. Callan, and W.B. Croft, *INQUERY System Overview*, TIPSTER Text Program (Phase I) (San Francisco, CA), Morgan Kaufmann, 1994, pp. 47–67.

[BH96] C.A. van den Berg and van der A. Hoeven, *Monet meets OO7*, Proceedings of OODS'96, January 1996.

[BK95] P.A. Boncz and M.L. Kersten, *Monet: An Impressionist Sketch of an Advanced Database System*, Basque International Workshop on Information Technology, Data Management Systems (BIWIT'95), IEEE Computer Society Press, July 1995.

[BK99] P.A. Boncz and M.L. Kersten, *MIL Primitives For Querying A Fragmented World*, VLDB Journal **8** (1999), no. 2.

[BL85] C. Buckley and A.F. Lewit, *Optimisation of Inverted Vector Searches*, SIGIR '85: Proceedings of the 8th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM SIGIR, ACM, August 1985, pp. 97–110.

[BMK99] P.A. Boncz, S. Manegold, and M.L. Kersten, *Database Architecture Optimized for the new Bottleneck: Memory Access*, In Atkinson et al. [AOV+99].

[BMN94] K. Böhm, A. Müller, and E. Neuhold, *Structured Document Handling - a Case for Integrating Databases and Information Retrieval*, In Nicholas and Mayfield [NM94], pp. 147–154.

[BRG88] E. Bertino, F. Rabitti, and S. Gibbs, *Query Processing in a Multimedia Document System*, ACM Transactions on Office Information Systems **6** (1988), no. 1, 1–41.

[Bro95] E.W. Brown, *Execution Performance Issues in Full-Text Information Retrieval*, Ph.D.T./Technical Report 95-81, University of Massachusetts, Amherst, October 1995.

[Bur92] F.J. Burkowski, *Retrieval Activities in a Database Consisting of Heterogeneous Collections of Structured Text*, SIGIR '92: Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM SIGIR, ACM, June 1992, pp. 112–125.

[BVB99] H.E. Blok, A.P. de Vries, and H.M. Blanken, *Top N MM query optimization: The best of both IR and DB worlds*, Conferentie Informatiewetenschap 1999 [Eng.: Information Science Conference 1999] (CIW'99) (Amsterdam) (P. De Bra and L. Hardman, eds.), Werkgemeenschap Informatiewetenschap, November 1999.

[BWK98] P.A. Boncz, A.N. Wilschut, and M.L. Kersten, *Flattening an Object Algebra to Provide Performance*, Proceedings of the 14th International Conference on Data Engineering (ICDE'98), IEEE Transactions on Knowledge and Data Engineering, IEEE Computer Society, February 1998.

[BWZ+01] H.E. Blok, M. Windhouwer, R. van Zwol, M. Petkovic, P.M.G. Apers, M.L. Kersten, and W. Jonker, *Flexible and scalable digital library search*, Proceedings of the 27th VLDB Conference, VLDB, September 2001.

[Car75] A.F. Cardenas, *Analysis and performance of inverted data base structures*, Communications of the ACM **18** (1975), no. 5, 253–263.

[CCH92] J.P. Callan, W.B. Croft, and S.M. Harding, *The INQUERY Retrieval System*, 3rd International Conference on Database and Expert Systems Applications (DEXA'92) (A.M. Tjoa and I. Ramos, eds.), 1992, pp. 78–83.

[CG96a] S. Chaudhuri and L. Gravano, *Optimizing Queries over Multimedia Repositories*, IEEE Data Engineering Bulletin **19** (1996), no. 4, 45–52, Also see [CG96b].

[CG96b] S. Chaudhuri and L. Gravano, *Optimizing Queries over Multimedia Repositories*, In *Proceedings of the 1996 ACM SIGMOD*

International Conference on the Management of Data [ACM96], pp. 91–102.

[CG99] S. Chaudhuri and L. Gravano, *Evaluating Top-k Selection Queries*, In Atkinson et al. [AOV+99], pp. 397–410.

[Cha98] S. Chaudhuri, *An Overview of Query Optimization in Relational Systems*, In *Proceedings of the 1998 ACM SIGMOD International Conference on Principles of Database Systems (PODS'98)* [ACM98b], pp. 34–42.

[CHS99] F. Chu, J.Y. Halpern, and P. Seshadri, *Least Expected Cost Query Optimization: An Exercise in Utility*, Proceedings of the 1999 ACM SIGMOD International Conference on Principles of Database Systems (PODS'99), ACM Press, 1999, pp. 138–147.

[ÇJF+98] U. Çetintemel, B.Th. Jónsson, M.J. Franklin, C.L. Giles, and D. Srivastava, *Evaluating Answer Quality/Efficiency Tradeoffs*, Proceedings of the 5th International Workshop on Knowledge Represenation Meets Databases (KRDB '98): Innovative Application Programming and Query Interfaces (A. Borgida, V.K. Chaudhri, and M. Staudt, eds.), CEUR Workshop Proceedings, vol. 10, May 1998, pp. 19.1–19.4.

[CK97a] M.J. Carey and D. Kossmann, *On Saying "Enough Already!" in SQL*, In *Proceedings of the 1997 ACM SIGMOD International Conference on the Management of Data* [ACM97], pp. 219–230.

[CK97b] M.J. Carey and D. Kossmann, *Processing Top and Bottom N Queries*, IEEE Bulletin of the Technical Commitee on Data Engineering **20** (1997), no. 3, 12–19.

[CK98] M.J. Carey and D. Kossmann, *Reducing the Braking Distance of an SQL Query Engine*, In Gupta et al. [GSW98], pp. 158–169.

[CMN98] S. Chaudhuri, R. Motwani, and V. Narasayya, *Random Sampling for Histogram Construction: How much is enough?*, In *Proceedings of the 1998 ACM SIGMOD International Conference on the Management of Data* [ACM98a], pp. 436–447.

[CMN99] S. Chaudhuri, R. Motwani, and V. Narasayya, *On Random Sampling over Joins*, Proceedings of the 1999 ACM SIGMOD International Conference on the Management of Data, ACM Press, 1999, pp. 263–274.

[CP86] S. Ceri and G. Pelagatti, *Distributed Databases: Principles and Systems*, McGraw-Hill computer science series, International Student Editions, McGraw-Hill, Inc., 1986.

[CP97] D.R. Cutting and J.O. Pedersen, *Space Optimizations for Total Ranking*, Proceedings of RAIO'97, Computer-Assisted Information Searching on Internet, June 1997, pp. 401–412.

[CR94] C.M. Chen and N. Roussopoulos, *Adaptive Selectivity Estimation Using Query Feedback*, Proceedings of the 1994 ACM SIGMOD International Conference on the Management of Data, ACM Press, May 1994, pp. 161–172.

[Dat95] C.J. Date, *An Introduction to Database Systems*, 6th. ed., 1995, ISBN 0-201-54329-X.

[DKA+86] P. Dadam, K. Kuespert, F. Andersen, H.M. Blanken, R. Erbe, J. Guenauer, V. Lum, P. Pistor, and G. Walch, *A DBMS Prototype to Support NF² Relations: An Integrated View on Flat Tables and Hierarchies*, In *Proceedings of the 1986 ACM SIGMOD International Conference on the Management of Data* [ACM86], pp. 356–367.

[DR99a] D. Donjerkovic and R. Ramakrishnan, *Probabilistic Optimization of Top N Queries*, In Atkinson et al. [AOV+99], pp. 411–422.

[DR99b] D. Donjerkovic and R. Ramakrishnan, *Probabilistic Optimization of Top N Queries*, Technical Report CR-TR-99-1395, Department of Computer Sciences, University of Wisconsin-Madison, 1999.

[Exc99] *Excalibur Text Search Datablade*, Informix Tech Brief, 1999, See also [Inf].

[Fag98] R. Fagin, *Fuzzy Queries in Multimedia Database Systems*, In *Proceedings of the 1998 ACM SIGMOD International Conference on Principles of Database Systems (PODS'98)* [ACM98b], pp. 1–10.

[Fag99] R. Fagin, *Combining fuzzy information from multiple systems*, Journal on Computer and System Sciences **58** (1999), no. 1, 83–99, Special issue for selected papers from the 1996 ACM SIGMOD PODS Conference.

[FGCF00] O. Frieder, D.A. Grossman, A. Chowdhury, and G. Frieder, *Efficiency Considerations for Scalable Information Retrieval Servers*, Journal of Digital Information **1** (2000), no. 5.

[FM00] R. Fagin and Y.S. Maarek, *Allowing users to weight search terms*, Proceedings of RAIO'00, Computer-Assisted Information Searching on Internet, 2000, pp. 682–700.

[FR97] N. Fuhr and T. Rölleke, *A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems*, ACM Transactions on Information Systems **15** (1997), no. 1, 32–66.

[FSGM+98] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J.D. Ullman, *Computing Iceberg Queries Efficiently*, In Gupta et al. [GSW98], pp. 299–310.

[Fuh96] N. Fuhr, *Models for Integrated Information Retrieval and Database Systems*, IEEE Data Engineering Bulletin **19** (1996), no. 1, 3–13.

[Gan98] S. Ganguly, *Design and Analysis of Parametric Query Optimization Algorithms*, In Gupta et al. [GSW98], pp. 228–238.

[GBS99] T. Grabs, K. Böhm, and H.-J. Schek, *A Document Engine on a DB Cluster*, High Performance Transaction Processing Systems Workshop (Asilomar, California, USA), September 1999.

[GF98] D.A. Grossman and O. Frieder, *Information retrieval: algorithms and heuristics*, The Kluwer international series in engineering and computer science, Kluwer Academic, Boston, 1998, ISBN 0-7923-8271-4.

[GFHR97] D.A. Grossman, O. Frieder, D.O. Holmes, and D.C. Roberts, *Integrating Structured Data and Text: A Relational Approach*, Journal of the American Society of Information Science **48** (1997), no. 2, 122–132.

[GGMS96] S. Ganguly, P.B. Gibbons, Y. Matias, and A. Silberschatz, *Bifocal Sampling for Skew-Resistant Join Size Estimation*, In *Proceedings of the 1996 ACM SIGMOD International Conference on the Management of Data* [ACM96], pp. 271–281.

[GHF94] D.A. Grossman, D.O. Holmes, and O. Frieder, *A Parallel DBMS Approach to IR in TREC-3*, In *Proceedings of the Third Text Retrieval Conference (TREC-3)* [TRE94], pp. 279–288.

[Gra93] G. Graefe, *Query Evaluation Techniques for Large Databases*, ACM Computing Surveys **25** (1993), no. 2, 73–170.

[GSW98] A. Gupta, O. Shmueli, and J. Widom (eds.), *Proceedings of the 24th VLDB Conference*, VLDB, Morgan Kaufmann, 1998.

[Har94] D.K. Harman, *Evaluation Techniques and Measures*, In *Proceedings of the Third Text Retrieval Conference (TREC-3)* [TRE94], pp. A5–A13.

[HCL+90] L.M. Haas, W. Chang, G.M. Lohman, J. McPherson, P.F. Wilms, G. Lapis, B. Lindsay, H. Pirahesh, M.J. Carey, and E. Shekita, *Starburst Mid-Flight: As the Dust Clears*, IEEE Transactions on Knowledge and Data Engineering **2** (1990), no. 1, 143–160.

[Hel98] J.M. Hellerstein, *Optimization Techniques for Queries with Expensive Methods*, ACM Transactions on Database Systems **23** (1998), no. 2, 113–157.

[Hie98] D. Hiemstra, *A Linguistically Motivated Probabilistic Model of Information Retrieval*, Proceeding of the second European Conference on Research and Advanced Technology for Digital Libraries (ECDL'98) (C. Nicolaou and C. Stephanidis, eds.), Springer-Verlag, 1998, pp. 569–584.

[Hie00] D. Hiemstra, *A probabilistic justification for using $tf \cdot idf$ term weighting in information retrieval*, International Journal on Digital Libraries **3** (2000), no. 2, 131–139.

[Hie01] D. Hiemstra, *Using language models for information retrieval*, Ph.D. thesis, University of Twente, Enschede, The Netherlands, January 2001.

[HS93] J.M. Hellerstein and M. Stonebreaker, *Predicate Migration: Optimizing Queries with Expensive Predicates*, Proceedings of the 1993 ACM SIGMOD International Conference on the Management of Data, ACM Press, May 1993, pp. 267–276.

[IB86] A. IJbema and H.M. Blanken, *Estimating bucket accesses: A practical approach*, Proceedings of the Second International Conference on Data Engineering (ICDE'86), IEEE Computer Society, IEEE, February 1986, pp. 30–37.

[IC91] Y.E. Ioannidis and S. Christodoulakis, *On the Propagation of Errors in the Size of Join Results*, Proceedings of the 1991 ACM SIGMOD International Conference on the Management of Data (J. Clifford and R. King, eds.), ACM Press, June 1991, pp. 268–277.

[Inf] *Informix Corporation*, URL: http://www.informix.com/.

[Ioa96] Y.E. Ioannidis, *Query Optimization*, The Computer Science and Engineering Handbook, CRC Press, Boca Raton, Florida, 1996, ISBN 0-8493-2909-4, ch. 45, pp. 1038–1057.

[IP95] Y.E. Ioannidis and V. Poosala, *Balancing Histogram Optimality and Practicality for Query Result Size Estimation*, Proceedings of the 1995 ACM SIGMOD International Conference on the Management of Data, ACM Press, 1995, pp. 233–244.

[JFS98] B.Th. Jónsson, M.J. Franklin, and D. Srivastava, *Interaction of Query evaluation and Buffer management for IR*, In *Proceedings of the 1998 ACM SIGMOD International Conference on the Management of Data* [ACM98a].

[JK84] M. Jarke and J. Koch, *Query Optimization in Database Systems*, ACM Computing Surveys **16** (1984), no. 2, 111–152.

[JN95] K. Järvelin and T. Niemi, *An NF^2 Relational Interface for Document Retrieval, Restructuring and Aggregation*, SIGIR '95: Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM SIGIR, ACM, June 1995, pp. 102–110.

[KN98] M.L. Kersten and N.J. Nes, *Fitness Joins in the Ballroom*, Cwi report, CWI, Amsterdam, The Netherlands, July 1998.

[LMS94] A.Y. Levy, I.S. Mumick, and Y. Sagiv, *Query Optimization by Predicate Move-Around*, Proceedings of the 20th VLDB Conference, VLDB, 1994.

[LNS90] R.J. Lipton, J.F. Naughton, and D.A. Schneider, *Practical Selectivity Estimation through Adaptive Sampling*, Proceedings of the 1990 ACM SIGMOD International Conference on the Management of Data (H. Garcia-Molina and H.V. Jagadish, eds.), ACM Press, June 1990, pp. 1–11.

[Lyn88] C. Lynch, *Selectivity estimation and query optimization in large databases with highly skewed distributions of column values*, Proceedings of 14th VLDB Conference, August 1988, pp. 240–251.

[Mut85] B. Muthuswamy, *A Detailed Statistical Model for Relational Query Optimization*, Proceedings of the 1985 ACM SIGMOD International Conference on the Management of Data, ACM Press, June 1985, pp. 439–448.

[NBY97] G. Navarro and R. Baeza-Yates, *Proximal Nodes: A Model to Query Document Databases by Content and Structure*, ACM Transactions on Information Systems **15** (1997), no. 4, 400–435.

[NM94] C.K. Nicholas and J. Mayfield (eds.), *Third International Conference on Information and Knowlegde Management (CIKM'94)*, ACM SIGIR/SIGMIS, ACM Press, November 1994.

[PC98] J.M. Ponte and W.B. Croft, *A Language Modeling Approach to Information Retrieval*, SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM SIGIR, ACM, August 1998, pp. 275–281.

[Per84] M. Persin, *Document filtering for fast ranking*, SIGIR '84: Proceedings of the 7th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM SIGIR, ACM, August 1984, pp. 339–349.

[PIHS96] V. Poosala, Y.E. Ioannidis, P.J. Haas, and E.J. Shekita, *Improved Histograms for Selectivity Estimation of Range Predicates*, In *Proceedings of the 1996 ACM SIGMOD International Conference on the Management of Data* [ACM96], pp. 294–305.

[PSS+87] H.-B. Paul, H.-J. Schek, M.H. Scholl, G. Weikum, and U. Deppisch, *Architecture and Implementation of the Darmstadt Database Kernel System*, Proceedings of the 1987 ACM SIGMOD International Conference on the Management of Data, ACM Press, June 1987, pp. 196–207.

[RDR+98] R. Ramakrishnan, D. Donjerkovic, A. Ranganathan, K.S. Beyer, and M. Krishnaprasad, *SRQL: Sorted Relational Query Language*, Proceedings of SSDBMS'98 (Piscataway, NJ), IEEE, IEEE, July 1998.

[RH96] E. Riloff and L. Hollaar, *Text Databases and Information Retrieval*, The Computer Science and Engineering Handbook, CRC Press, Boca Raton, Florida, 1996, ISBN 0-8493-2909-4, ch. 50, pp. 1125–1141.

[Rij79] C.J. van Rijsbergen, *Information Retrieval*, 2nd. ed., Butterworths, London, 1979.

[Roc71] J.J. Rocchio, *Relevance feedback in information retrieval*, The SMART retrieval system: Experiments in automatic document processing (G. Salton, ed.), Series in automatic computation, Prentice-Hall, 1971, ISBN 0-13-814525-3, pp. 313–323.

[SB88] G. Salton and C. Buckley, *Term-Weighting Approaches in Automatic Text Retrieval*, Information Processing & Management **24** (1988), no. 5, 513–523.

[Sch93] P. Schäuble, *SPIDER: A Multiuser Information Retrieval System for Semistructured and Dynamic Data*, SIGIR '93: Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM SIGIR, ACM, June 1993, pp. 318–327.

[Ses98] P. Seshradri, *Enhanced abstract data types in object-relational databases*, The VLDB Journal **7** (1998), no. 3, 130–140.

[SLR96] P. Seshadri, M. Livny, and R. Ramakrishnan, *E-ADTs: Turbo-Charging Complex Data*, IEEE Data Engineering Bulletin **19** (1996), no. 4, 11–18.

[SLR97] P. Seshadri, M. Livny, and R. Ramakrishnan, *The Case for Enhanced Abstract Data Types*, Proceedings of the 23th VLDB Conference, VLDB, 1997, pp. 66–75.

[Son96] G. Sonnenberg, *Exploiting the Functionality of Object-Oriented Database Management Systems for Information Retrieval*, IEEE Data Engineering Bulletin **19** (1996), no. 1, 14–23.

[SP82] H.-J. Schek and P. Pistor, *Data Structures for an Integrated Data Base Management and Information Retrieval System*, Proceedings of 8th VLDB Conference, September 1982, pp. 197–207.

[SP97] P. Seshradri and M. Paskin, *PREDATOR: An OR-DBMS with Enhanced Data Types*, In *Proceedings of the 1997 ACM SIGMOD International Conference on the Management of Data* [ACM97], pp. 568–571.

[SR86] M. Stonebraker and L.A. Rowe, *The Design of Postgres*, In *Proceedings of the 1986 ACM SIGMOD International Conference on the Management of Data* [ACM86], pp. 340–355.

[SSM96] D. Simmen, E. Shekita, and T. Malkemus, *Fundamental Techniques for Order Optimization*, In *Proceedings of the 1996 ACM SIGMOD International Conference on the Management of Data* [ACM96], pp. 57–67.

[Ste95] H. Steenhagen, *Optimization of Object Query Languages*, Ph.D. thesis, University of Twente, Enschede, The Netherlands, October 1995.

[TC90] H. Turtle and W.B. Croft, *Inference Networks for Document Retrieval*, Computer and Information Science Technical Report 90-07, University of Massachusetts, Amherst, Massachusetts, February 1990.

[TRE94] *Proceedings of the Third Text Retrieval Conference (TREC-3)*, NIST Special publications, Gaithersburg, Maryland, November 1994.

[Ull88] J.D. Ullman, *Principles of database and knowledgebase systems*, vol. 2, Principles of computer science series, no. 14, Computer Science Press, Rockville, Maryland, 1988, ISBN 0-7167-8162-X.

[VB98a] A.P. de Vries and H.M. Blanken, *Database technology and the management of multimedia data in miRRor*, Proceedings of SPIE, Multimedia Storage and Archiving Systems III, vol. 3527, November 1998.

[VB98b] A.P. de Vries and H.M. Blanken, *The Relationship between IR and Multimedia Databases*, IRSG'98, the 20th BCS Colloquium on Information Retrieval (M.D. Dunlop, ed.), 1998.

[VCC96] S.R. Vasanthakumar, J.P. Callan, and W.B. Croft, *Integrating INQUERY with an RDBMS to Support Text Retrieval*, IEEE Data Engineering Bulletin **19** (1996), no. 1, 24–33.

[VDBA99] A.P. de Vries, M.G.L.M. van Doorn, H.M. Blanken, and P.M.G. Apers, *The miRRor MMDBMS Architecture*, In Atkinson et al. [AOV+99], pp. 758–761.

[VH99] A.P. de Vries and D. Hiemstra, *The miRRor DBMS at TREC*, Proceedings of the Eigth Text Retrieval Conference (TREC-8) (Gaithersburg, Maryland), NIST Special publications, November 1999, pp. 725–734.

[Vri98] A.P. de Vries, *MiRRor: Multimedia Query Processing in Extensible Databases*, 14th Twente Workshop on Language Technology, Language Technology in Multimedia Information Retrieval (TWLT 14) (Enschede, The Netherlands), University of Twente, December 1998, pp. 37–47.

[Vri99] A.P. de Vries, *Content and Multimedia Database Management Systems*, Ph.D. thesis, University of Twente, Enschede, The Netherlands, December 1999.

[VW99] A.P. de Vries and A.N. Wilschut, *On the Integration of IR and Databases*, 8th IFIP 2.6 Working Conference on Data Semantics 8, January 1999, pp. 16–31.

[WB00] R. Weber and K. Böhm, *Trading Quality for Time with Nearest-Neighbor Search*, Proceedings of the 7th Conference on Extending Database Technology (EDBT 2000), Lecture Notes in Computer Science, Springer, March 2000.

[WSK99] M.A. Windhouwer, A.R. Schmidt, and M.L. Kersten, *Acoi: A System for Indexing Multimedia Objects*, International Workshop on Information Integration and Web-based Applications & Services (Yogyakarta, Indonesia), November 1999.

[Yao77a] S.B. Yao, *An Attribute Based Model for Database Access Cost Analysis*, ACM Transactions on Database Systems **3** (1977), no. 1, 45–67.

[Yao77b] S.B. Yao, *Approximating Block Accesses in Database Organizations*, Communications of the ACM **20** (1977), no. 4, 260–261.

[Yao79] S.B. Yao, *Optimization of Query Evaluation Algorithms*, Proceedings of the 1979 ACM SIGMOD International Conference on the Management of Data, ACM Press, June 1979, pp. 133–155.

[ZA00] R. van Zwol and P.M.G. Apers, *The webspace method: On the integration of database technology with information retrieval*, In proceedings of CIKM'00 (Washington, DC.), November 2000.

[Zip49] G.K. Zipf, *Human Behavior and the Principle of Least Effort*, Addison-Wesley, Reading, MA, USA, 1949.

Chapter 3

Problem: analysis & approach

3.1 Introduction

In this chapter we take a closer look at the three research questions, we stated in Chapter 1. In Section 3.2 we review these questions given the state-of-the-art we sketched in the previous chapter. Next, in Section 3.3, we describe our approach. There we also have a closer look at the experimental system needed to support the training and validation of the models, which we intend to construct in the next three chapters, to answer the research questions.

3.2 Research questions revisited

As we saw in the previous chapter, fragmentation is a commonly accepted means to divide data in a database in portions. Since the third research question concerns a better understanding of horizontal fragmentation in the context of this dissertation, we handle this question first, as we already announced in Chapter 1. So, we start in Section 3.2.1, to analyze the related work for the third research question: *'Can horizontal fragmentation facilitate the trading of quality for speed to support set-based IR top-N query optimization?'*. In this section we also present the first principles of our approach on how we setup the horizontal fragmentation in a database that supports probabilistic IR. Given the considerations with respect to fragmentation we take a closer look at the first two research questions in Sections 3.2.2 and 3.2.3.

3.2.1 Fragmentation & set-based IR top-N query optimization

Recall the third research question, as presented in Chapter 1:

Q3	*Can horizontal fragmentation facilitate the trading of quality for speed to support set-based IR top-N query optimization?*

As we saw in Section 2.2.3 in the previous chapter, fragmentation is a commonly accepted technique in the database field to divide data into portions. In most cases it is used when distributing data. We think that horizontal fragmentation, as a partitioning of the set of tuples in a relation, also might serve another purpose when dealing with IR top-N query optimization in a DBMS. But before we can describe how we envision this additional purpose, we have to describe our approach to model state-of-the-art IR in a relational DBMS. After we have defined the model we can talk about what exactly we want to fragment.

As we already mentioned in the previous chapters, boolean keyword search is not considered state-of-the-art anymore. We choose to use probabilistic IR techniques, in particular we use a $tf \cdot idf$ model derived from the state-of-the-art, well-founded probabilistic IR model presented in [Hie01].

Recall the key parameters in the $tf \cdot idf$ model:

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

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

inverse document frequency: For each term, idf is the inverse number of the df.

We assume the tf and df statistics to be present in the database at query time. Furthermore, we assume a query to be a set of keywords.

TFstats($term$, doc, tf)	(3.1)
DFstats($term$, df)	(3.2)
Q($term$)	(3.3)

Figure 3.1: Relations (schema)

In Figure 3.1 we have displayed a relational schema to capture this information. Figure 3.2 presents these relations in a graphical manner.

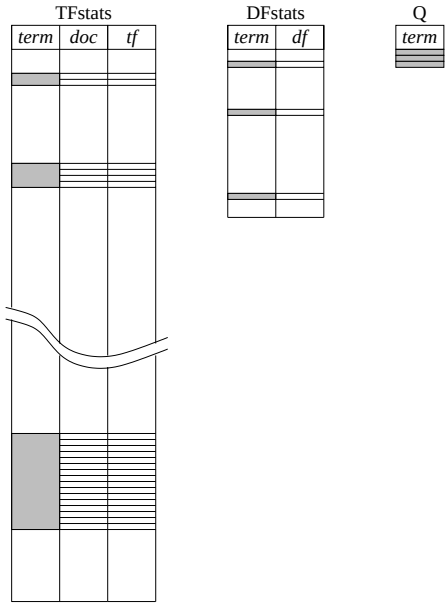

Figure 3.2: Relations (graphical representation)

The TFstats relation typically grows huge. To deliver a ranked list of documents for a query Q this relation is first restricted to the terms in the query using a semi join: $\text{TFstats}_{\text{Q}} = \text{TFstats} \ltimes \text{Q}$. The rest of the $tf \cdot idf$ ranking algorithm is centered around this restricted version of TFstats.

As we already mentioned in the previous chapter, query terms with a high *df* are considered a-priori less interesting, i.e., less discriminative, for the query result. This is also reflected in the score formula where the score contribution of a query term to the score of a document is inverse proportional to the *df* of the query term. Many IR top-N query optimization techniques exploit this knowledge by evaluating the query in order of ascending *df*. By computing upper and lower bounds of the already computed scores and growth possibilities given what is left, these algorithms can cut off the query evaluation without damaging the results much or even not at all. We refer to the previous chapter for a more elaborate overview of those techniques.

However, an IR algorithm in a dedicated IR system is in control of the data. In a DBMS the IR algorithm should operate in a set-at-a-time manner and behave according to the data independence principle. But when the data is modeled in relations as we presented above, the database way of processing would be to process the whole TFstats at once for all query terms in Q, i.e.,

performing the semi join between the entire TFstats and Q and then process the remainder of the algorithm for the entire resulting TFstats_Q.

One option to solve this would be to implement the entire IR ranking algorithm, including IR cut off techniques, at the physical level. But, as mentioned before, this conflicts with the multi-model approach and is prone to cause efficiency problems. Since, this is undesirable, we reject this option.

A slightly comparable approach would be to use database top-N optimization techniques. Unfortunately, these techniques are often too limited to be used directly. However, probabilistic top-N optimization techniques in the database field do resemble the idea we propose to guess a cut off. The difference with our approach is that those database techniques are designed to work on exact match queries on structured data for which it is possible to automatically detect whether the cut off was too restrictive or not. We have to deal with IR queries for which this proof of correctness for the answer is impossible. Therefore automatic adjustment of the cut off boundary is impossible in our case, too.

We propose to horizontally fragment our database relations according to the same principles as the IR optimization techniques iterate over the terms, to allow IR optimization techniques to exploit the fragmentation. To do so, we assume the DFstats relation to be ordered ascending on df. We now can fragment the DFstats horizontally. We can characterize each fragmentation by a sequence $F = [f_1, f_2, \ldots, f_{|F|}]$ of $|F|$ fractions $f_i = \frac{|\text{DFstats}_i|}{|\text{DFstats}|}$ for each fragment DFstats_i with $i \in \{1, 2, \ldots, |F|\}$. Fragment DFstats_1 contains the $f_1 \cdot |\text{DFstats}|$ first tuples of DFstats, DFstats_2 contains the next $f_2 \cdot |\text{DFstats}|$, DFstats_3 the next $f_3 \cdot |\text{DFstats}|$, and so on. The fragments are thus sub-sets of terms with increasing df. We can fragment the TFstats relation accordingly, constructing $|F|$ derived fragments via $\text{TFstats}_i = \text{TFstats} \ltimes \text{DFstats}_i$.

This allows computation of the query results on a per-fragment basis instead of on a per-term basis. Instead of computing $\text{TFstats}_\text{Q} = \text{TFstats} \ltimes \text{Q}$ we can compute $\text{TFstats}_{i\text{Q}} = \text{TFstats}_i \ltimes \text{Q}$ and execute the (main part of the) ranking algorithm per $\text{TFstats}_{i\text{Q}}$ instead of for the entire TFstats_Q. This provides us with the option to decide for which fragments $i \in \{1, 2, \ldots, |F|\}$ we want to do this, or not. In other words, derived horizontally fragmenting the TFstats relation would thus allow set-based optimization of IR top-N queries, either by generalizing the IR cut off techniques or by implementing a term-fragment index to efficiently skip irrelevant fragments.

Into how many fragments we split the relations can be varied, i.e., we can vary F. Factors like the granularity desired for the cut off appear as interesting parameters we would like to understand. Furthermore, horizontally fragmenting our data is most likely a rather time consuming operation. So, one would like

to perform this at database design time. This requires the need to make a good guess where to place the fragment boundaries. Too few, too large, fragments might might reduce the effectiveness of the IR cut off techniques. Too many, too small fragments, might result in too much administrative overhead for the DBMS. In the degenerated case the latter situation might turn into so-called nested-loop processing, which is considered as inefficient.

Of course, we need to find out whether this approach is interesting enough. In Chapter 4 we perform several experimental studies to see the effects of fragmentation on performance and their possible use in facilitating the porting of IR top-N optimization techniques to the database context and database top-N optimization techniques to the IR application area.

3.2.2 Estimating selectivity

In this section we take a closer look at the first research question:

Q1	*Can we estimate selectivity as a function of the used portion of the data?*

Let us presume that the horizontal fragmentation approach indeed provides good opportunities to improve performance. So, from now on we talk in terms of fragments instead of arbitrary portions of data.

Given that we use horizontal fragmentation we are of course interested in a model that can predict the savings in execution time when a certain fragment with certain properties is ignored during query evaluation. This is first of all interesting at query optimization time to determine the execution costs of the IR query part. Secondly, we can also use such a model to reason backwards to database design time when we have to decide into how many fragments we should split the data.

A lot of work has been done on the development of cost models in the database field. Even some work has been done for databases with integrated text retrieval capabilities, like for instance MULTOS. But none of those cost models were designed to accurately deal with probabilistic IR in a fragmented database. In IR research of course some complexity studies have been performed concerning basic IR algorithms. In an IR system typically no sophisticated cost-based choices have to made at runtime. So, a common notion of a cost model as typical part of the system does not exist in the IR world, in contrast with the database world.

A cost model typically requires information about the cardinalities of the relations involved in the processing of the query. The cardinalities of the base

relations, such as each TFstats and DFstats, can be obtained easily right after the text data has been (partially) indexed. The cardinalities of each $\text{TFstats}_{i\text{Q}}$ needs to be known as well. These cardinalities need to be estimated since they cannot be obtained from the base relations directly. This is not a trivial issue: to estimate the cardinality of each of these relations we need to be able to estimate the selectivity of the semi join ($\ltimes$) between each TFstats_i and Q. It is this selectivity that we want to find a good estimator for.

In the database field also much work had been done on selectivity estimation. We refer to the previous chapter for an overview. We find that the techniques currently known in literature do not suffice. The data in TFstats is most likely Zipfian distributed[1], since it concerns natural language data. Also TFstats and its fragments TFstats_i most likely are very large. This poses some special requirements.

Histograms are able to cope with the Zipfian distribution and deliver accurate predictions. However, in our case, the required histogram per fragment is DFstats_i, which is very large by itself. So, computing the selectivity, which would boil down to semi joining Q with DFstats_i and then adding up all the *df* values divided by $|\text{DFstats}_i|$, would be quite expensive already. We find this unacceptable for an action that has to be done at query optimization time and should therefore be relatively cheap.

Using a parametric method could overcome this performance problem of estimating the selectivity. But, parametric models are typically not as accurate as the histogram methods. Since, the Zipf distribution is highly skewed and TFstats is typically huge, we think the resulting estimation errors form a too great deficiency of this approach.

The other known approaches suffer from drawbacks similar to the histogram or parametric method.

Furthermore, we are interested in the selectivity on a fragment TFstats_i of TFstats, which is not just any subset: it was derived horizontally fragmented based on the horizontal fragmentation of DFstatsṠo, we think the development of a special selectivity model is justified.

This selectivity model should meet the following requirements:

- The selectivity model should be fast in its use at query optimization time.
- The selectivity model should be able to present accurate results for highly skewed data, in particular Zipfian distributed data.

[1] For data that is Zipfian distributed holds 'index' × 'frequency' = 'constant', assuming a descending order of the data elements on frequency [Zip49].

- The selectivity model should be able to exploit and deal with the special properties of the horizontal fragments resulting from the special way they are constructed.

Given all these observations, we can now reformulate the first research question as follows:

Q1 *Can we estimate selectivity as a function of the used fragments?*

This selectivity model is one of the main topics of this dissertation. In Chapter 5 we present our results concerning the selectivity model.

3.2.3 Estimating quality

Like we did for the first and third research question in the previous two sections, we now review the second one:

Q2 *Can we estimate the quality behavior as a function of the used portion of the data?*

Like for the estimation of the selectivity, we now restrict ourselves to horizontal fragments when talking about portions of data.

As we already suggested in the previous chapters, the inexact nature of IR top-N queries might provide an additional opportunity to gain speed. The normal procedure would be to select all fragments of TFstats that contain at least one query term, and then process the ranking for those fragments.

An IR cut off mechanism might be useful to skip any of those fragments TFstats_i as soon as the top-N has reached a fixed state, similar to the techniques we described in the previous chapter. But, as argued before, in Section 2.3.2, this brings along some additional administrative work. If the query optimizer could decide in advance which of the fragments TFstats_i of TFstats could be ignored, we could save on this administrative overhead. Of course, one does not want to do this blindly, just saying that ignoring fragments reduces the quality of the answer without knowing how much the quality actually degrades. In other words, we would like to have a model that can predict the quality implications.

A notion of answer quality is not really known in the database field, since typical database queries are answered in an exact manner. Any notion that does exist, if any at all, is borrowed from other fields, like the IR field.

In the IR field much has been done on quantifying the quality of the results. The quality of an answer is seen as a measure of the retrieval effectiveness of the system delivering the answer. We refer to the previous chapter for an overview of quality metrics.

In the database field also some work has been done on the issue of trading quality for speed as presented in [BMN94] for an SGML document database and in [WB00] for an image retrieval database.

Some work has been done in the IR field on the issue of trading quality for speed. But all those approaches assumed at least that they were working on a known collection, which made it relatively easy to construct a trade-off model. We think this is not realistic for the following reasons.

Since quality is rather subjective, it is usually impossible to build a deterministic model that predicts the quality. Nevertheless, it is possible to make a more generic model that can be trained, for instance, using statistical methods. However, only for a limited set of document collections relevance information is available for training. An example are the well-known TREC [TRE] documents collections. TREC is an acronym for Text REtrieval Conference [Har94], organized annually by NIST, the National Institute of Standards and Technology [NIS]. But these collections are only a subset of all possible document collections in the world. A trade-off model that only works for these special collections is not very useful.

We do have a special requirement. We want a model that can be trained on a known collection, like for instance a TREC collection, but then can be transfered to a collection for which no relevance information is available. This would allow prediction of the quality implications for any indexed document collection in the real world, as long as the training collections are representative enough.

Similar to the selectivity model, the quality model should also be fast in its use, since it is typically used at query optimization time.

Given these consideration we find the development of a new answer quality prediction model justified. Summarizing, such a quality model should meet the following requirements:

- The quality model should be fast in its use at query optimization time.
- The quality model should be able to generalize: once trained for a known collection, or set of collections, it should be usable for another collection.
- The quality model should be able to exploit and deal with the special properties of the horizontal fragments resulting from the special way they are constructed.

Given all these observations, we can now reformulate the second research question as follows:

Q2	*Can we estimate the quality behavior as a function of the relative size of the used fragments?*

In Chapter 6 we present a first attempt to build such a generalizing model by mathematical derivation from our retrieval score function.

3.3 Experimental system

In the previous section we reviewed our basic research questions. In this section we take a closer look at the systems we need to support the research needed to answer these questions.

From the start we had a strong bias towards the miЯRor system. It is part of ongoing research in our group in the area of improving databases to support new application domains. Issues like multimedia retrieval, XML document management, and such play a major role. Our work, preferably, should fit in that context. Facilities like extensibility and rich data models and query languages, like (X)NF^2
scuseXNF@XNF^2 and OO, should be supported. Furthermore, our research should be finished within the running period of the project, being four years. So, building a completely new DBMS from scratch, for instance, is out of the question, since that takes many years. Therefore, setting up the experimental system should not take too much time and should therefore be easy to realize. Since this requirement is not special to our situation, we do not state it as a specific requirement.

To check whether a choice for miЯRor as experimental context is justified we first formulate our requirements. In Section 3.3.1, we take a closer look at the state-of-the-art in DBMSs with integrated IR processing capabilities. We use the known possibilities as a guideline while sketching the requirements that our experimental system should satisfy. Then, in Section 3.3.2, we present the miЯRor system and we argue why this system, indeed, meets the requirements.

We conclude this section with Section 3.3.3, where we specify the hardware platforms we used, and Section 3.3.4, that describes the data collections we used.

3.3.1 Requirements analysis

In the first part of this chapter we reviewed our research questions. In this section we setup a set of requirements for our experimental system. The main objective of this system is, of course, to support the experiments required to answer these research questions. In this section we review the known classes of systems presented in the previous chapter, including the miRRor system, to see to what level those meet these requirements.

Requirements

The first requirement follows directly from the fact that we want to use a specific IR model. Since not all systems support probabilistic IR, this is an important selection criterion:

R1	*The system should support a state-of-the-art probabilistic IR model.*

Since fragmentation plays an important role in our work, it should be properly supported by the experimental system:

R2	*The system should support horizontal data fragmentation.*

The remaining requirements are of a more practical nature. Our experimental system, preferably, should fit the multi-model approach as proposed in [Vri99] because we need precisely that what this approach was designed for. Fragmentation as we use it in our context is mainly a physical issue. But the logical level should be adapted to deal with it. Also, the additional semantics provided by the extensibility at the logical level is required to allow a future optimizer to recognize the IR structures. The optimizer needs this additional information if it has to take special care of query optimization for the IR part of a query. In particular, exploitation of a special-purpose selectivity model for the IR part of the query, requires the optimizer to be able to detect when to use this model instead of just some standard model.

Summarizing, this leads to the following requirement:

R3	*The choice of the system should anticipate embedding in the multi-model context of the research in the group.*

The next requirement is twofold. First of all, this dissertation presents some results that can be used in the implementation of a query optimizer. If these results are to be used, access to the internals of the DBMS are required to make the necessary modifications.

Furthermore, the optimizer architecture should allow the introduction of a generalization of the E-ADT principle as used by the Predator [SLR96, SLR97, SP97, Ses98] system (see previous chapter for details). To clarify this, please recall the example document collection data definition presented in the previous chapter on page 39.

```
DOCCOLL: SET<
           DOCREP<
             DOCID:  STRING,
             AUTHOR: STRING,
             TEXT:   SET<
                       STRING
                     >
           >
         >
```

This description typically exploits the features provided by an XNF^2 system that has a `LIST` and a `DOCREP` structure defined.

Assume both structures are defined in separate extension modules. This is a reasonable assumption, since the `DOCREP` structure is typically designed for a special application area. The `LIST` structure has a more general purpose.

Furthermore, recall the accompanying ranking query on page 40:

```
TOP10 := sublist[1,10](
           sortdesc[THIS.S](
             set2list(
               map[TUPLE<THIS.DOCID, S: score(THIS,Q)>](
                 DOCCOLL
               )
             )
           )
         )
```

The `sublist` operation can be seen as a special selection. Generalization of IR top-N optimization techniques to a set-based way of processing, would mean

that this special selection, in combination with the `sortdesc`, has to be pushed into the `score` operator of the `DOCREP` structure.

Since, both the `LIST` and the `DOCREP` structure are defined in separate extension modules, the E-ADT solution of Predator does not solve this problem. Recall that the E-ADT approach only works within the extension module the enhanced part belongs to. Obviously, this kind of query optimization requires the optimizer to deal with two structures in separate extensions.

Another situation would be where we have a query concerning both the `AUTHOR` attribute and the `TEXT` attribute. This might be the case when the query asks for a top-10 of all documents of authors who's name matches a certain string pattern. Now the optimizer needs to reason over two structures in separate extensions as well since it has to find out whether to do the selection on the author names first, which concerns the `SET` structure extension, and then rank the remaining documents, which concerns the `DOCREP` structure extension, or vice-versa. Note that in this case, the two structures are part of different sub-expressions of the query, in contrast with the first case, which concerns two structures appearing at different levels of the same (sub-)expression of the query.

Clearly, the E-ADT approach of Predator is not sufficient to provide the proper means to do query optimization in this context. An extra abstraction has to be made in our opinion to explicitly bridge the gap caused by the orthogonality of typical structure extensions.

We prefer to call the E-ADT approach an *intra-object* or *intra-structure* optimizer approach, depending on whether it concerns an OO or XNF^2 system. We propose to extend this with an *inter-object*, or *inter-structure*, optimizer approach that explicitly provides optimizer extensions that can work across the boundaries posed by the orthogonality present in typical extensible DBMSs.

So, access to the internals of the DBMS is required to make the necessary modifications to support such an inter-structure optimization technique. Furthermore, the adaptations required to comply with the multi-model approach also require certain access to the internals.

Combined, these issues lead to the following requirement:

R4	*The internals of the system should, preferably, be accessible to support modification of the query optimizer.*

In the next section we analyze the system classes found in literature, as we presented them in the previous chapter, to see whether they meet these requirements.

Analysis

In the previous chapter we described five classes of systems that provide combined querying of text and structured attributes. In this section we review these five classes and describe the advantages and disadvantages of each in light of our experimental needs, as stated above.

The first option we perceived was the class of coupled systems. Since, in this option the IR data is not stored in the DBMS, it does not provide the proper means to do the desired experiments.

The second class of systems exploited the existing query language. This option does allow the implementation of the desired probabilistic IR model and is easy to realize. However, for the support of horizontal fragmentation we have to rely on the used DBMS, and we do not know how much access to the internals is needed to implement our specific fragmentation. Furthermore, we doubt the efficiency of this approach which might cause performance problems when scaling to larger data collections. Also, integration of the results of this dissertation in the future will be difficult, since in that case access to the internals of the system is required to modify the query optimizer. Next to that, we do find an NF^2, or preferably even XNF^2, data model and query language a more intuitive means to model and query the kind of data we are dealing with than the usually supported SQL interface.

The third option, accessing an entire IR system via a single operator, suffers from a somewhat similar problem as the first option: the IR data is not freely available in the DBMS for fragmenting. Either, the data resides outside the DBMS, because the operator extension is just a wrapper for an external system, or, the data should be accessed though an ADT provided by the IR extension module.

Extension of the query language with IR operators, the fourth option, has a lot of positive sides. It does support the probabilistic IR model of our choice. Also, with the right DBMS, fragmentation should be no problem. And from a context perspective, it can be implemented to fit the multi-model approach followed in our group. However, the query languages of the systems presented in literature are still SQL based, lacking the nice features of the nested models provided by XNF^2 and OO systems. Also, realizing such a system requires some extensive programming. In case of exploiting the extension mechanism of a DBMS this leaves us with the future problem of lacking access to the internals to incorporate the results of this dissertation. In case of implementing or internally modifying an existing system, this latter problem is solved, but the programming is much more work.

The final and fifth class, and in particular full integration, has almost no disadvantages. If an existing system is used, problems might occur in the

future when incorporating the results of this dissertation when access to the system internals are needed. If a new system is to be built from scratch, this will not be a problem. But the programming task is huge in that case.

Fortunately, we have the miЯRor system. This means we have an existing system, so no excessive programming overhead, plus complete access to the internals for future incorporation of the results of this dissertation. Also, the miЯRor approach does completely adhere to the multi-model approach proposed in [Vri99], being the prototype developed as part of that research. In the next section we elaborate more on our choice for this system as experimental platform.

3.3.2 Choice: miЯRor, Moa, and Monet

As we already concluded in the previous section, the miЯRor system would be a good choice as experimental system. We are most interested in the two key components of miЯRor: the Moa XNF^2 logical language and the low-level Monet main-memory DBMS, which is used by Moa as physical layer.

Moa has been designed to meet the demands of new application domains. In the miЯRor context it has been extended with a new structure to capture the text retrieval capabilities. This new structure implements the same probabilistic IR model we use in this dissertation. The NF^2 model of Moa provides nice facilities to capture the complex data structures often required by applications such as text retrieval. Due to the NF^2 capabilities a document collection is nicely modeled as a set of documents, being a set containing bags of words, i.e., the documents.

Part of the IR functionality is captured by the extension in Moa, but another part is captured by the definition of procedures at the MIL level in Monet, such as certain probabilistic operators. This adheres nicely to the multi-model approach.

Monet, being a main-memory DBMS, is very predictable in its performance behavior. Because it works in main-memory, only CPU costs play a role, when running on a single CPU. Since the hot-set needs to fit in main-memory, disk IO is quite predictable.

Also, fragmentation is a natural part of the Monet way of processing. Monet uses a fully vertically fragmented data model. This means that all relations are fragmented into relations of only two columns wide, thus preventing attributes of needlessly occupying memory space when not used in actual query processing. Next to that, if tables grow too large, the database administrator has several means to horizontally fragment the data.

Normal DBMSs usually operate disk based and use some buffer management

technique to process long relations in batches or in a so-called pipeline. Monet is not designed to work in close relation with a file system, it expects to get its data in reasonable sizes that fit in main-memory. When relations become too large to be processed in one piece in memory, horizontal fragmentation should be used to partition the relation into chunks manageable in main-memory by Monet.

Since, the miЯRor system, and Moa in particular, is being developed in our own group, we have full access to the internals. Even more, Moa currently lacks an optimizer, so complete freedom exists to suit a future implementation to support the results of this dissertation if desired. We do not have direct access to the internals of Monet. However, we have close relations to the group at CWI [CWI] that is developing Monet.

Finally, the miЯRor prototype already contains several text collections indexed and ready for use. This makes it possible to start our research right away, without wasting much time on setting up a basic system and indexing data.

3.3.3 Systems specifications

We used a Monet 4.x.x DBMS, running either on a dedicated PC running Linux 2.2.16 or a Sun Enterprise E3500 running SunOS 5.6.

The PC has two Pentium™ III 600 MHz CPUs, 1 GB of main-memory, and a 100 GB disk array mounted in RAID 0 (striping) mode. The Sun has four 336 MHz UltraSPARC™-II CPUs, 2 GB of main-memory, and a 20 GB disk volume (dedicated to our experiments) of a large disk array.

On the PC no other users were allowed to run processes while experiments were being conducted and in most cases the experiments were limited to run on one single CPU, leaving the other CPU free for the OS. On the Sun no performance experiments were conducted, so the actual system load was not of importance.

3.3.4 Data collections

For our experiments we use the document collections provided by NIST [NIS] for TREC-6 [TRE97]. As mentioned before, TREC is an acronym for Text REtrieval Conference [Har94], organized annually by NIST, the National Institute of Standards and Technology [NIS]. These collections were already available from experiments with the miЯRor prototype [VH99], but more importantly are commonly accepted as test collections for IR experiments.

Table 3.1 shows some of the key statistics of the used collections after stemming and stopping. These collections range from a couple of dozen MB to about 1

Table 3.1: TREC-6 document collection statistics (as present in our system after stemming and stopping)

Collection	# terms	# term-document pairs	# documents
FR94	91,807	7,700,575	55,505
CR	69,433	5,189,218	27,921
FBIS	202,940	17,385,471	130,471
FT	175,593	26,544,084	210,158
LATIMES	176,853	19,565,029	131,890

GB in data size.

The advantage of these collections above just some document collection is that for these collections also queries with corresponding relevance judgments are available. This is in particularly practical when we want to look at the quality implications of our actions.

The TREC collections are accompanied by a set of 50 queries containing from 9 up to and including 61 terms with an average of 27.441 terms. The TREC quality benchmark tools require that a top-1000 is returned for each of these queries.

References

[BMN94] K. Böhm, A. Müller, and E. Neuhold, *Structured Document Handling - a Case for Integrating Databases and Information Retrieval*, Third International Conference on Information and Knowlegde Management (CIKM'94) (C.K. Nicholas and J. Mayfield, eds.), ACM SIGIR/SIGMIS, ACM Press, November 1994, pp. 147–154.

[CWI] *National Research Institute for Mathematics and Computer Science (CWI)*, URL: http://www.cwi.nl/.

[Har94] D.K. Harman, *Evaluation Techniques and Measures*, Proceedings of the Third Text Retrieval Conference (TREC-3) (Gaithersburg, Maryland), NIST Special publications, November 1994, pp. A5–A13.

[Hie01] D. Hiemstra, *Using language models for information retrieval*, Ph.D. thesis, University of Twente, Enschede, The Netherlands, January 2001.

[NIS] *National Institute of Standards and Technology (NIST)*, URL: http://www.nist.gov/.

[Ses98] P. Seshradri, *Enhanced abstract data types in object-relational databases*, The VLDB Journal **7** (1998), no. 3, 130–140.

[SLR96] P. Seshadri, M. Livny, and R. Ramakrishnan, *E-ADTs: Turbo-Charging Complex Data*, IEEE Data Engineering Bulletin **19** (1996), no. 4, 11–18.

[SLR97] P. Seshadri, M. Livny, and R. Ramakrishnan, *The Case for Enhanced Abstract Data Types*, Proceedings of the 23th VLDB Conference, VLDB, 1997, pp. 66–75.

[SP97] P. Seshradri and M. Paskin, *PREDATOR: An OR-DBMS with Enhanced Data Types*, Proceedings of the 1997 ACM SIGMOD International Conference on the Management of Data, ACM Press, 1997, pp. 568–571.

[TRE] *Text REtrieval Conference (TREC)*, URL: http://trec.nist.gov/.

[TRE97] *Proceedings of the Sixth Text Retrieval Conference (TREC-6)*, NIST Special publications, Gaithersburg, Maryland, November 1997.

[VH99] A.P. de Vries and D. Hiemstra, *The miRRor DBMS at TREC*, Proceedings of the Eigth Text Retrieval Conference (TREC-8) (Gaithersburg, Maryland), NIST Special publications, November 1999, pp. 725–734.

[Vri99] A.P. de Vries, *Content and Multimedia Database Management Systems*, Ph.D. thesis, University of Twente, Enschede, The Netherlands, December 1999.

[WB00] R. Weber and K. Böhm, *Trading Quality for Time with Nearest-Neighbor Search*, Proceedings of the 7th Conference on Extending Database Technology (EDBT 2000), Lecture Notes in Computer Science, Springer, March 2000.

[Zip49] G.K. Zipf, *Human Behavior and the Principle of Least Effort*, Addison-Wesley, Reading, MA, USA, 1949.

Chapter 4

Fragmentation & set-based IR top-N query optimization

4.1 Introduction

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 good database characteristics. The key issue we address in this chapter concerns the effects of our fragmentation approach on the speed and quality of the answers, supported by experimental results.

This chapter is an edited version of a previously published paper [BVBA01]:

H.E. Blok, A.P. de Vries, H.M. Blanken, and P.M.G. Apers, *Experiences with IR TOP N Optimization in a Main Memory DBMS: Applying 'the Database Approach' in New Domains*, in Advances in Databases, 18th British National Conference on Databases (BNCOD 18), July 2001.

In Chapter 1 we defined three research questions. In the previous chapter we argued that we first address the third research question. The main reason for this reordering of the research questions is the fact that fragmentation plays a facilitating role in the context of the other two research questions. In this chapter we address this third research question:

Q3	*Can horizontal fragmentation facilitate the trading of quality for speed to support set-based IR top-N query optimization?*

The next two chapters use the insight obtained in this chapter when addressing the other two research questions.

This chapter is structured as follows. In Section 4.2 we compare the typical, element-at-a-time, IR way of query processing with a, set-based, database variant to rank documents. Next, in Section 4.3, we take a closer look at our approach to horizontally fragment the data. We also present a fragment-at-a-time, i.e. subset-at-a-time, variant of our set-based retrieval algorithm, exploiting similar cut off principles as the presented IR algorithm in Section 4.2. Having set the context, we describe the setup of our experiments in Section 4.4. This section is followed by Section 4.5 where we present the results of the experiments. We conclude this chapter with some concluding remarks in Section 4.6.

4.2 IR query processing

In this section we compare the typical, element-at-a-time, IR way of query processing with a, set-at-a-time, database like way processing. But first, we revisit the previously introduced data model we use to store the statistics required for the $tf \cdot idf$ IR model we use.

4.2.1 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. We use a variant of the $tf \cdot idf$ retrieval model. Recall from the previous chapters, the three basic parameters:

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

document frequency: For each term, df is the number of documents in

TFstats($term$, doc, tf)	(4.1)
DFstats($term$, df)	(4.2)
Q($term$)	(4.3)

Figure 4.1: Relations (schema, also see Figure 3.1)

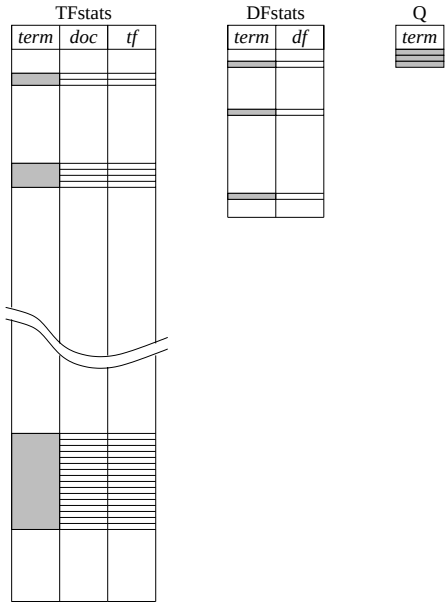

Figure 4.2: Relations (graphical representation, also see Figure 3.2)

which the term occurs.

inverse document frequency: For each term, *idf* is the inverse number of the *df*.

In a more database like notation we can describe these statistics as the relations 4.1 and 4.2, respectively, as shown in Figure 4.1. Furthermore, we introduce the set of query terms, i.e., the unary relation 4.3 shown in Figure 4.1. Figure 4.2 presents these relations in a graphical manner.

In the $tf \cdot idf$ retrieval model, the score of a document given a query is almost completely determined by the sum of the product between the *tf* and *idf*, the inverse of *df*, of the query terms occurring in the document, sometimes normalized with the document length in one way or another. See [VH99] for more detailed information about our specific score formula and retrieval model.

4.2.2 Query processing in an IR system

The algorithm shown in Figure 4.3 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 assume that DFstats is ordered ascending on df.

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

4.2.3 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.[1]

In the implementation, the information retrieval techniques are supported by extensions at both levels of the miЯRor DBMS. While the exact score 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. Also note that since we work on a binary model, several tables represent together the columns of TFstats. However, for didactic reasons we stick to the normal table approach since this has no significant consequences for the core of our idea.

Again, the algorithm, as shown in Figure 4.4, consists of three parts.

4.2.4 Discussion

The main performance bottle-neck lies in handling the TFstats table: TFstats contains over 26 million entries in the experiments performed for this chapter — which is only about a quarter of the complete TREC data set. Only **Part B**[2] in the algorithm described above handles a very large amount of data. At a first glance, the pre-selection on query terms, $\text{TFstats}_{\text{Q}} = \text{TFstats} \ltimes \text{Q}$, at the beginning of **Part B**, may potentially reduce the remaining dataset to a manageable size. But, it is not uncommon in IR experiments that for the average query this reduced remaining dataset still is very large. Sometimes

[1] For the impatient: the usefulness of this seemingly minor step becomes more clear when we present our optimization techniques based on data fragmentation in Section 4.3.

[2] 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 4.2.3.

Part A Limit the TFstats and DFstats to match the terms in query Q:

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

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

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

Part B Loop over the terms to compute the score contribution per document-term pair, and update the document score incrementally each time a new score contribution for that document becomes available. Stop as soon as a test (based on the processed DFstats_Q and TFstats_Q values, knowing that DFstats_Q can only increase, cf. [Fag99]) shows that no document could obtain a score better than the current top-N.

Like before, `t1` and `t2` are tuple-variables. Furthermore we assume the existence of a table `SCORE` that has two columns: `document` and `score`.

```
foreach t1 in DFstats_Q do
  # Find the matching tf values
  TFstats_Q_sel = findrecords(TFstats_Q, t1.term)

  foreach t2 in TFstats_Q_sel do
    tfidf = t2.tf / t1.df;
    if (t2.document in SCORE) then
      updatescore(SCORE, t2.document, tfidf)
    else
      addscore(SCORE, 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 SCORE
do
  if (i < N)
  then
    INSERT t1 INTO TOPRANK
  else
    exitloops
  endif
  i = i + 1
end
```

Figure 4.3: Tuple-oriented IR query evaluation algorithm

Part A Do some initialization given query Q.

Part B Limit the TFstats and DFstats to match the terms in query Q:

$$\text{TFstats}_{\text{Q}} = \text{TFstats} \ltimes \text{Q}$$

and

$$\text{DFstats}_{\text{Q}} = \text{DFstats} \ltimes \text{Q}.$$

Next, place the $\text{DFstats}_{\text{Q}}$ values next to the corresponding entries in $\text{TFstats}_{\text{Q}}$:

$$\text{TFDF}_{\text{lineup}} = \text{TFstats}_{\text{Q}} \bowtie \text{DFstats}_{\text{Q}}.$$

Now, compute the normalized $tf \cdot idf$ value per term-document pair, aggregating the last two columns into one:

$$\begin{aligned} \text{TFIDF} \quad = \quad & \text{SELECT } term, doc, \text{norm}(tf) * \text{norm}(\frac{1}{df}) \text{ AS tfidf} \\ & \text{FROM TFDF}_{\text{lineup}}. \end{aligned}$$

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

$$\begin{aligned} \text{SCORE} \quad = \quad & \text{SELECT } doc, \text{AGGR(tfidf)} \\ & \text{FROM TFIDF} \\ & \text{GROUP BY } doc. \end{aligned}$$

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 score. We abbreviated it here for reasons of simplicity.

Part C Normalize SCORE and select the top-*N* documents:

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

Figure 4.4: Set-oriented IR query processing

even up to about half of the database remains after that pre-selection. This is 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 $tf \cdot idf$ product. 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.

4.3 Data fragmentation and the top-N query optimization problem

In this chapter, 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.

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 4.5, confirms that the data in our test database (see also Section 4.4.1) indeed behaves as predicted by Zipf, validating the underlying reasoning behind our approach.

4.3.1 The fragmentation algorithm

Recall our first basic formalization of the horizontal fragmentation of DFstats and TFstats in the previous chapter. There we defined a set F of fractions f_i, with $\sum_{i=1}^{|F|} f_i = 1$. Now, Figure 4.6 shows the simplified basics of our fragmentation algorithm for splitting the data up in $|F|$ fragments, using the additional information about the term distribution.

Now consider the simple case for which $|F| = 2$. Since the terms in DFstats_1

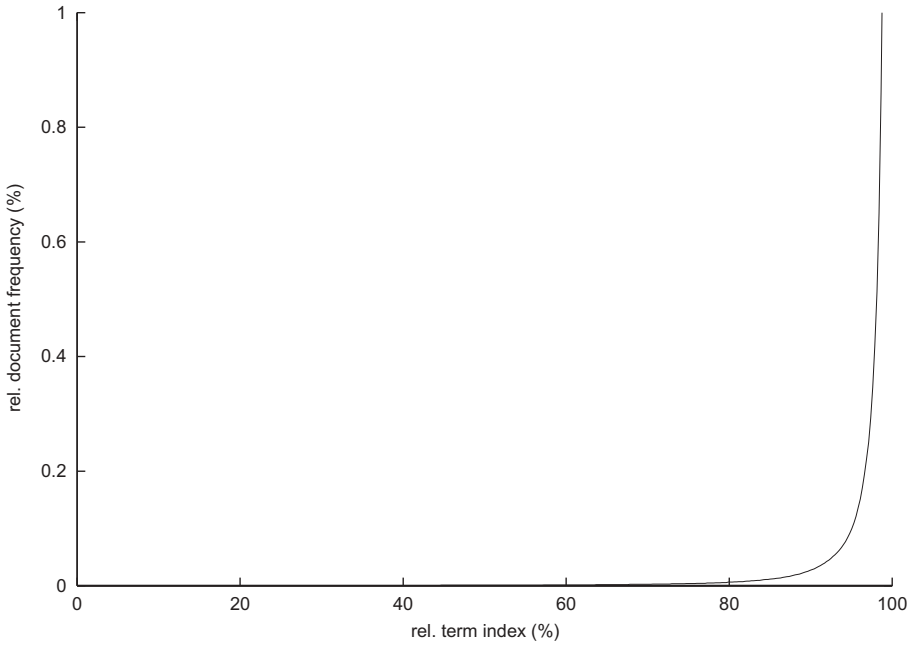

Figure 4.5: Relative document frequency (zoomed on y-axis to show lower values)

Step 1 Sort the DFstats ascending on the df values, i.e., terms that occur in many documents get lower in the list compared to terms that occur in less documents.

$$\begin{aligned} \text{DFstats}_{\text{sorted}} \quad = \quad & \text{SELECT } * \\ & \text{FROM DFstats} \\ & \text{ORDER BY idf DESC} \end{aligned}$$

Step 2 Create $|F|$ fragments DFstats_i such that

$$\begin{aligned} & \text{SELECT COUNT}(*) \\ & \text{FROM DFstats}_i \\ & = \\ & \text{SELECT } f_i \cdot \text{COUNT}(*) \\ & \text{FROM DFstats}_{\text{sorted}} \end{aligned}$$

for $i \in \{1, 2, \ldots, |F|\}$, and

$$\begin{aligned} & \text{SELECT MAX}(df) \\ & \text{FROM DFstats}_i \\ & \leq \\ & \text{SELECT MIN}(df) \\ & \text{FROM DFstats}_{i+1} \end{aligned}$$

for $i \in i\{1, 2, \ldots, |F| - 1\}$.

Step 3 Create $|F|$ fragments TFstats_i such that

$$\text{TFstats}_i = \text{TFstats} \ltimes \text{DFstats}_i$$

for $i \in \{1, 2, \ldots, |F|\}$.

Notice that, for $f_1 = 0.95$, TFstats_1 would now contain approximately 5% (not 95%!) of the tuples of TFstats, and TFstats_i, $i \geq 2$ the rest, due to the high skewedness of the data. So for $|F| = 2$ this would mean that TFstats_2 would contain 95% of the tuples of TFstats, approximately.

Figure 4.6: Fragmentation algorithm

have a low df, and therefore a corresponding high idf, their contribution to the score of a document is likely to be higher than that of terms in DFstats_2 (having higher df, and thus lower corresponding idf, values). In other words, the terms in DFstats_1 are 'a priori' more promising than the terms in DFstats_2. Fortunately, these 'interesting' terms only use about 5% of the data (in case $f_1 = 0.95$). So, in case all query terms are stored in the first fragment, we only need to compute the results using DFstats_1 and TFstats_1. This would mean that the following semi join $\text{TFstats}_\text{Q} = \text{TFstats} \ltimes \text{Q}$ in **Part B** of the algorithm would become $\text{TFstats}_{1\text{Q}} = \text{TFstats}_1 \ltimes \text{Q}$, which is 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 chapter try to determine the effects of ignoring the second fragment on the quality of the answer.

For the general case where $|F| > 2$ this principle also holds: fragments with a lower i are more likely to deliver a more significant contribution to the score and with relatively less computational effort than fragments with higher i.

For reasons of simplicity, it is sometimes more practical to join TFstats and DFstats before fragmenting the data, and propagate the fragmentation into TFstats and DFstats fragments subsequently. This other method is particularly handy to obtain fragments of (almost) equal data size. 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.

4.3.2 Fragment-based IR query processing with top-N cut off

The algorithm in Figure 4.7 shows the top-N cut off idea in a similar manner like the set-based description of the retrieval algorithm as described in Subsection 4.2.3, 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 4.2.2.

Note that this algorithm is a so-called *unsafe* top-N cut off algorithm as we described in Chapter 2. 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 provide sufficient evidence that the top-N cannot be improved anymore. In the algorithm described here the 'topNcanimprove' variable represents the information needed to make this cut off decision. However, this also means that at the cut off moment certain information, e.g., score contributions, has not been taken into account. This usually results in a top-N

Part A Similar to **Part A** in Subsection 4.2.3. Set i to the first fragment that contains a query term.

Part B Similar to **Part B** in Subsection 4.2.3, but this time using fragment i instead of the unfragmented TFstats and DFstats. Let's call the result SCORE_i.

Part B′ Merge SCORE_i into any existing SCORE or otherwise set $\text{SCORE} = \text{SCORE}_\text{i}$.

Part C Normalize SCORE and select the top-N documents:

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

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

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

and the highest score in the remaining intermediate results:

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

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

$$\text{contribUB} = \text{MAXRCONTRIB}(\text{TFIDF}_\text{j}, \ldots, \text{TFIDF}_{|\text{F}|})$$

and the lowest possible score contribution

$$\text{contribLB} = \text{MINRCONTRIB}(\text{TFIDF}_\text{j}, \ldots, \text{TFIDF}_{|\text{F}|}).$$

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

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

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

$$\begin{aligned}\text{SCORE} = {} & \text{SELECT} * \text{FROM SCORE} \\ & \text{WHERE score} \geq \text{topLB} + \text{contribLB} - \text{contribUB}\end{aligned}$$

as soon as:

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

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.

Figure 4.7: Fragment-based IR query processing with top-N cut off

containing the same documents but with an incomplete score. 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.

The *safe* alternative to the unsafe method does indeed return the top-N with the correct score values and inherently can deliver them in the correct order. To obtain these final score values the score 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 (slightly) reduces the profit of top-N cut off due to the extra work that has to be done.

Going even further on the unsafe principle, we can drop the requirement in **Part C″** that the intermediate ranking only can be restricted when

$$\text{MIN(SCORE)} \quad \leq \quad \text{topLB} \; + \; \text{contribLB} \; - \; \text{contribUB}.$$

The algorithm then becomes even 'more' unsafe: documents that have no score at the moment that COUNT(TOPRANK) > N are ignored, even when they would have received a high score otherwise. In turn, the method is very likely to achieve a much better performance due to the earlier and more restrictive limitation imposed on SCORE and all fragments $j > i$. The 'level of unsafe-ness' can be controlled by adding some documents (with initial score 0.0) to SCORE 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 score when they do not contain any query terms, thus not disrupting the ranking process in case they were wrongly added in advance. We call this variant of the usual unsafe method, a *heuristic unsafe* method.

In our case we control the level of unsafe-ness using a factor t (where $0.0 < t < 1.0$) to select the $t \times$ '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 (e.g., 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 t causes the performance to drop rapidly because of the then extremely high number of documents that are forced to be ranked.

4.4 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?

Mean average precision We define the mean average precision (map) as:

$$map \equiv \frac{\sum_{Q \in \mathcal{Q}} ap(Q)}{|\mathcal{Q}|}$$

where $ap(Q)$ is the average precision for each query $Q \in \mathcal{Q}$.

Mean retrieved relevant We define the mean retrieved relevant (mrr) as:

$$mrr \equiv \frac{\sum_{Q \in \mathcal{Q}} rr(Q)}{|\mathcal{Q}|}$$

where $rr(Q)$ is the number of relevant documents retrieved for each query $Q \in \mathcal{Q}$.

Notice that the well-known *recall* measure is defined as $rr(Q)$ divided by the total number of relevant documents for query Q.

Mean execution time The mean execution time (met) is defined as:

$$met \equiv \frac{\sum_{Q \in \mathcal{Q}} et(Q)}{|\mathcal{Q}|}$$

where $et(Q)$ is the (wall clock) execution time measured for each query $Q \in \mathcal{Q}$.

Figure 4.8: Quality and performance measures

2. What are the consequences for the speed?

3. What are the consequences for the quality of the query results, also taking into account the impact of safe/unsafe top-N optimization?

4.4.1 Data set and evaluation measures

Since we want to investigate the trade-off between quality and speed we need a good benchmark for the precision and recall. As we mentioned before, the TREC relevance judgments are the most widely accepted retrieval quality benchmarks. 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. The FT document collection is sufficiently large to show the important effects. We denote the set of all 50 queries by $\mathcal{Q}$. For more information regarding TREC collections we refer to the previous chapter.

We defined four series of experiments, which we evaluate using the measures described in Figure 4.8.

4.4.2 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.

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

$$\mathrm{DFstats}_1 \ltimes \mathrm{Q} = \emptyset.$$

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

Both types of experiments are executed for several different fragmentations, where the relative size in terms of the first fragment, i.e., f_1, varies from 90% to 99.9%, using the fragmentation algorithm described above.

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

We expect that the speed decreases 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 increases the execution time considerably; though resulting in a better quality than for Series IIa.

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. So, $|F| = 25$ and $[f_1, f_2, \ldots, f_{25}]$ is chosen in such manner that $|\text{TFstats}_i| = |\text{TFstats}_j|$ for all $i, j \in \{1, 2, \ldots, 25\}$ and $i \neq j$.

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 4.3 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 4.3 certain conditions can be relaxed 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 $t \in \{0.00,\ 0.05,\ 0.10,\ 0.25,\ 1.00\}$. As explained before we used t to pre-select the fraction of a priori most interesting documents that should not be ignored in case of intermediate result restriction in **Part C″**. 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.

Since variant IIIa takes into account all query terms, we expect the *map* and *mrr* to be equal to the figures measured in Series I. The *met* probably is going

to be better (i.e., 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 [Bro95]. Note that in [Bro95] 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 $t = 0.00$ and quality to be lower but not really bad. For growing t we expect the performance to degrade rapidly since the overhead grows significantly. However, in the case of $t = 1.00$ the quality should reach the same levels as measured for the IIIa variant.

Series IV: Influence of query length and top-N size

Since 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 *met* and (b) the effect of the size of required top-N on the *met*. 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 is going to 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 already results 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 *met* values for lower N. The shorter the required top-

N, the higher the lowest score 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 SCORE. This means that more elements in SCORE are likely to be cut away. Also, the higher score values usually tend to be further away from their closest neighbors. So, the higher the topLB, the higher the chances that restUB is 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.

4.5 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™ 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.

4.5.1 Series I: Baseline

In this subsection we present the *map*, *mrr*, and *met* for the baseline run of the retrieval experiments. Table 4.1 shows the measured values, next to the values provided by TREC as the benchmark. The *met* of course is not available for the benchmark.

Table 4.1: [Series I] Baseline result statistics (TREC benchmark included for comparison)

	map (%)	*mrr*	*met* (s)
Benchmark	100.0	31.8	-
Series I	31.0	22.9	44.4

The *mrr* of 22.9 means that the recall of our unfragmented rum is

$$\frac{mrr_{unfragmented}}{mean\ actual\ no.\ relevant} = \frac{22.9}{31.8} = 0.72.$$

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 *met* 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.

4.5.2 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 4.9, 4.10, and 4.11 plot the *map*, the *mrr*, and the *met* of Series IIa, respectively, together with the baseline performance of the unfragmented case. Figure 4.10 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 $map_{fragmented}$ increases with increasing term count of the first fragment, moving towards the $map_{unfragmented}$. This also holds for the $mrr_{fragmented}$, respectively $mrr_{unfragmented}$. The shape of the plot in Figure 4.11 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 *met* increases ever faster as the term count of the first fragment increases, reaching an *met* of just over 44 seconds when the first fragment contains all terms (100%).

If the first fragment contains 99% of the terms, the *met* is still 3.8 s while the *mrr* is 16.2 (or, average recall is 0.51) and the *map* 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).

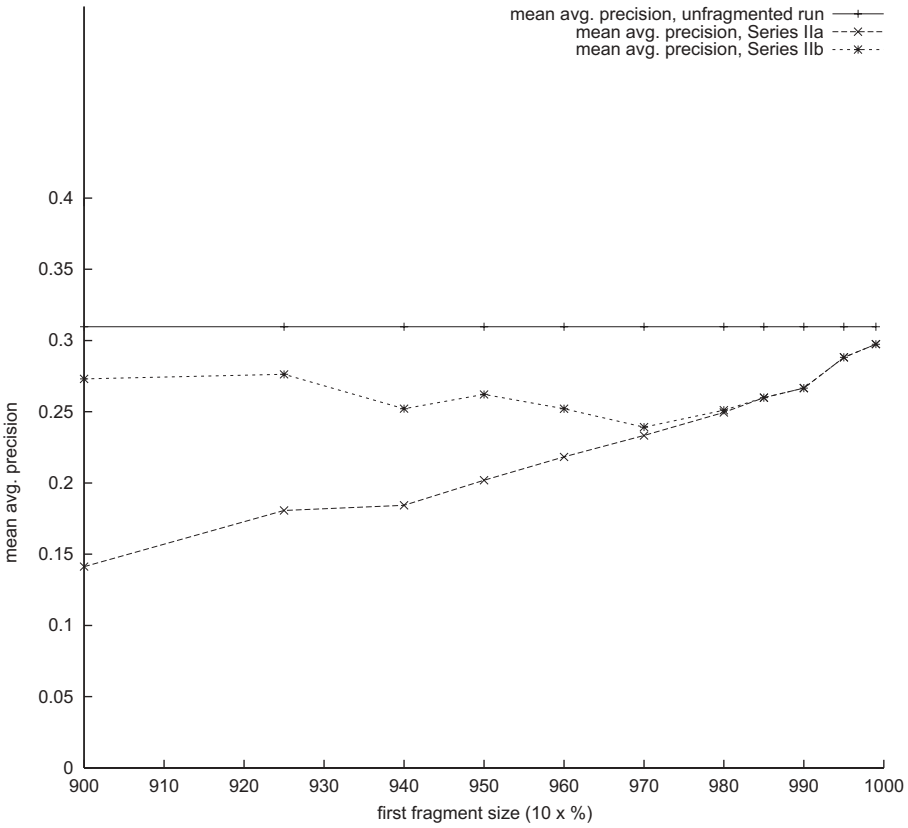

Figure 4.9: [Series IIa/b] *map* for several relative sizes, one fragment used, preferably the first

Series IIb: Use second fragment when first one is unable to handle query

Even the best known retrieval models don't exceed the 40% *map* 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 thatu, already quite poor, upper limit of 40% *map*. 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 4.9 and 4.10, the quality of the results has improved significantly thanks to the switching technique. But, the *met* has risen significantly (Figure 4.11). This observation particularly holds for the fragment size ranges below 98.5%. For larger frag-

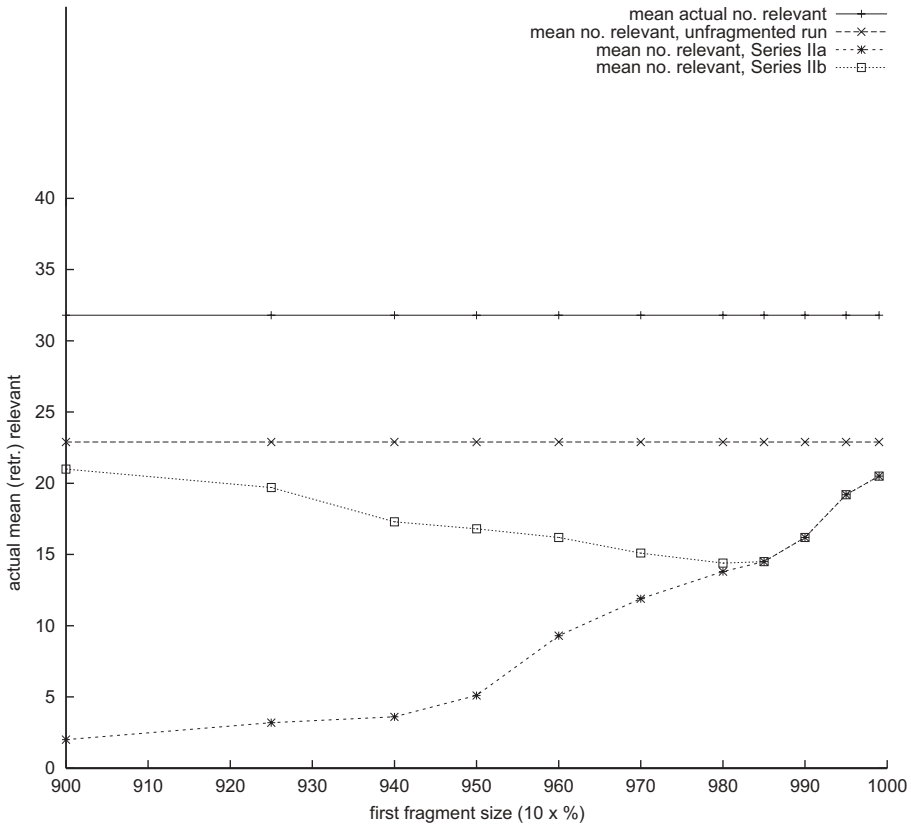

Figure 4.10: [Series IIa/b] mrr for several relative sizes, one fragment used, preferably the first

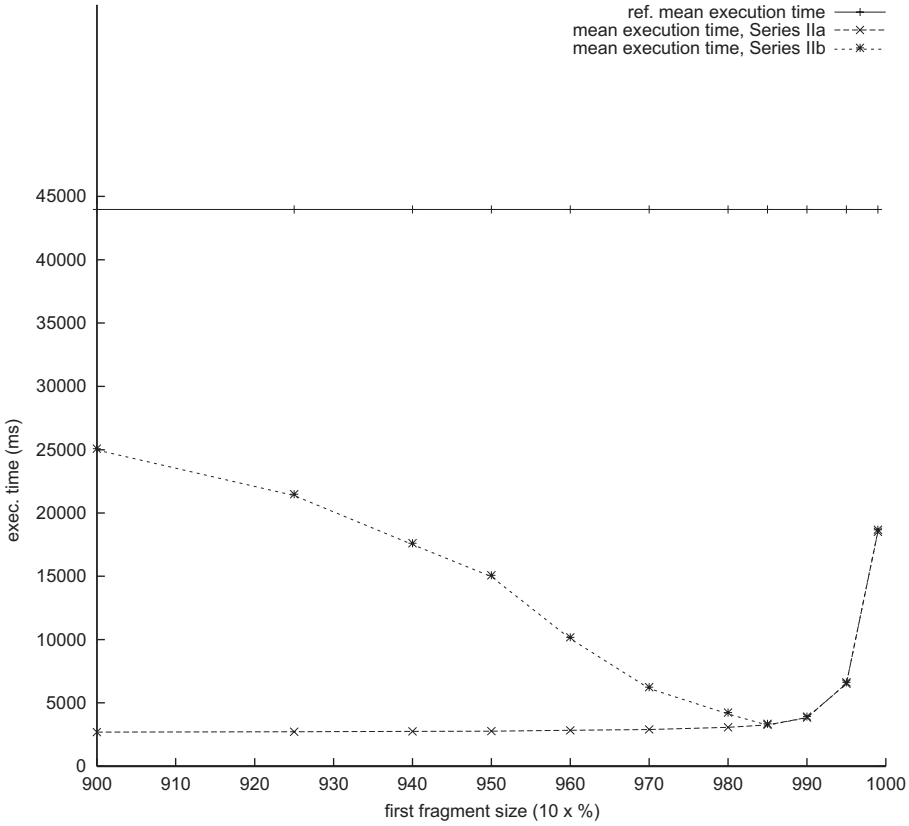

Figure 4.11: [Series IIa/b] met for several relative sizes, one fragment used, preferably the first

ments, the *met* 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 more often the system switches to a rather large second fragment; resulting in quite high 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 *met* 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 *met* 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 *map* never drops below 0.23, and the *mrr* 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%).

Discussion

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

4.5.3 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 4.2 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.

Table 4.2: [Series III] Results of experiments with 25 fragments, with and without top-N cut off (TREC benchmark and Series I results included for comparison)

	map (%)	*mrr*	*met* (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 ($t = 0.00$)	30.0	15.1	7.9
Series IIIc ($t = 0.05$)	29.8	15.6	13.5
Series IIIc ($t = 0.10$)	29.7	15.9	18.7
Series IIIc ($t = 0.25$)	30.0	17.6	33.0
Series IIIc ($t = 1.00$)	30.1	22.9	89.1

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 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 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 t. For the IIIa (and IIIb) case the computational effort (indeed) appeared to increase for fragments with terms with higher *df* — i.e., 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 *map* stayed almost the same, while the *mrr* 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 $t = 1.00$ 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.

4.5.4 Series IV: Influence of query length and top-N size

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 4.12, 4.13, and 4.14 plot the *map*, the *mrr*, and the *met* of Series IVa, respectively, together with the baseline performance of the unfragmented case.

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 t values. For $t = 0.00$ the execution times per query only lightly increases for growing query lengths. However, for growing t 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 4.15, 4.16, and 4.17.

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

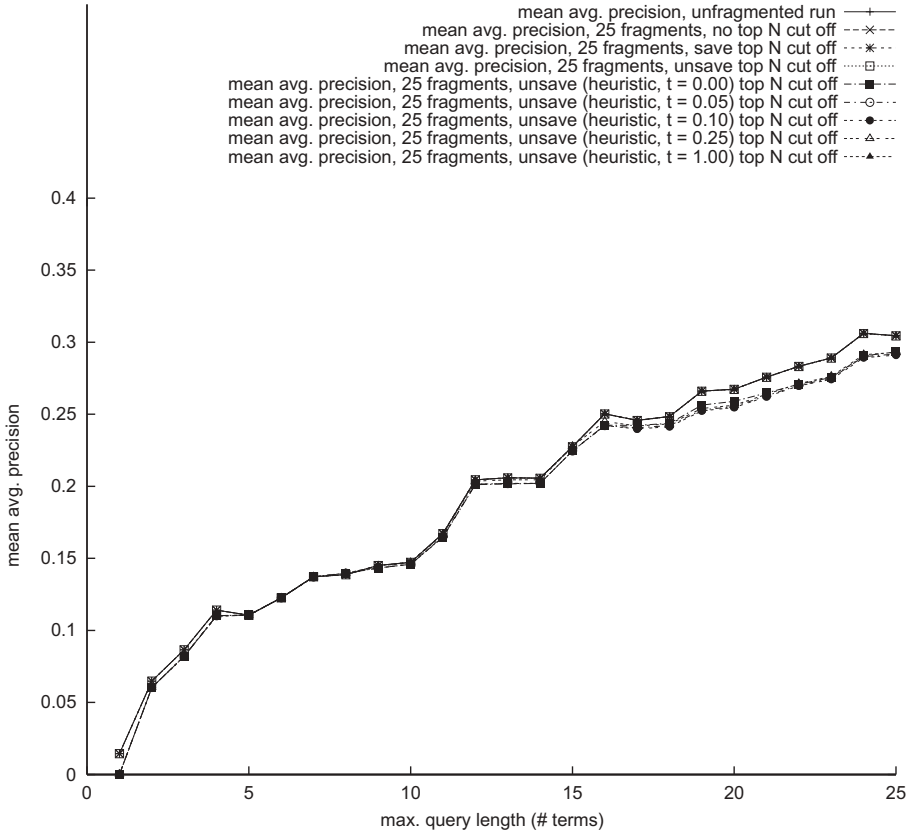

Figure 4.12: [Series IVa] map for several max. query lengths, no/safe/unsafe/-(heuristic) unsafe top-N cut off

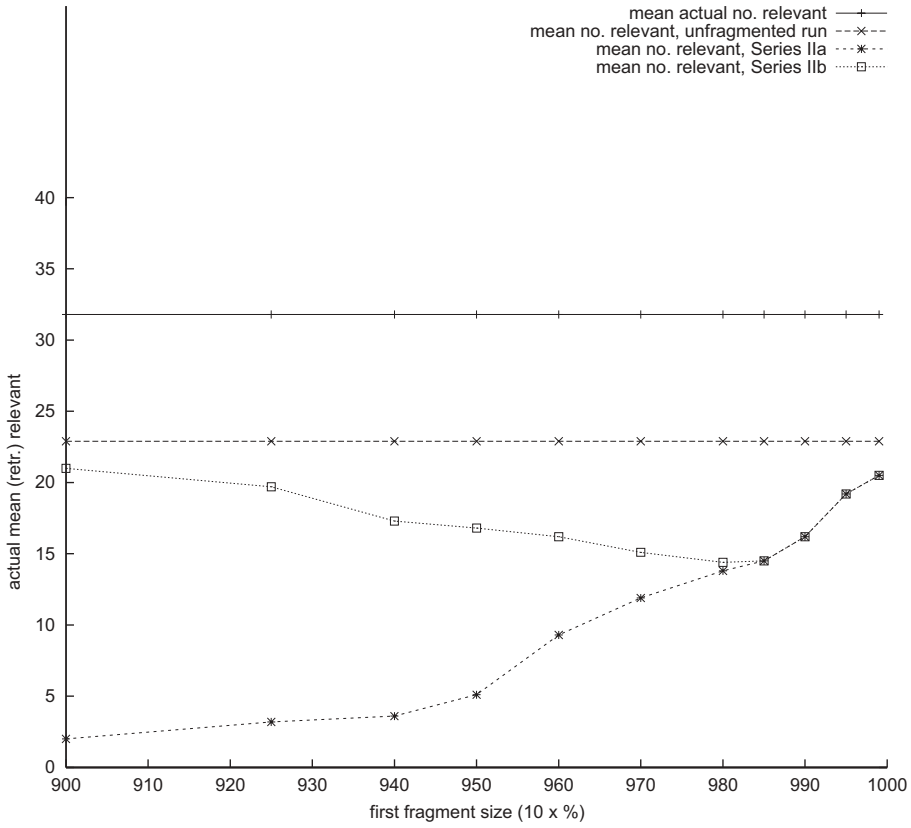

Figure 4.13: [Series IVa] mrr for several max. query lengths, no/safe/unsafe/-(heuristic) unsafe top-N cut off

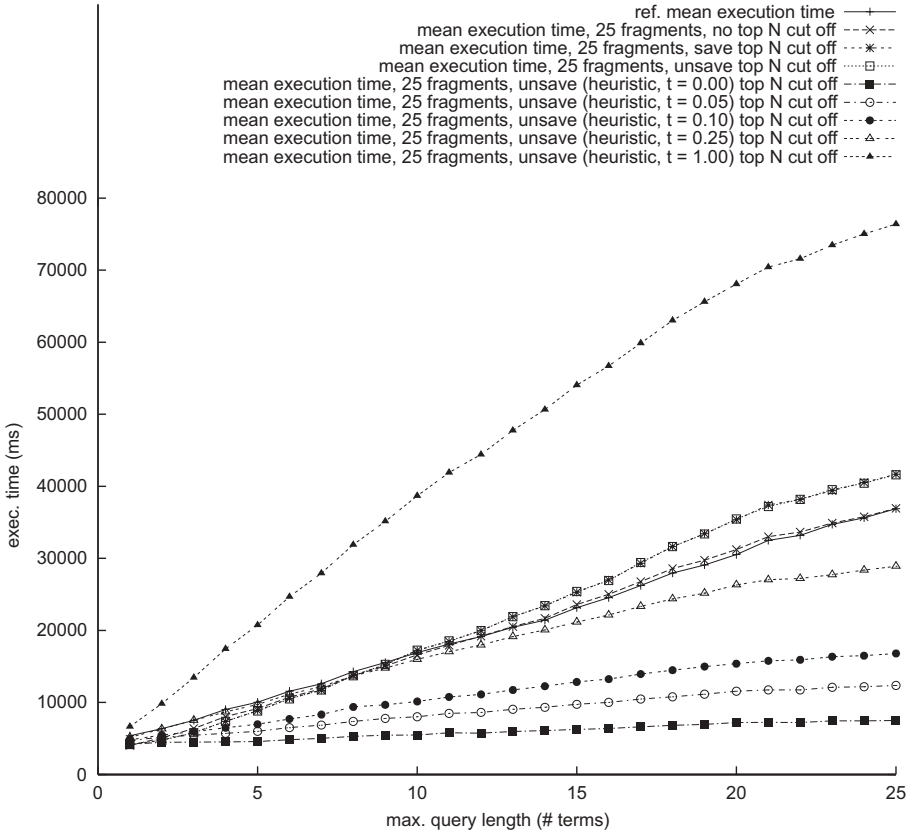

Figure 4.14: [Series IVa] met for several max. query lengths, no/safe/-unsafe/-(heuristic) unsafe top-*N* cut off

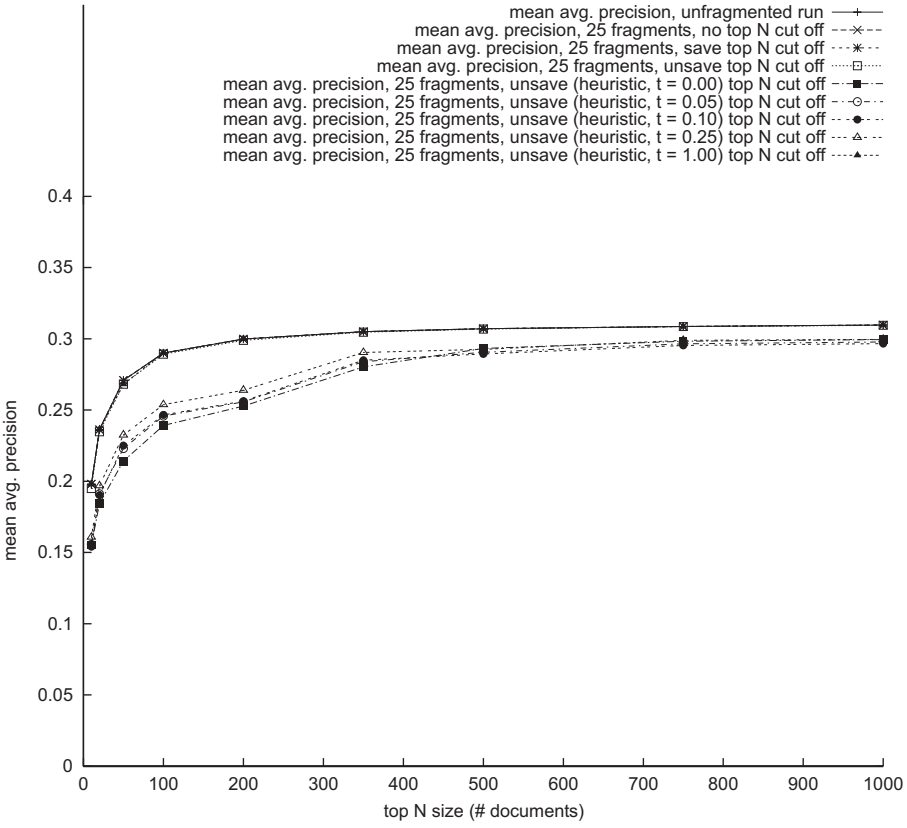

Figure 4.15: [Series IVb] map for several top-N sizes, no/safe/unsafe/(heuristic) unsafe top-N cut off

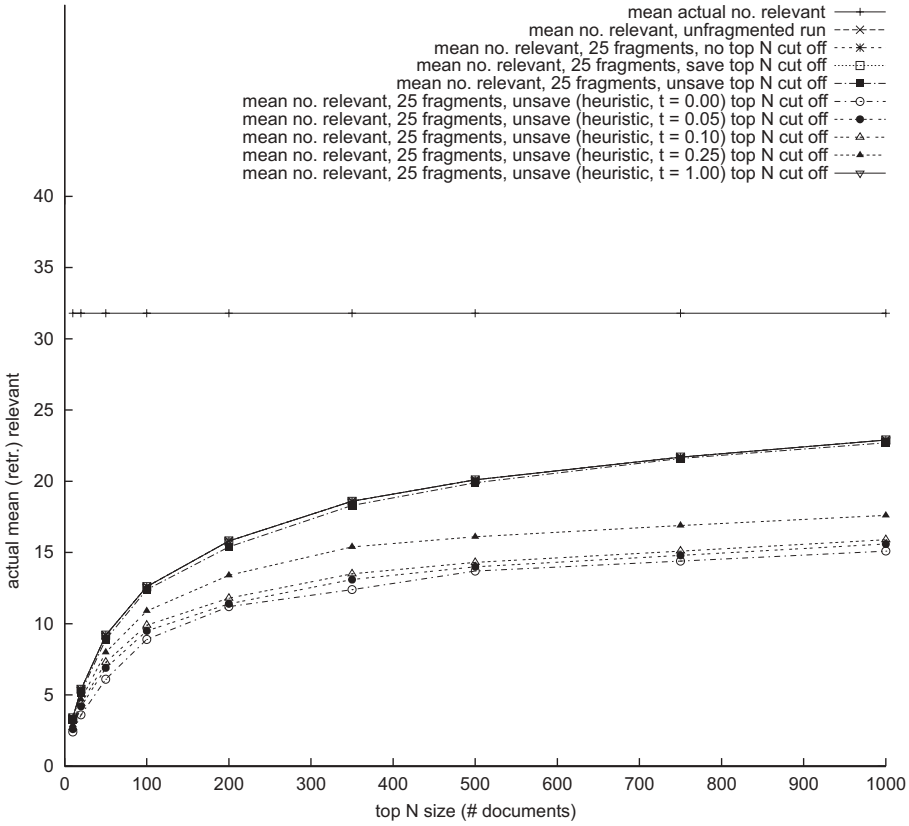

Figure 4.16: [Series IVb] mrr for several top-*N* sizes, no/safe/unsafe/(heuristic) unsafe top-*N* cut off

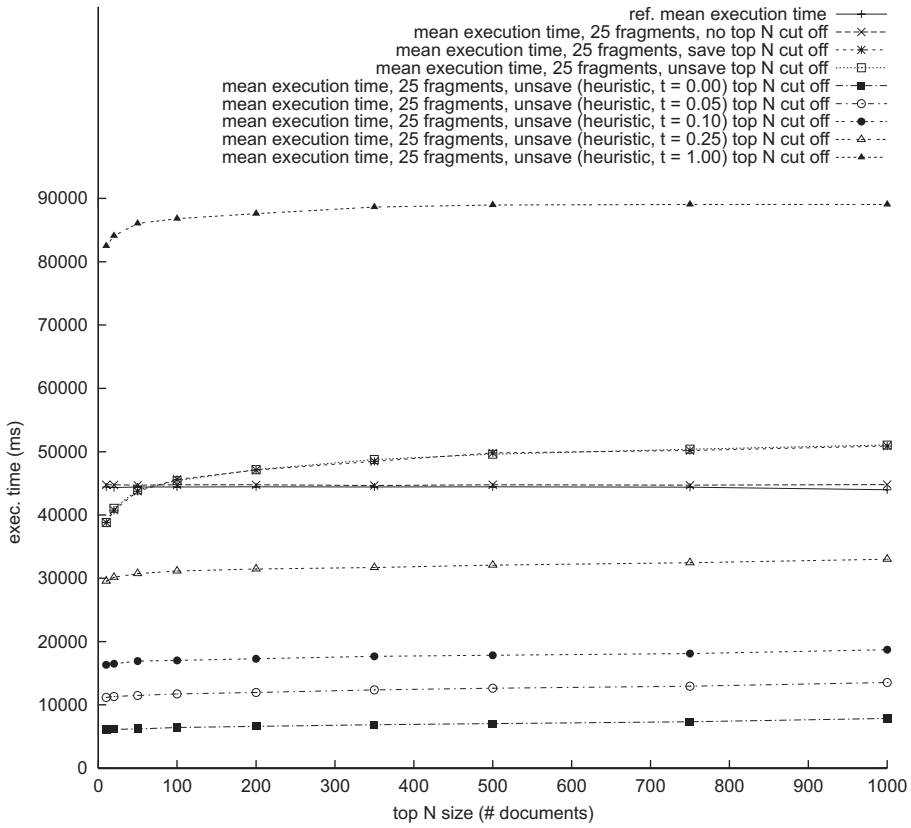

Figure 4.17: [Series IVb] met for several top-*N* sizes, no/safe/unsafe/(heuristic) unsafe top-*N* cut off

is supported by the fact that the heuristic unsafe method always results in significantly lower execution times for lower values of t, 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.

Discussion

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 t 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.

4.6 Concluding remarks

This chapter presents a good case for the suitability of the 'database approach' in the non-standard domain of information retrieval. We first specified the typical IR 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 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 that the smart usage of domain knowledge can 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 [BP98]. Of course Google then (already) operated on a data collection of about 100 times bigger than the one we used in this chapter. 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 have not incorporated in our system, yet.

Concluding, we think this chapter provides sufficient evidence to answer the research question

Q3	*Can horizontal fragmentation facilitate the trading of quality for speed to support set-based IR top-N query optimization?*

positively.

References

[BP98] S. Brin and L. Page, *The Anatomy of a Large-Scale Hypertextual Web Search Engine*, Proceedings of the Seventh International World Wide Web Conference (WWW7), WWW7 Consortium, April 1998.

[Bro95] E.W. Brown, *Execution Performance Issues in Full-Text Information Retrieval*, Ph.D.T./Technical Report 95-81, University of Massachusetts, Amherst, October 1995.

[BVBA01] H.E. Blok, A.P. de Vries, H.M. Blanken, and P.M.G. Apers, *Experiences with IR TOP N Optimization in a Main Memory DBMS: Applying 'the Database Approach' in New Domains*, Advances in Databases, 18th British National Conference on Databases (BNCOD 18) (Chilton, UK) (B. Read, ed.), Lecture Notes in Computer Science, vol. 2097, CLRC Rutherford Appleton Laboratory, Springer, July 2001.

[Fag99] R. Fagin, *Combining fuzzy information from multiple systems*, Journal on Computer and System Sciences **58** (1999), no. 1, 83–99, Special issue for selected papers from the 1996 ACM SIGMOD PODS Conference.

[VH99] A.P. de Vries and D. Hiemstra, *The miRRor DBMS at TREC*, Proceedings of the Eigth Text Retrieval Conference (TREC-8) (Gaithersburg, Maryland), NIST Special publications, November 1999, pp. 725–734.

Chapter 5

Estimating selectivity

5.1 Introduction

Recall from the previous chapters that when talking about selectivity we mean the relative fraction of the term frequency table, TFstats, that is selected when semi joined with a query represented as a table with search terms.

In this chapter, we propose a comprehensive selectivity model and derive a mathematically closed formula to predict the selectivity of a query. The general purpose of this model is to answer another one of our three research questions:

Q1 *Can we estimate selectivity as a function of the used fragments?*

Also recall the three requirements for our selectivity model: fast in its use at query optimization time, accurate results for highly skewed data, and able to exploit and deal with the special properties of our specially constructed horizontal fragments.

Although our discussion is in the context of modeling information retrieval

This chapter is an edited version of a paper submitted for second review:

H.E. Blok, R.S. Choenni, H.M. Blanken, and P.M.G. Apers, *A selectivity model for fragmented relations in information retrieval*, submitted to IEEE Transactions on Knowledge and Data Engineering.

An earlier version of this paper is also available as CTIT Technical Report [BCBA01].

as a database application, the selectivity model is applicable for a number of database applications. The storage of integrated data is rapidly growing, especially in the field of data warehouses. This development supports the progress of a number of advanced applications, such as data mining, decision support systems, multi-media databases. To meet the performance demands of these applications, a widely used strategy is to exploit main-memory capacity by loading a partition of the data in the main-memory such that it is most beneficial. A similar strategy can be applied in the field of information retrieval, as we saw in the previous chapter. Furthermore, in mobile computing systems that autonomously operate, loading the suitable data partition in the system is of vital importance. Such systems may be found in the military arena.

The remainder of this chapter is structured as follows. First, we elaborate a bit more on our approach and related work in Section 5.2. Next, we derive our model in Section 5.3. In Section 5.4, we present experimental verification of our model. Finally, Section 5.5 concludes the chapter.

5.2 Approach

Recall from the previous chapters the three key relations (see Figure 5.1 and Figure 5.2):

Term-document pairs [Expr. 5.1] For each term occurring in a certain document, *doc*, a *term-doc* pair is recorded in the relation TFstats, together with its term frequency *tf*. As mentioned before, this relation, which actually is an *inverted list*, is grouped by term and then ordered on ascending group count. This relation is usually Zipfian distributed and very large. The size of TFstats usually is very large. See Table 3.1 in Chapter 3 for some concrete figures.

Document frequencies [Expr. 5.2] The DFstats relation contains for every term its document frequency, *df*. The *df* of a term is the number of documents in which that term occurs and equals the group count of the term in TFstats. The reader might argue that this relation is redundant, and therefore, a waste of resources. However, the collection TFstats is considered rather static and the DFstats relation usually is relatively small compared to the TFstats relation. The size of DFstats usually does not exceed $2 \cdot 10^5$ elements. Again, we refer to Table 3.1 in Chapter 3 for some concrete figures. Precomputing this relation considerably helps saving time during query evaluation. Also, we assume that DFstats is ordered ascendingly on *df*, meaning that it is ordered similarly to TFstats.

TFstats($term$, doc, tf)	(5.1)
DFstats($term$, df)	(5.2)
Q($term$)	(5.3)

Figure 5.1: Relations (schema, also see Figures 3.1 and 4.1)

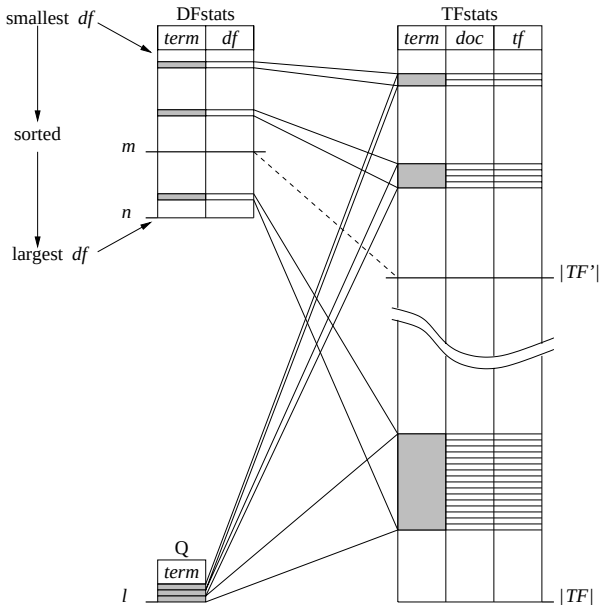

Figure 5.2: Relations (graphical representation, also see Figures 3.2 and 4.2)

Query [Expr. 5.3] This relation is nothing more than a set of terms, constructed each time a user query is presented to the system for evaluation. This relation is relatively small compared to the other two relations presented here. It usually does not contain more than 100 terms.

Our selectivity model is focussed towards the prediction of the number of hits in relation TFstats($term$, doc, tf), or a fragment of it, given a set of terms. In other words, our model predicts the size of the (semi)join between the relations TFstats, or a fragment of it, and Q.

For simplicity we only consider the case where we have two fragments, so $|F| = 2$. Let us assume that we are only interested in the first m tuples of relation DFstats — which has n tuples in total — so $f_1 = \frac{m}{n}$ and $f_2 = 1 - f_1$, and their corresponding tuples in TFstats, which we call TFstats$'$. The problem is to predict the size of the (semi)join between TFstats$'$ and Q.

We assume that the query and data distributions are known a priori. In the field of information retrieval, the query and data are assumed to be equally distributed, thus fulfilling our assumption, since the data distribution is typically known. In defining our selectivity model, we also take into account the aspect of horizontal data fragmentation.

To evaluate our model, we have performed a number of experiments to determine whether the predicted selectivity values obtained by our model meet the experimental selectivity values. The experimental values have been obtained by measuring the selectivity for a number of representative document collections (which are part of the TREC data set).

Recall from Chapter 2 that in the literature, a large number of efforts has been reported on the prediction of selectivity factors in different contexts and under different assumptions. Roughly two directions can be distinguished in the prediction of selectivity factors. Research in the first direction has been focussed to the prediction of the number of page or block accesses. Although our problem definition has similarities with above-mentioned problem, it differs on some fundamental points. First of all, we are interested in the plain number of hits (which can be regarded as tuples) in a relation. In the prediction of block accesses, the number of 'hits' is given, which is not the case in our problem definition. Secondly, we do not assume a particular distribution of the data. So, our problem definition indeed differs from the problem of predicting disk accesses.

The second research direction mainly focuses on the prediction of intermediate join or selection result sizes. This area has also been subject to research extensively and can be divided into four categories: non-parametric, parametric, curve fitting, and sampling.

Our problem definition fits in this second research direction. However, our approach differs on two fundamental points compared to the above-mentioned categories. First, we focus on the estimation of the selectivity for a fragmented database. In our case, the fragment size can be regarded as a parameter.

Second, the model we propose in this chapter to estimate the selectivity for fragmented databases, does not fit very well in the categorization typically used in the second research direction. Our model is not a sampling, curve fitting, or non-parametric method. We propose a model that relies on two parameters that are computed from the data distribution. However, we do not approximate the data distribution by a 'standard' distribution function. Instead, we use the real discrete distribution of the data and compute a mathematical approximation to the real estimated value for the selectivity (see Section 5.3 for the details). Therefore, at best our approach can be regarded as a combination of a non-parametric and a parametric approach. To our best knowledge, such an integrated approach has not been reported before in the literature.

t_i	$\equiv$	'a term'	(5.4)
T	$\equiv$	$[t_1, t_2, t_3, \ldots, t_i, \ldots, t_n], n \geq 1$	(5.5)
d_j	$\equiv$	'a document', $[t_i \mid t_i$ is 'a term'$]$	(5.6)
D	$\equiv$	$\{d_1, d_2, d_3, \ldots, d_j, \ldots, d_{\mid D \mid}\}$	(5.7)
C	$\equiv$	$\{(t_i, d_j) \mid t_i \in d_j, d_j \in D\}$	(5.8)
df_i	$\equiv$	$\mid \{d_j \mid (t_i, d_j) \in C\} \mid$	(5.9)
DF	$\equiv$	$[df_1, df_2, df_3, \ldots, df_i, \ldots, df_n]$	(5.10)
Q	$\equiv$	$\{t_i \mid t_i \in T\},\ l = \mid Q \mid$	(5.11)

Figure 5.3: Basic mathematical definitions [1/2]

5.3 Mathematical model

In this section we present a mathematical approach to our selectivity problem. To make it more accessible, we first introduce some terminology. Having presented that, we present another problem, analogous to the original one, which demonstrates the key issues more intuitively. Using this analogy, we describe a mathematical model for the expected selectivity ratio in the third subsection.

5.3.1 Preliminary definitions and properties

This subsection is devoted to some necessary mathematical definitions. Furthermore, we also present a pair of handy, basic properties directly following from the presented definitions.

Definitions

To enable a somewhat more elegant mathematical formulation during the construction of a selectivity model, we introduce a mathematical notation corresponding to the relational definitions presented in Section 5.1.

First of all, we identify terms [Expr. 5.4] and we assume that our vocabulary consists of n *unique* terms [Expr. 5.5]. For sake of convenience, T is ordered such that t_1 is the most discriminative term (i.e., it occurs in the least number of documents of all terms) and t_n the least discriminative term (i.e., it occurs in the highest number of documents of all terms)[1].

Secondly, we identify documents [Expr. 5.6] and the set of all documents [Expr. 5.7] known to our system. A document is defined as a list of terms. In the following, we mean by $t_i \in T$ the i-th term in list T.

[1] Perhaps unnecessarily we note that lists can have duplicates. However, in our case T does not have duplicates.

$$
\begin{array}{lcll}
T' & \equiv & [t_1, t_2, t_3, \ldots, t_i, \ldots, t_m], 1 \leq m \leq n & (5.12) \\
Q' & \equiv & \{t_i | t_i \in Q \wedge t_i \in T'\} & (5.13) \\
C' & \equiv & \{(t_i, d_j) | t_i \in T' \wedge t_i \in d_j \in D\} & (5.14) \\
 & \subset & C & \\
C'_Q & \equiv & \{(t_i, d_j) | t_i \in Q' \wedge t_i \in d_j \in D\} & (5.15) \\
 & \subset & C' & \\
DF' & \equiv & [df_1, df_2, df_3, \ldots, df_m], 1 \leq m \leq n & (5.16)
\end{array}
$$

Figure 5.4: Basic mathematical definitions [2/2]

Using these notions, we define the set of all term-document pairs known to our system [Expr. 5.8]. A pair (t_i, d_j) means that term t_i occurs in document d_j. This is the set representation of the TFstats$(term, doc, tf)$ relation, as defined previously. Since the tf attribute is not relevant to the remainder of this chapter we have left it out for clarity.

Based on this set, we define the document frequency per term t_i [Expr. 5.9] and DF is the list of document frequencies for all t_i [Expr. 5.10].

Finally, we define the set of query terms, and the query length [Expr. 5.11]. Note that we assume no *out-of-vocabulary* words, meaning that all terms occurring in a query are known to the system (after stemming). For sufficiently large data collections, T is so large that this assumption is reasonable. We might have to reconsider this assumption, if it appears that our selectivity model does not perform well enough in practice.

As explained before, we are only interested in the more discriminative terms, say m of the n terms in total. Since we assumed T to be ordered from most to least discriminative, we can take the m first elements, resulting in T' [Expr. 5.12].

The more discriminative part of a query [Expr. 5.13] directly follows from this. In most cases we use k to denote $|Q'|$. Note that $|Q'| \leq |Q|$.

From the restricted vocabulary and query directly follow corresponding restricted versions of the set C [Expr. 5.15 and 5.16, respectively]. Note, in practice C'_Q is obtained by a (semi)join between C' and the query, and serves as input for the rest of the query plan that produces the document ranking.

As before, we can now define the list of all document frequencies for the more discriminative terms [Expr. 5.16]. Note that we omitted the term attribute of the DFstats relation in this list-wise notation, since the index i of each df_i already implies the term it is related to.

Finally, we define the selectivity ratio as the relative fraction of the most discriminative collection part selected by the query [Expr. 5.17]. It is our

$$sel_{\sigma_{measured}} \equiv \frac{|C'_Q|}{|C'|} \tag{5.17}$$

Figure 5.5: Measured selectivity ratio definition

$$\sum_{i=1}^{n} df_i = |C| \tag{5.18}$$

$$\sum_{i=1}^{m} df_i = |C'| \tag{5.19}$$

Figure 5.6: Basic properties

objective to estimate this fraction.

Properties

From the definitions of df_i [Expr. 5.9] and C [Expr. 5.8] directly follows a nice mathematical relation between these two [Expr. 5.18]. A similar relation [Expr. 5.19] holds between df_i and C' [Expr. 5.15]. We use both relations in the remainder of this chapter in several formula manipulations.

5.3.2 An analogy: playing math darts

To make the discussion more accessible, we reformulate our problem in terms of a dart game.

As an analogy for our TFstats relation, we construct a special dart board, as shown in Figure 5.7. The board consists of concentric rings (so no sectors like one is accustomed to for the normal dart board). The surface area of each ring i is equal to df_i, where the 'bulls eye' in our case corresponds to the records in TFstats belonging to term 1 and the outer most ring to term n.

A query of length l can be seen as throwing l darts completely aselectively at our dartboard ('just' tell the darts player to do so instead of the usual non-aselect aiming for the bull). Note that this means that the probability to hit a ring is therefore proportional to its surface area, assuming that $l << n$.

Also, we do not allow one ring to be hit more than once. In case a dart hits a ring that has been hit already, the dart has to be taken out and thrown again (corresponding to the property that each query term occurs only once in a query). Furthermore, we assume that our dartboard is so huge that each dart hits the board (corresponding to our 'no out-of-vocabulary terms' assumption stated before).

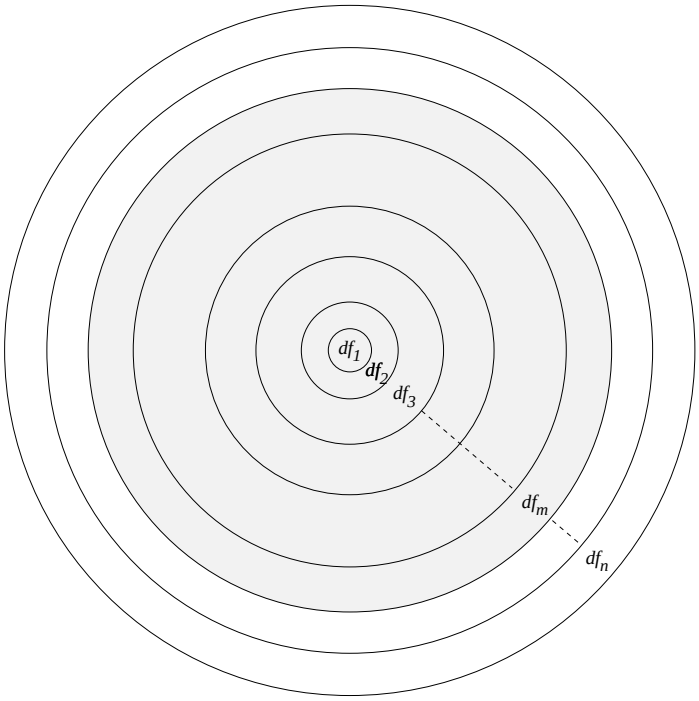

Figure 5.7: Our mathematical dartboard: each ring i has surface area df_i

$$(X = x) \equiv \text{The total surface area of the inner } m \text{ rings that are hit, equals } x, \text{ given that } l \text{ darts have been thrown at the board.} \tag{5.20}$$

$$(K = k) \equiv \text{The event that } k \text{ of the given } l \text{ darts hit the inner } m \text{ rings}, k \leq l \tag{5.21}$$

$$p_i \equiv \frac{df_i}{|C'|} \tag{5.22}$$

$$\vec{p} = (p_1, p_2, p_3, \ldots, p_m) \tag{5.23}$$

$$\alpha \equiv \sum_{i=1}^{m} \frac{df_i^2}{\left(\sum_{i=1}^{m} df_i\right)^2} = \frac{\sum_{i=1}^{m} df_i^2}{|C'|^2} \tag{5.24}$$

$$\beta \equiv \sum_{i=1}^{m} \frac{df_i}{\sum_{i=1}^{n} df_i} = \frac{|C'|}{|C|} \tag{5.25}$$

Figure 5.8: Additional definitions

After l darts have been thrown at the board, we compute the total surface area of the rings hit with ordinal number less or equal to m, i.e., add the square size of each gray ring in Figure 5.7 that has been hit to the total.

The problem we want to solve now, is rephrased as: *what is the expected total surface area of the rings hit within the inner m rings?*

In the remainder, we use this problem in our modeling efforts, instead of referring to the original selectivity problem.

5.3.3 Selectivity model

In this subsection, we first introduce some additional definitions and properties. Next, we use these definitions and properties to finally construct a mathematical expression describing the desired expected surface area of rings hit within the inner m rings.

Additional mathematical definitions

We introduce some additional mathematical definitions to formalize our problem a bit more.

Before we can define the expected values, in which we are interested, we present

$$SDF_Z \equiv \sum_{i \in Z} df_i, \text{ where } Z \subset \{1, 2, 3, \ldots, m\} \tag{5.26}$$

$$h_k : \vec{p} \mapsto \sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \\ |Z|=k}} \prod_{i \in Z} p_i \tag{5.27}$$

$$g_k : \vec{p} \mapsto \left(\sum_{i=1}^{m} p_i \right)^k \tag{5.28}$$

Figure 5.9: Utility function definitions

$$P(K = k) = \binom{l}{k} \beta^k (1 - \beta)^{(l-k)} \tag{5.29}$$

$$P(X = x | K = k) = \sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \\ SDF_Z = x \\ |Z|=k}} \prod_{i \in Z} p_i \tag{5.30}$$

$$E(X|K = k) \equiv \sum_{x} P(X = x|K = k)x \tag{5.31}$$

$$E(X) = \sum_{k=1}^{l} E(X|K = k)P(K = k) \tag{5.32}$$

Figure 5.10: Main probability and expected value definitions

some auxiliary definitions, shown in Figure 5.8:

[**Expr. 5.20**] The event that the total surface area of inner m rings that has been hit is x.

[**Expr. 5.21**] The number of darts that has hit one of the inner m rings (i.e., the number or query terms that falls within the first m terms) is k.

[**Expr. 5.22**] The probability of a dart hitting ring i of the inner m.

[**Expr. 5.23**] The probability vector for the inner m rings.

[**Expr. 5.24**] The average ring area of the inner m rings, weighted over the probability to hit that ring. This formula appears to be of practical use later on in this chapter, but is not of much significance by itself.

[**Expr. 5.25**] The probability of a dart hitting the inner m ring area.

Furthermore, we listed some utility functions in Figure 5.9. These functions play a supporting role in the remainder of this chapter. Note, an element $i \leq m$ of Z corresponds to a term t in T', and df_i is the df value of t.

$$sel_{\sigma_{theoretical}} \equiv \frac{E(X)}{|C'|} \tag{5.33}$$

Figure 5.11: Expected selectivity ratio definition (theoretical)

$$\frac{\partial}{\partial p_{i'}} h_k(\vec{p}) = \sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \setminus i' \\ |Z|=k-1}} \prod_{i \in Z} p_i \tag{5.34}$$

$$\frac{\partial}{\partial p_{i'}} h_k(\vec{p}) \leq \frac{\partial}{\partial p_i} g_k(\vec{p}) \tag{5.35}$$

$$\frac{\partial}{\partial p_{i'}} g_k(\vec{p}) = k g_{k-1}(\vec{p}) \tag{5.36}$$

Figure 5.12: Utility function properties

Using the definitions in Figures 5.8 and 5.9, we define the probability that k of the l darts (i.e., query terms) hit the inner m ring area [Figure 5.10, Expr. 5.29]. Furthermore, we define the conditional probability that the surface area of rings hit in the inner m ring area is x, given that k darts hit that area [Expr. 5.30].

The conditional expected surface area of hit rings in the inner m ring area [Expr. 5.31] follows directly from [Expr. 5.30]. In turn, the unconditional version of this expected value [Expr. 5.32] follows directly from [Expr. 5.31] and [Expr. 5.29].

Actually, we are interested in the relative surface area hit in the inner m ring area (i.e., the selectivity ratio). This means that we have to divide by the total surface area of the inner m ring area $|C'|$ to get the definition of the expected selectivity ratio [Expr. 5.33].

Note, we assume that l is much smaller than m. Otherwise our probability and expected value definitions would not be valid. Fortunately this requirement is met in practice, since l typically does not exceed 100 whereas m is typically 100 times, or more, larger.

Additional mathematical properties

In addition to the definitions we presented above, we need some properties of h_k and g_k in order to make the final formula manipulation steps more accessible [Figure 5.12]. Note that [Expr. 5.34] and [Expr. 5.36] make use of the fact that h_k and g_k are polynomials. A proof of both expressions can be found in Appendix A, Sections A.1 and A.2, respectively.

[Expr. 5.35] follows directly from the fact that g_k is equal to h_k plus some additional higher order terms.

Actual model construction

Here we derive our final model, mainly by means of manipulating the given formulas.

We start by using [Expr. 5.30] to expand [Expr. 5.31] and rewrite it into a more useful form:

$$\begin{aligned} E(X|K=k) &\equiv \sum_x P(X=x|K=k)x \\ &= \sum_x \left(\sum_{\substack{Z\subset\{1,2,3,\ldots,m\} \\ SDF_Z=x \\ |Z|=k}} \prod_{i\in Z} p_i \right) x \\ &= \sum_{\substack{Z\subset\{1,2,3,\ldots,m\} \\ x=SDF_Z \\ |Z|=k}} \prod_{i\in Z} p_i \, x \\ &= \sum_{\substack{Z\subset\{1,2,3,\ldots,m\} \\ |Z|=k}} \left(\prod_{i\in Z} p_i\right)\left(\sum_{i\in Z} df_i\right). \end{aligned} \tag{5.37}$$

If we take [Expr. 5.37] and use it to expand [Expr. 5.32] we get:

$$\begin{aligned} E(X) &= \sum_{k=1}^{l} E(X|K=k)P(K=k) \\ &= \sum_{k=1}^{l} \sum_{\substack{Z\subset\{1,2,3,\ldots,m\} \\ |Z|=k}} \left(\prod_{i\in Z} p_i\right)\left(\sum_{i\in Z} df_i\right) P(K=k). \end{aligned} \tag{5.38}$$

Via regrouping of the terms in the inner sums we get:

$$E(X) = \sum_{k=1}^{l} \sum_{i'=1}^{m} p_{i'}\, df_{i'} \sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \setminus i' \\ |Z|=k-1}} \prod_{i \in Z} p_i\, P(K=k). \tag{5.39}$$

We refer to Appendix A, Section A.3, for a more elaborate explanation of this step. Note that for the situation $k = 1$ the '$\sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \setminus i' \\ |Z|=k-1}} \prod_{i \in Z} p_i$'-part is assumed to be 1 for convenience sake. This assumption is made to correct a singularity that might otherwise arise in this case.

The '$\prod_{i \in Z} p_i$'-part can be replaced by an expression in h_k, using [Expr. 5.27] and [Expr. 5.34]:

$$E(X) = \sum_{k=1}^{l} \sum_{i'=1}^{m} p_{i'}\, df_{i'} \frac{\partial}{\partial p_{i'}} \left(h_k(\vec{p})\right) P(K=k). \tag{5.40}$$

Successively applying [Expr. 5.35] and [Expr. 5.36] gives:

$$\begin{aligned} E(X) &\leq \sum_{k=1}^{l} \sum_{i'=1}^{m} p_{i'}\, df_{i'} \frac{\partial}{\partial p_{i'}} \left(g_k(\vec{p})\right) P(K=k) \\ &= \sum_{i'=1}^{m} p_{i'}\, df_{i'} \sum_{k=1}^{l} k g_{k-1}(\vec{p}) P(K=k). \end{aligned} \tag{5.41}$$

From the definitions of g_k and p_i ([Expr. 5.28] and [Expr. 5.22] respectively) follows directly:

$$E(X) \leq \sum_{i'=1}^{m} p_{i'}\, df_{i'} \sum_{k=1}^{l} k 1^{k-1} \binom{l}{k} \beta^k (1-\beta)^{(l-k)}. \tag{5.42}$$

Since '$\sum_{k=1}^{l} k \binom{l}{k} \beta^k (1-\beta)^{(l-k)}$' is the expression for the expected value of a binomial distribution with parameters l and β we get:

$$E(X) \leq \sum_{i'=1}^{m} p_{i'}\, df_{i'} l\beta. \tag{5.43}$$

If we substitute $E(X)$ in [Expr. 5.33], we get (with some minor rewriting and substituting [Expr. 5.24]):

$$sel_{\sigma_{estimated}} \equiv l\alpha\beta \tag{5.45}$$

Figure 5.13: Expected selectivity ratio definition (estimator)

$$\begin{aligned} sel_{\sigma_{theoretical}} &\equiv \frac{E(X)}{|C'|} \\ &\leq l\left(\frac{\sum_{i=1}^{m} p_i df_i}{|C'|}\right)\beta \\ &= l\alpha\beta. \end{aligned} \tag{5.44}$$

We now define our estimator for the expected selectivity ratio as the right hand side of this expression, as shown in Figure 5.13. This completes our model.

5.4 Experimental verification

This section describes the test databases that have been used, and the experimental results that have been obtained on them.

5.4.1 Setup

As mentioned before we used document collections from the TREC set for the evaluation of our selectivity model. In this case we used the FT, LATIMES, and CR collections. For the details of these collections we refer to Table 3.1 in Chapter 3.

The queries we used, are the 50 retrieval queries, also known as topics in the IR field, from TREC-6. These queries range in length (l) from 9 up to and including 61 terms with an average of 27.441 terms. This query length might seem unrealistically large since queries typically entered by a user consist of only a few terms. Note however, that many applications exist in which the user does not enter the actual query that is actually executed by the retrieval system. Examples of such situations are: automatic query expansion (used by certain relevance feedback and thesaurus exploiting techniques), or searching for similar texts given an example text (one might think of patent verification).

For all three collections, we evaluated our selectivity model for 11 cases of m, with m such that

$$\frac{m}{n} \in \{0.9, 0.925, 0.94, 0.95, 0.96, 0.97, 0.98, 0.985, 0.99, 0.995, 0.999\}.$$

The main reason for selecting the fraction of used terms from the interval $[0.9, 0.999]$ is that these situations concern the most critical area of the Zipf distribution of the data. Below 0.9, the distribution is so flat that it is almost uniform, and, therefore, not really interesting. We excluded fractions above 0.999 from our results, since the only logical next fraction would have been 1.0. However, that case is handled quite differently by our experimental DBMS, since it then has no fragmentation at all, resulting in some practical problems to compare the outcome with the rest. Next to that, the case for 1.0 did not seem to have any additional impact on the overall conclusions of our model compared to the case for 0.999.

5.4.2 Results

After having computed α [Expr. 5.24] and β [Expr. 5.25] for both collections, we ran the 50 queries for each of the 11 choices of m (relative to n), resulting in 550 cases per collection. For each of these 550 cases we computed the exact fraction of retrieved tuples in C' via the formula for $sel_{\sigma_{measured}}$ [Expr. 5.17]. We also computed $sel_{\sigma_{estimated}}$ [Expr. 5.45] for each of these cases.

In Figure 5.14, we plotted for each of the 550 cases for the FT collection the points $(sel_{\sigma_{measured}}, sel_{\sigma_{estimated}})$. We also plotted the line where the points should be located ideally, in case that our model is perfect. Note that in that case $sel_{\sigma_{estimated}}$ should be equal to $sel_{\sigma_{measured}}$.

In a similar manner, Figure 5.15 and Figure 5.16 show the results for the LATIMES and CR collection, respectively.

As is obvious, for all three collections, the point clouds are nicely arranged along the ideal line, demonstrating that our model indeed performs quite well.

5.5 Concluding remarks

In this chapter, we have presented a selectivity model that estimates the relative number of entries in the term frequency relation, TFstats, that satisfy a query consisting terms with an a priori known distribution. The entries in the term frequency relation, which represents an inverted list, are grouped by term and ordered on their corresponding document frequency. These document frequencies are stored per term in a relation called DFstats.

Our model only uses two parameters, α and β, which need to be determined before the model can be used. These two parameters are determined by some simple computation on the data distribution of the inverted list, which can be efficiently handled in a single table scan. Since updates are relatively rare in

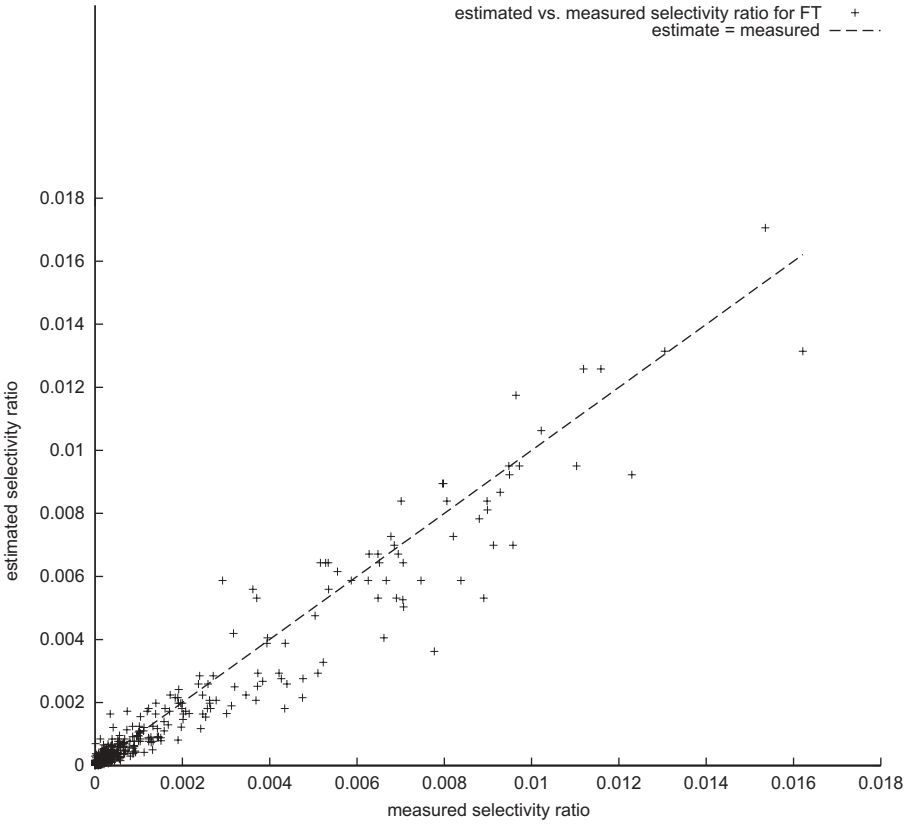

Figure 5.14: Estimated selectivity ratio vs. measured selectivity ratio [FT collection]

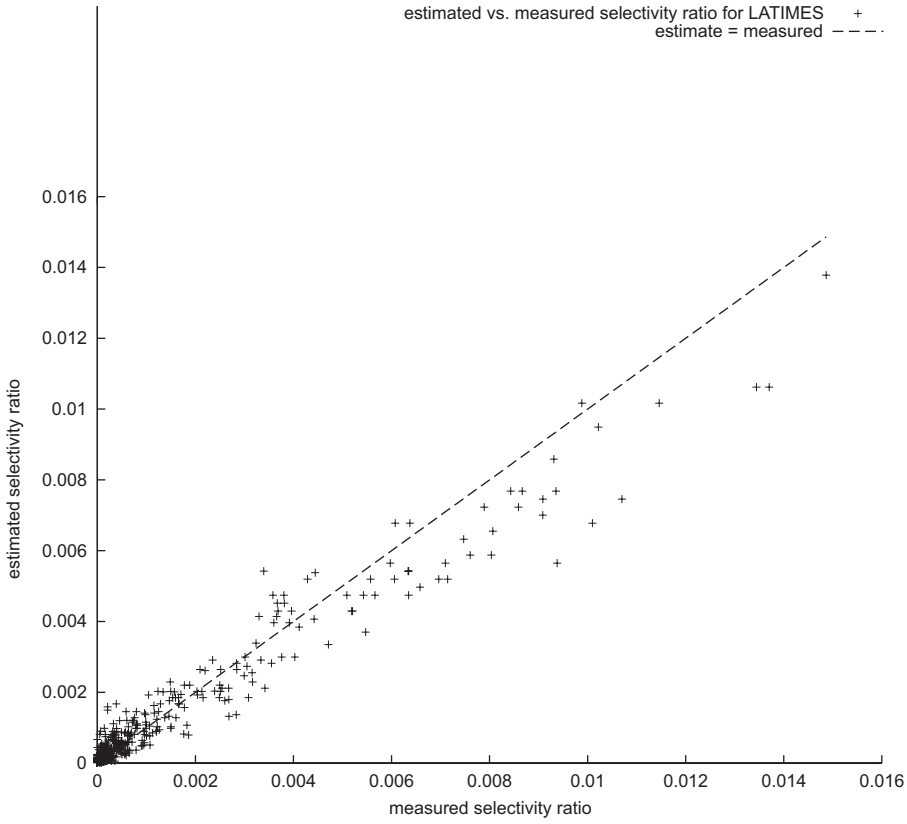

Figure 5.15: Estimated selectivity ratio vs. measured selectivity ratio [LATIMES collection]

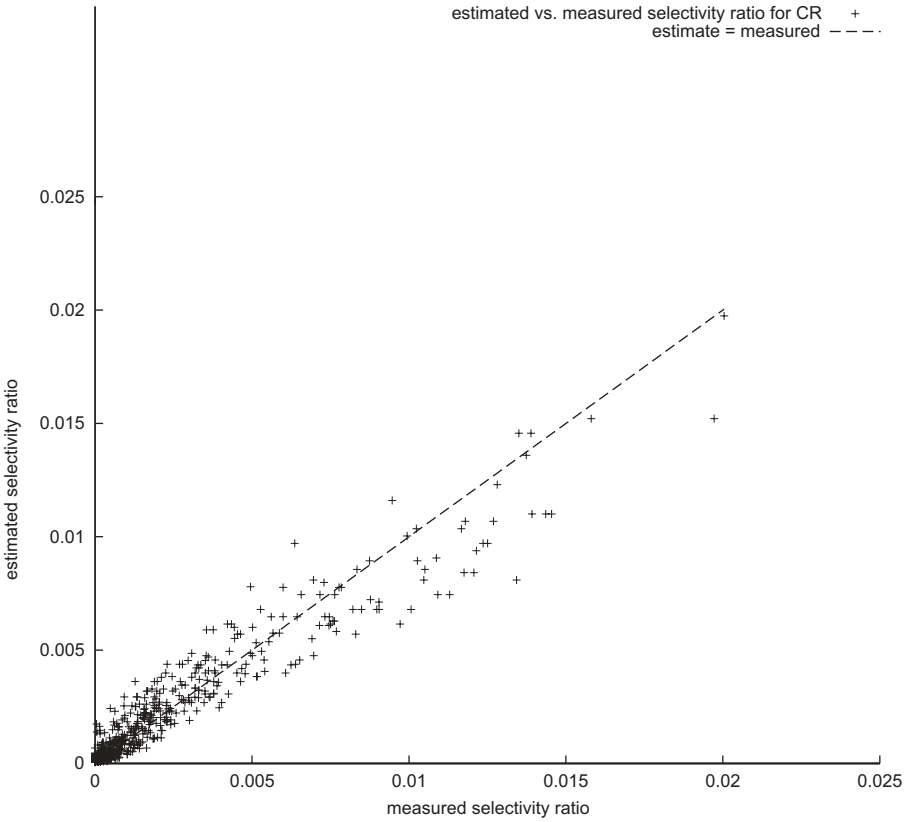

Figure 5.16: Estimated selectivity ratio vs. measured selectivity ratio [CR collection]

the context of advanced database applications such as information retrieval, updating of the parameters is not a big problem. Next to that, the parameters can easily be updated incrementally.

We have demonstrated the accuracy of our model by testing it for three information retrieval databases. The results show that the estimated selectivity values match the real selectivity value, measured during these tests, quite well.

We can conclude that our model does meet the three requirements we stated in Chapter 3: it's fast, accurate, and takes our horizontal fragmentation into account.

These results provide a clear, but partial, answer to the research question of interest in this chapter:

Q1 *Can we estimate selectivity as a function of the used fragments?*

We only looked at the size distribution of a two fragment case. For this case the answer is clearly positive.

However, we think our model easily can be generalized to the case of more fragments, with similarly good accuracy. Suppose we are interested in the selectivity for an arbitrary fragment, ranging from the terms m' to m'' (where, of course, $m < m' < m'' \leq n$). We could then throw away the first $m'-1$ terms from the DFstats relation, and the corresponding tuples from the TFstats relation, and renumber the terms starting from 1 again: $t_{m'}$ becomes t_1, $t_{m'+1}$ becomes t_2, etcetera. Then we can use our original model again, this time for the terms with indices 1 till $m''-m'+1$. We did not perform the corresponding experiments due to time limitations. This is a topic for future work.

Our model is useful in advanced application domains such as data warehousing and information retrieval. In those domains the amount of data that needs to be stored becomes huge, requiring techniques like distribution, partitioning, and parallelizing. Also main-memory computing is often regarded as a technique that can help to improve the efficiency of resource utilization in those new domains. All these techniques can be facilitated by fragmentation. A tool needed to allow exploiting fragmentation in an effective way, is a notion of the costs related to it. Our selectivity model is an attempt to meet this need.

References

[BCBA01] H.E. Blok, R.S. Choenni, H.M. Blanken, and P.M.G. Apers, *A selectivity model for fragmented relations in information retrieval*,

Technical Report 01-02, Centre for Telematics and Information Technology (CTIT), University of Twente, Enschede, The Netherlands, January 2001.

Chapter 6

Estimating quality

6.1 Introduction

In the field of information retrieval (IR), not only the need exists for a system with high precision and recall values, i.e., high retrieval quality (concerning the information need of a user), but also with a low response time. However, in general, the less time we spend on processing our document collections, the worse our precision and recall values will be.

In the field of database technology we have a similar problem. Not in the area of query processing, as the 'quality' of the results always has to be 100%, but in the area of query optimization. Query optimizers do not search for an optimal query plan, which typically takes far too much time. Instead, query optimizers search for a good query plan that can be found rather quickly.

This chapter is devoted to the quantitative aspects of the trade-off between response time and output quality in the context of information retrieval. We present the results how to extrapolate the properties of the trade-off obtained on a certain data set, to another data set which was not involved in the 'train-

This chapter is an edited version of a previously published paper [BHC+01a]:

H.E. Blok, D. Hiemstra, R.S. Choenni, F.M.G. de Jong, H.M. Blanken, and P.M.G. Apers, *Predicting the cost-quality trade-off for information retrieval queries: Facilitating database design and query optimization*, in Tenth International Conference on Information and Knowlegde Management (ACM CIKM'01), November 2001.

An earlier, extended version of this paper is also available as CTIT Technical Report [BHC+01b].

ing' of the trade-off parameters.

Recall our second research question:

Q2	*Can we estimate the quality behavior as a function of the relative size of the used fragments?*

and the three requirements we stated for our quality model in Chapter 3: fast in its use at query optimization time, should be able to generalize, and able to exploit and deal with the special properties of our special choice of horizontal fragments.

To answer an IR query, tuples of the TFstats and DFstats relations are evaluated, or more precisely, tuples of the fragments of the TFstats and DFstats relations are evaluated. Evaluating all the fragments containing query terms slows down responsiveness. Neglecting fragments of the DFstats relation, and correspondingly of the TFstats relation, will improve the response time but will decrease the quality of the output of a query. Given the imprecise nature of IR queries, insight in the trade-off between response time and quality is useful in building IR systems but also for the user. The user can set threshold values for these parameters, which is a way to express what quality or response time is acceptable. For example, on the one hand, a user may tell an IR system that the user wants an answer for a query within x time units. The system in its turn can tell the user what quality the user may expect if the response time should be within x time units. On the other hand, the user may also specify to the system what quality is acceptable for him/her.

The remainder of this chapter is structured as follows. First, we describe our approach in more detail in Section 6.2. Next, we introduce several mathematical definitions to set the formal context. In Section 6.4 we present a first model, both its mathematical construction and the experimental verification. Subsequently, we present a second model, which is a generalization of the first model, in Section 6.5. Again we include both the mathematical construction of the model and its experimental validation. We conclude this chapter in Section 6.6.

6.2 Approach

Our approach is based on the following three principles.

Firstly, we are interested in top-N queries, a very common form of IR queries. Therefore, we adopt the precision as the basis for our quality notion. To be more precise, we use the average precision, since this metric also includes

positional information and not just whether a certain fraction of the top-N is relevant. We do not use recall as our quality metric, since it is too much dependent on N. Note that the recall can always be increased up to 100% by choosing N large enough. In the degenerated case, i.e., choosing N equal to the number of documents in the collection, the system always retrieves all documents, including all relevant ones.

Secondly, we choose the fragments containing the terms with highest document frequencies to be the part we neglect during query processing, instead of just a random subset of the TFstats. These terms are considered to be the least discriminative and also provide the numerically least significant per-term contribution to the score of a document in our score function. This means that these terms usually have the least significant impact on the final ranking of an arbitrary document. Furthermore, these terms occupy the most space in the TFstats relation, so ignoring these terms delivers the largest cost reduction. In this chapter we look at the situation where we have two fragments, i.e., $|F| = 2$. We are interested in the effects of changing the size of the fragments. To be more precise, our model uses the fraction of terms with lower document frequencies, i.e., f_1, as its main steering variable. We refer to Chapter 4 for a more elaborate description of the underlying fragmenting approach.

And, thirdly, we do not try to estimate the quality behavior for each particular query but the quality behavior of the system in general. Therefore, we based our model on the mean of the average precision!mean over a set of queries, instead of just the average precision for a single query.

Given these three choices, we first constructed a model for the simple case. This model can be trained, using the least mean square (LMS) method, on the same collection as one wants to predict the quality for. We call this model, the *first order model*. Next, we present a generalized, *second order*, model that uses the LMS method on the trained parameters of the first order model of a set of training collection to estimate the parameters for the first order model of the test collection. This estimated first order model allows us to predict the quality behavior for that test collection.

This brings us to the main reason for our interest in the cost-quality trade-off in general and its application in integrating IR seamlessly into a database environment as we already addressed in Chapter 3. In database query processing, one prefers to work set-at-a-time instead of element-at-a-time. Therefore, traditional IR top-N optimization techniques are sometimes not easy to incorporate in a DBMS. The results of the previous chapter show that it would be beneficial to avoid the overhead in administrative work resulting from IR top-N optimization techniques. The ability to predict during query optimization which parts of the data set can be ignored — given the knowledge of the related quality implications — would therefore be very interesting. In practice

one usually has no significant quality information available for the entire data set, say the document collection statistics in a web search engine. However, for a small (sub)set one might be able to get good relevance judgments given a certain set of representative queries, for instance, the WebTREC[1] plus accompanying queries and relevance judgments. So, it would be useful if one could use this well-documented data set to train a quality model and then transfer the obtained properties to the general case. Our second order model provides precisely these generalization properties.

As we saw in Chapter 2, in the IR-field, some research has been done on the retrieval cost-effectiveness trade-off. However, we go somewhat further by generalizing quality properties obtained on a given set of collections to another collection, as described above.

In database research, the area of probabilistic top-N query optimization closely resembles the basic idea presented in this chapter, with the restriction that in that research only the optimization and query evaluation algorithm are probabilistic in a certain sense but the answers are still deterministic. As holds for all database query optimization in general, top-N optimization [CK98, FSGM+98, CG99] tries to prune the search space as quickly as possible. In probabilistic query optimization, one tries to guess which parts of the search space are highly unlikely to be of importance to the final answer, so one can ignore these parts as soon as possible. In traditional database probabilistic top-N optimization, a so called restart of the query processing is required when one detects that the search space has been limited too much. In the IR case, we cannot really detect that the search space has been limited too much, since the absolute correctness of the outcome of a query is not defined. We can only try to estimate what the quality implications are of limiting the search space.

6.3 Definitions

In this section, we introduce a whole series of mathematical definitions to allow better formalization of the approach later on. The first, and main, part of this section concerns the definitions of the datasets, the query set, and the ranking and quality functions. In the second part, we introduce a denotation to facilitate the estimation of several variables and parameters in the remainder of this chapter. We also introduce a relative error metric in order to get a notion of how well certain estimates approximate their measured counterparts.

[1] The WebTREC collection consists of a 10 and 100 GB document collection obtained from the World Wide Web. Like the normal TREC collections, these collections are used to evaluate retrieval systems.

Figure 6.1 contains the definitions of the set of collections $\mathcal{C}$, denoted by their respective names. To facilitate the discussion of the training and testing of our model in the remainder of this chapter we also introduce the notion of a $\mathcal{C}^{train}$ and a $\mathcal{C}^{test}$. How these subsets are constructed is described below.

In this chapter we look at the situation where we have two fragments, so $|F| = 2$. To simplify the notations in this chapters we introduce the fragmenting coefficient f, which we use instead of f_1. We limit the possible values of f to those in F_1, though in theory one could take any $0 \leq f \leq 1$. For convenience sake, we introduce the following terminology:

Definition 6.1 (f fragmented) *An f fragmented collection C is a collection split into two fragments, for which the first fragment contains $f \cdot 100\%$ of the terms of C with lowest document frequency and only this first fragment is taken into consideration during query evaluation (i.e., ranking).*

scxplquality!model!test collections

$$\mathcal{C} \equiv \{\text{FR94}, \text{CR}, \text{FBIS}, \text{FT}, \text{LATIMES}\}, \quad C \in \mathcal{C} \tag{6.1}$$
$$\mathcal{C}^{train} \subset \mathcal{C}, \text{ a set of training collections} \tag{6.2}$$
$$\mathcal{C}^{test} \subset \mathcal{C}, \text{ a set of test collections} \tag{6.3}$$
$$F_1 \equiv \{0.9, 0.925, 0.94, 0.95, 0.96, 0.97, 0.98, 0.985, 0.99, 0.995, 0.999\} \tag{6.4}$$
$$f \in F_1 \tag{6.5}$$

Figure 6.1: Dataset and fragmenting definitions

For each collection $C \in \mathcal{C}$ (f fragmented, where applicable), we define several statistics and data structures (Figure 6.2):

terms We distinguish n_C unique terms after stemming and stopping (Expr. 6.6 and 6.7). Note, that we have numbered the terms on ascending document frequency, which we can do without loss of generality (also see Expr. 6.14). In an f fragmented collection C, we only use the $m_C = f \cdot n_C$ first terms (Expr. 6.10).

documents We model documents as lists of terms (Expr. 6.8 and 6.9).

term frequency For each unique document term pair, we administer the number of times that that term occurs in that particular document (Expr. 6.11 and 6.12). We also compute a normalized version (Expr. 6.13) within the $[0, 1]$ range to facilitate a mathematically better founded score function (see below).

document frequency For each term, we store the number of documents it occurs in (Expr. 6.14 and 6.15). As for the term frequency, we also

$$t_{C\,i} \equiv \text{a term in collection } C \tag{6.6}$$
$$T_C \equiv [t_{C\,1}, t_{C\,2}, t_{C\,3}, \ldots, t_{C\,i}, \ldots, t_{C\,n_C}] \tag{6.7}$$
$$d_{C\,j} \equiv \text{a document in collection } C, [t_i | t_i \text{ is 'a term'}] \tag{6.8}$$
$$D_C \equiv \{d_{C\,1}, d_{C\,2}, d_{C\,3}, \ldots, d_{C\,j}, \ldots, d_{C|D_C|}\} \tag{6.9}$$
$$T_{f\,C} \equiv [t_{C\,1}, t_{C\,2}, t_{C\,3}, \ldots, t_{C\,i}, \ldots, t_{C\,m_C}], \text{ where } m_C \leq n_C \text{ and } m_C \equiv f \cdot n_C \equiv |T_{f\,C}| \tag{6.10}$$
$$tf_{C\,ij} \equiv \text{term frequency for term } t_{C\,i} \text{ in document } d_{C\,j} \tag{6.11}$$
$$TF_C \equiv \{(t_{C\,i}, d_{C\,j}, tf_{C\,ij}) | i \in \{1, \ldots, n_C\}, j \in \{1, \ldots, |D_C|\}\} \tag{6.12}$$
$$ntf_{C\,ij} \equiv \frac{tf_{C\,ij}}{\sum_{i=1}^{n_C} tf_{C\,ij}} \tag{6.13}$$
$$df_{C\,i} \equiv |\{d_{C\,j} | (t_{C\,i}, d_{C\,j}, tf_{C\,ij}) \in TF_C\}| \tag{6.14}$$
$$DF_C \equiv [df_{C\,i} | \forall t_{C\,i} \in T_C] \tag{6.15}$$
$$ndf_{C\,i} \equiv \frac{df_{C\,i}}{\tau_C}, \text{ where } \tau_C \equiv \sum_{i=1}^{n_C} df_{C\,i} \tag{6.16}$$

Figure 6.2: Collection statistic definitions

introduce a normalized version of the document frequency (Expr. 6.16). As mentioned, we numbered the terms on ascending document frequency, so: $df_{C\,i+1} \geq df_{C\,i}$, $\forall i \in \{1, \ldots, n_C - 1\}$.

Note that these definitions differ slightly in denotation from those in the previous chapters.

Table 6.1: TREC-6 document collection statistics (as present in our system after stemming and stopping, also see Table 3.1)

Collection	n_C	τ_C	$\lvert D_C \rvert$	map_C
FR94	91,807	7,700,575	55,505	0.1780
CR	69,433	5,189,218	27,921	0.2370
FBIS	202,940	17,385,471	130,471	0.2529
FT	175,593	26,544,084	210,158	0.3097
LATIMES	176,853	19,565,029	131,890	0.2878

For the interested reader, we listed some key statistics for all collections in $\mathcal{C}$ in the Table 6.1.

Next to data sets we also need queries (Figure 6.3). For convenience sake, we model our queries as sets, though in reality the queries can contain multiple occurrences of the same term. However, not modeling the queries with this

$$Q \equiv \{t_i | t_i \in T_C\} \quad (6.17)$$
$$\mathcal{Q} \equiv \{Q \in \mathcal{Q}\}, \text{ the set of TREC topics/queries} \quad (6.18)$$
$$\mathcal{Q}^{train} \subset \mathcal{Q}, \text{ the set of training queries} \quad (6.19)$$
$$\mathcal{Q}^{test} \subset \mathcal{Q}, \text{ the set of test queries} \quad (6.20)$$

Figure 6.3: Query related definitions

capability[2] is not a problem given our context. Again, we already distinguish the notion of training and test queries but defer the actual construction of these query sets to the experimental sections below.

Since we need some sub expressions of the score function used in our system (sometimes also known as ranking function), we have listed the relevant definitions in Figure 6.4.

$$score_{C\,ij} \equiv 1 + \frac{ntf_{C\,ij}}{ndf_{C\,i}} \quad (6.21)$$
$$score_{C\,j}(Q) \equiv \sum_{i \in V} \log\left(score_{C\,ij}\right), \text{ where } V = \{i | t_{C\,i} \in Q\} \quad (6.22)$$
$$score_{C\,fj}(Q) \equiv \sum_{i \in V} \log\left(score_{C\,ij}\right), \text{ where } V = \{i | t_{C\,i} \in Q \wedge i \leq m_C\} \quad (6.23)$$
$$p_{C\,i} \equiv ndf_{C\,i} \quad (6.24)$$
$$Escore_{C\,ij} \equiv \sum_{i=1}^{n_C} score_{C\,ij} \cdot p_{C\,i} \quad (6.25)$$
$$Escore_{C\,fij} \equiv \sum_{i=1}^{m_C} score_{C\,ij} \cdot p_{C\,i} \quad (6.26)$$
$$nEscore_{C\,fij} \equiv \frac{Escore_{C\,fij}}{Escore_{C\,ij}} \quad (6.27)$$

Figure 6.4: Ranking related definitions

Expression 6.21 is the score contribution of a term i for a document j, as used by our system to rank the documents. The score contribution is motivated by the use of language models for information retrieval, a recently developed approach to information retrieval that performs among the best approaches in experimental evaluations [PC98, HK98].

Expression 6.22 defines the score of a document given a query Q by summing the logarithm of $score_{C\,ij}$ for each i. The resulting algorithm is a member of the family of $tf \cdot idf$ term weighting algorithms, which are used in many approaches to ranked retrieval [SB88]. For the relation between language mod-

[2] For instance, modeling the queries as lists would have allowed multiple occurrences of the same query term, but we refrained from that for simplicity.

eling algorithms and the traditional $tf \cdot idf$ term weighting algorithms, we refer to [Hie00]. Expression 6.23 is the variant of the score function used in a fragmented case.

For several reasons we need a notion of the probability that a certain term is term i (Expression 6.24). The main reason to choose the probability this way follows directly from the Zipfian behavior of natural language [Zip49]. Based on this probability, we can define the estimated values in Expressions 6.25 and 6.26. Expression 6.27 is a normalized version of Expression 6.26. The use of these expressions is clarified in more detail below.

Finally, Figure 6.5 shows the quality metrics we use.

$$ap_C(Q) \equiv \text{average precision for query } Q \text{ on collection } C \in \mathcal{C} \tag{6.28}$$

$$ap_{C\,f}(Q) \equiv \text{average precision for query } Q \text{ on } f \text{ fragmented collection } C \in \mathcal{C} \tag{6.29}$$

$$map_C \equiv \frac{\sum_{Q \in \mathcal{Q}} ap_C(Q)}{|\mathcal{Q}|} \tag{6.30}$$

$$map_{C\,f} \equiv \frac{\sum_{Q \in \mathcal{Q}} ap_{C\,f}(Q)}{|\mathcal{Q}|} \tag{6.31}$$

$$nmap_{C\,f} \equiv \frac{map_{C\,f}}{map_C} \tag{6.32}$$

Figure 6.5: Quality metric definitions

We base our aggregated quality measure on the average precision. We refer to Chapter 2 for an explanation of this quality metric.

Since we are not interested in the quality behavior of just a query in particular, we aggregate over a set of queries (Expressions 6.30 and 6.31). Furthermore, we are only interested in relative changes in this aggregated quality measure, so we use a normalized variant (Expression 6.32).

Besides these model related definitions, we introduce the following denotation:

Denotation 6.1 *We denote the estimated counterpart of a certain variable or parameter x by $\overline{x}$.*

The main reason for introducing this denotation is the fact that we estimate several parameters and variables in the remainder of this chapter, so we need the proper means to distinguish between the actual parameter/variable and its estimated counterpart.

Furthermore, we use this denotation recursively. For example, the estimated value of an (already) estimated value $\overline{x}$ is written as $\overline{\overline{x}}$.

To provide a good notion of the error of the models we also define a relative error measure.

Definition 6.2 (Relative error) *The relative error $\epsilon_{\overline{x}}$ of an estimated value $\overline{x}$ of a variable or parameter x is defined as:*

$$\epsilon_{\overline{x}} \quad \equiv \quad \frac{\overline{x} - x}{x}$$

for $x \neq 0$.

The advantage of this metric is that it can be *pushed through* the normalization of a relative entity.

Lemma 6.1 (Relative error transparency) *Given a relative entity y, based on an entity x that has been normalized by dividing by a given number N:*

$$y \quad \equiv \quad \frac{x}{N}$$

Then $\epsilon_{\overline{y}} = \epsilon_{\overline{x}}$.

Proof Substitute $y = \frac{x}{N}$ in $\epsilon_{\overline{y}}$ (using Definition 6.2):

$$\begin{aligned}
\epsilon_{\overline{y}} &= \frac{\overline{y} - y}{y} \\
&= \frac{\overline{\left(\frac{x}{N}\right)} - \frac{x}{N}}{\frac{x}{N}} \\
&= \frac{N\overline{\left(\frac{x}{N}\right)} - x}{x}
\end{aligned}$$

Since N is a constant we can write:

$$\begin{aligned}
\epsilon_{\overline{y}} &= \frac{N\frac{\overline{x}}{N} - x}{x} \\
&= \frac{\overline{x} - x}{x} \\
&= \epsilon_{\overline{x}}
\end{aligned}$$

□

For example, Lemma 6.1 holds for the relative error $\epsilon_{\overline{nmap}_{C\,f}}$, because $\overline{nmap}_{C\,f}$ is the estimator for $nmap_{C\,f}$, which is defined as $\frac{map_{C\,f}}{map_C}$ (see Expression 6.32 in Figure 6.5). So, a relative error $\epsilon_{\overline{nmap}_{C\,f}} = e$ also means that $\epsilon_{\overline{map}_{C\,f}} = e$ as well.

6.4 First order approach

In this section, we construct a model to predict $nmap_{C\,f}$ for a given collection $\mathcal{C}$ that is f fragmented. We train the model using a given set of queries $\mathcal{Q}^{train}$. Once the model has been trained, we test its accurateness on a set of test queries $\mathcal{Q}^{test}$.

This section has been divided in three parts. First, we introduce our theoretical (i.e., mathematical) approach and use it to derive an elegant regression model. Then, we present the experimental setup used to validate this model. Finally, we present the results of these experiments.

6.4.1 Model

As stated above, we are interested in predicting $nmap_{C\,f}$ when we know the value of f. Since we do not have any illusion in solving the general problem of information retrieval, i.e., ranking documents perfectly, our only lead is taking a closer look at our ranking method. Of course, no ranking method is perfect, neither is the one we use. However, since a state of the art ranking performs clearly better than just a random ordering of some documents, it clearly does something in the right direction.

The $ap_{C\,f}(Q)$ for an arbitrary query Q can only change if, as a consequence of decreasing f, a document that correctly appears in the top-N swaps places with a document that should not have been retrieved. In a more formal manner, let us assume we have two documents j_i and j_2 with document scores $score_{C\,j_1}$ and $score_{C\,j_2}$, respectively, such that $score_{C\,j_1}(Q) \geq score_{C\,j_2}(Q)$, for an arbitrary query Q. The problem of interest then boils down to the question what should happen to f to make $score_{C\,f\,j_1}(Q) < score_{C\,f\,j_2}(Q)$, for that same query Q. Trying to come up with an analytical solution for this question appears to be very difficult. However, the main player in the ranking is the score contribution of each individual term i for an arbitrary document j, $score_{C\,ij}$. The remainder of this chapter demonstrates that taking $score_{C\,ij}$ as the basis for estimating $nmap_{C\,f}$ works out quite well, and in contrast with using $score_{C\,fj}$ is analytically manageable.

Since we are not interested in actual $score_{C\,ij}$ values but only in its general behavior for 'average' queries and how it degrades for decreasing f we use $nEscore_{C\,fij}$ instead. By taking the expected value instead of the actual value, we abstract from the special effects of just a particular query, likewise we take the mean of the $ap_{C\,f}(Q)$, i.e., $map_{C\,f}$. Dividing by $Escore_{C\,ij}$ normalizes the range between 0 and 1 abstracting from the actual numerical range. Similarly, we normalize our quality measure as well, resulting in $nmap_{C\,f}$ as the actual

quality measure instead of $ap_{C\,f}(Q)$.

Now, let us assume any change in $nEscore_{C\,f\,ij}$ proportionally effects $nmap_{C\,f}$, in other words:

$$\eta_{C\,0} + \eta_{C\,1} nEscore_{C\,f\,ij} \quad = \quad nmap_{C\,f} \tag{6.33}$$

This leaves us with the question what the influence of f is on $nEscore_{C\,f\,ij}$. The remainder of this subsection concerns the actual construction of this model for $nmap_{C\,f}$ with f as explaining variable.

We start with assuming that the $df_{C\,i}$ values are distributed according to Zipf [Zip49]. We also assumed that the terms are ordered ascendingly on their frequency, so: $df_{C\,i} \leq df_{C\,i+1}$. The 'official' Zipf's law – 'index' × 'frequency' = 'constant' – assumes a descending order on frequency. Combining this information into one formula gives:

$$df_{C\,i} \quad \equiv \quad \frac{a_C}{n_C - i + 1} \tag{6.34}$$

where a_C is the 'constant' in Zipf's law, for a certain collection C.

Using Expression 6.34 and the fact that a sum over many small steps can be approximated by an integral we can now rewrite definition 6.26:

$$\begin{aligned}
Escore_{C\,f\,ij} &= \sum_{i=1}^{m_C} score_{C\,ij} \cdot p_{C\,i} \\
&= \sum_{i=1}^{m_C} score_{C\,ij} \cdot ndf_{C\,i} \\
&\approx \int_{i=1}^{m_C} score_{C\,ij} \cdot ndf_{C\,i}\, di \\
&= \int_{i=1}^{m_C} \left(1 + \frac{ntf_{C\,ij}}{ndf_{C\,i}}\right) \cdot ndf_{C\,i}\, di \\
&= \int_{i=1}^{m_C} ndf_{C\,i} + ntf_{C\,ij}\, di \\
&= \int_{i=1}^{m_C} \frac{a_C}{(n-i+1)\tau_C} + ntf_{C\,ij}\, di.
\end{aligned} \tag{6.35}$$

Similarly, we can rewrite definition 6.25:

$$\begin{aligned} Escore_{C\,ij} &= \sum_{i=1}^{n_C} score_{C\,ij} \cdot p_{C\,i} \\ &\approx \int_{i=1}^{n_C} score_{C\,ij} \cdot ndf_{C\,i}\; di \\ &= \int_{i=1}^{n_C} \frac{a_C}{(n-i+1)\tau_C} + ntf_{C\,ij}\; di \end{aligned} \tag{6.36}$$

Next, we substitute Expressions 6.35 and 6.36 in definition 6.27:

$$nEscore_{C\,fij} = \frac{\int_{i=1}^{m_C} \frac{a_C}{(n-i+1)\tau_C} + ntf_{C\,ij}\; di}{\int_{i=1}^{n_C} \frac{a_C}{(n-i+1)\tau_C} + ntf_{C\,ij}\; di}. \tag{6.37}$$

Evaluating each $ntf_{C\,ij}$ would be comparable in effort to evaluating a query, which we either cannot do during database design or do not want during query optimization. Since we do not want to do such expensive database accesses, we do not know the value of each $ntf_{C\,ij}$. Furthermore, for our quality model we are only interested in the global change of the document scores, not in the change in scores of any specific document. Therefore, the normalized term frequency $ntf_{C\,ij}$ might be approximated by γ_C, which is the average normalized term frequency of the document-term pairs in the database. Given this assumption Expression 6.37 reduces to:

$$nEscore_{C\,fij} \approx \frac{\int_{i=1}^{m_C} \frac{a_C}{(n-i+1)\tau_C} + \gamma_C\; di}{\int_{i=1}^{n_C} \frac{a_C}{(n-i+1)\tau_C} + \gamma_C\; di} \tag{6.38}$$

This effectively reduces our variant of $tf \cdot idf$ weighting to a variant of idf weighting, which was motivated by a Zipf-like distribution in [SJ72]. Although we explicitly derive equation 6.38 from our language modeling ranking algorithm, we hypothesize that the same approximation holds for any term weighting algorithm that includes an idf component.

Evaluation of the integral parts in Expression 6.38 and some further rewriting

results in:

$$\begin{aligned} & nEscore_{C\,fij} \\ &\approx -\frac{-\gamma_C\tau_C\, m_C + a_C \log(n_C - m_C + 1) + \gamma_C\tau_C - a_C \log n_C}{\gamma_C\tau_C\, n_C - \gamma_C\tau_C + a_C \log n_C} \\ &= \frac{\gamma_C\tau_C\, m_C}{\aleph} - \frac{a_C \log(n_C - m_C + 1)}{\aleph} - \frac{\gamma_C\tau_C}{\aleph} + \frac{a_C \log n_C}{\aleph} \end{aligned} \tag{6.39}$$

where:

$$\aleph = \gamma_C\tau_C\, n_C - \gamma_C\tau_C + a_C \log n_C$$

Next, we substitute fn_C for m_C and simplify the expression a bit, using the knowledge that n_C is quite large:

$$\begin{aligned} & nEscore_{C\,fij} \\ &\approx \frac{\gamma_C\tau_C\, fn_C}{\aleph} - \frac{a_C \log(n_C - fn_C + 1)}{\aleph} - \frac{\gamma_C\tau_C}{\aleph} + \frac{a_C \log n_C}{\aleph} \\ &\approx \frac{\gamma_C\tau_C\, fn_C}{\aleph} - \frac{a_C \log(n_C(1-f))}{\aleph} - \frac{\gamma_C\tau_C}{\aleph} + \frac{a_C \log n_C}{\aleph} \\ &= \frac{\gamma_C\tau_C\, n_C}{\aleph} f - \frac{a_C}{\aleph}\log(1-f) - \frac{a_C \log n_C}{\aleph} - \frac{\gamma_C\tau_C}{\aleph} + \frac{a_C \log n_C}{\aleph} \\ &= \frac{\gamma_C\tau_C\, n_C}{\aleph} f - \frac{a_C}{\aleph}\log(1-f) - \frac{\gamma_C\tau_C}{\aleph} \end{aligned} \tag{6.40}$$

Since, $\aleph$ depends mainly on τ_C and n_C, the second term in this sum has negligible influence, reducing the basic expression to:

$$\begin{aligned} nEscore_{C\,fij} &\approx \frac{\gamma_C\tau_C\, n_C}{\aleph} f - \frac{\gamma_C\tau_C}{\aleph} \\ &= \varphi_{C\,0} f + \varphi_{C\,1} \end{aligned} \tag{6.41}$$

where:

$$\begin{aligned} \varphi_{C\,0} &= \frac{\gamma_C\tau_C\, n_C}{\aleph}, \\ \varphi_{C\,1} &= -\frac{\gamma_C\tau_C}{\aleph} \end{aligned}$$

This simplification is supported by the following mathematical limit analysis:

$$\begin{aligned}
&\lim_{n_C\to\infty}\lim_{\tau_C\to\infty} nEscore_{C\,fij}\\
&\quad\approx\quad \lim_{n_C\to\infty}\lim_{\tau_C\to\infty} \frac{\gamma_C\tau_C\,fn_C}{\aleph} - \frac{a_C\log(n_C - fn_C + 1)}{\aleph}\\
&\qquad\qquad - \frac{\gamma_C\tau_C}{\aleph} + \frac{a_C\log n_C}{\aleph}\\
&\quad=\quad \lim_{n_C\to\infty}\frac{fn_C - 1}{n_C - 1}\\
&\quad=\quad f
\end{aligned} \tag{6.42}$$

For convenience sake, we rewrite Expression 6.41 into the following form:

$$nEscore_{C\,fij} \quad\approx\quad \gamma_C\,(\varphi'_{C\,0}f + \varphi'_{C\,1}) \tag{6.43}$$

where:

$$\varphi'_{C\,0} \quad=\quad \frac{\tau_C\,n_C}{\aleph}, \tag{6.44}$$

$$\varphi'_{C\,1} \quad=\quad -\frac{\tau_C}{\aleph} \tag{6.45}$$

Substituting this rewritten form into Expression 6.33, our relation of interest between $nEscore_{C\,fij}$ and $nmap_{C\,f}$, gives:

$$\begin{aligned}
&\eta_{C\,0} + \eta_{C\,1}nEscore_{C\,fij} = nmap_{C\,f}\\
&\quad\Rightarrow\quad \eta_{C\,0} + \eta_{C\,1}\,(\varphi_{C\,0}f + \varphi_{C\,1}) \approx nmap_{C\,f}\\
&\quad\Rightarrow\quad \eta_{C\,0} + \eta_{C\,1}\gamma_C\,(\varphi'_{C\,0}f + \varphi'_{C\,1}) \approx nmap_{C\,f}\\
&\quad\Rightarrow\quad \eta_{C\,0} + \eta_{C\,1}\gamma_C\varphi'_{C\,1} + \eta_{C\,1}\gamma_C\varphi'_{C\,0}f \approx nmap_{C\,f}\\
&\quad\Rightarrow\quad \psi_{C\,0} + \psi_{C\,1}f \approx nmap_{C\,f}
\end{aligned} \tag{6.46}$$

where:

$$\psi_{C\,0} \quad=\quad \eta_{C\,0} + \eta_{C\,1}\gamma_C\varphi'_{C\,1} \tag{6.47}$$

$$\psi_{C\,1} \quad=\quad \eta_{C\,1}\gamma_C\varphi'_{C\,0} \tag{6.48}$$

Note that Expression 6.46 nicely fits the general observation that the less terms (lower f) one takes into account, the lower the answer quality (lower $nmap_{C\,f}$) one should expect.

6.4.2 Experimental setup

Since our model (Expression 6.46) is linear, we can use the LMS (least mean squares) method[3] to estimate the coefficients $\psi_{C\,0}$ and $\psi_{C\,1}$.

The queries we used, are the 50 retrieval queries, also known as topics in the IR field, from TREC-6. These queries range in length from 9 up to and including 61 terms with an average of 27 terms. This query length might seem unrealistically large since queries typically entered by a user consist of only a few terms. Note however, that many applications exist in which the user does not enter the actual query that is actually executed by the retrieval system. Examples of such situations are: automatic query expansion (used by certain relevance feedback and thesaurus exploiting techniques), or searching for similar texts given an example text (one might think of patent verification).

Since we need a training set $\mathcal{Q}^{train}$ and a test set $\mathcal{Q}^{test}$ of queries, we constructed two random subsets of $\mathcal{Q}$. Both subsets were constructed independently from each other by picking a query from one of the 50 queries in $\mathcal{Q}$ with a uniform probability of 60%. This, of course, does result in some overlap between $\mathcal{Q}^{train}$ and $\mathcal{Q}^{test}$, but since queries are drawn from the same 'virtual' pool in real world application we think this does not inflict a negative impact on the quality of the results. Also, 50 queries as total query pool is not very large so the larger the training and test sets, the less statistical noise we get on the resulting *nmap*, which is averaged over these sets.

The experimental training procedure we followed for each collection $C \in \mathcal{C}$ is described in Figure 6.6.

After training our model on a per-collection basis (i.e., determining $\overline{\psi}_{C\,0}$ and $\overline{\psi}_{C\,1}$), we tested the model using the procedure shown in Figure 6.7 (again, for each collection $C \in \mathcal{C}$).

For the interested reader, we have listed some key statistics of the used document collections in Table 6.1, including the actual map_C values.

6.4.3 Experimental results

In Figure 6.8, we plotted the estimated $nmap_{C\,f}$ (i.e., $\overline{nmap}_{C\,f}$) versus the measured $nmap_{C\,f}$. If our model were perfect, the estimated values would be equal to their corresponding measured value ($\overline{nmap}_{C\,f} = nmap_{C\,f}$). We also included this ideal line in the plot for reference.

[3] Note that, from a strict mathematical point of view, the use of the LMS method requires the error on $nmap_{C\,f}$ to be normally distributed. In this case we are not certain if this requirement is met, but we use the LMS method anyway, since it might still work very well in practice. Of course, in case of very bad experimental results this choice might have to be reconsidered.

Step 1 Produce a ranked top-1000 for each query $Q \in \mathcal{Q}^{train}$ on collection C in the normal manner (i.e., by taking all terms into account).

Step 2 Compute the average precision $ap_C(Q)$ for each of the queries of the previous step and compute map_C based on the $ap_C(Q)$ values, according to definition 6.30.

Step 3 Produce a ranked top-1000 for each query $Q \in \mathcal{Q}^{train}$ on collection C for each $f \in F$.

Step 4 Compute the average precision $ap_{C\,f}(Q)$ corresponding to each of the results of the previous step and compute $map_{C\,f}$ based on the $ap_{C\,f}(Q)$ values, according to definition 6.31.

Step 5 Compute $nmap_{C\,f}$ using definition 6.32.

Step 6 Compute $\overline{\psi}_{C\,0}$ and $\overline{\psi}_{C\,1}$ using the LMS method on Expression 6.46, given the computed $nmap_{C\,f}$.

Figure 6.6: First order training procedure (for a given collection $C \in \mathcal{C}$)

Step 1 Produce a ranked top-1000 for each query $Q \in \mathcal{Q}^{test}$ on collection C in the normal manner (i.e., by taking all terms into account).

Step 2 Compute the average precision $ap_C(Q)$ for each of the queries of the previous step and compute map_C based on the $ap_C(Q)$ values, according to definition 6.30.

Step 3 Produce a ranked top-1000 for each query $Q \in \mathcal{Q}^{test}$ on collection C for each $f \in F$.

Step 4 Compute the average precision $ap_{C\,f}(Q)$ corresponding to each of the results of the previous step and compute $map_{C\,f}$ based on the $ap_{C\,f}(Q)$ values, according to definition 6.31.

Step 5 Compute $nmap_{C\,f}$ using definition 6.32.

Step 6 Compute $\overline{nmap}_{C\,f}$ for each $f \in F$ using Expression 6.46, given $\overline{\psi}_{C\,0}$ and $\overline{\psi}_{C\,1}$.

Step 7 Compute the relative error between $\overline{nmap}_{C\,f}$ and $nmap_{C\,f}$ values using Definition 6.2.

Figure 6.7: First order test procedure (for a given collection $C \in \mathcal{C}$)

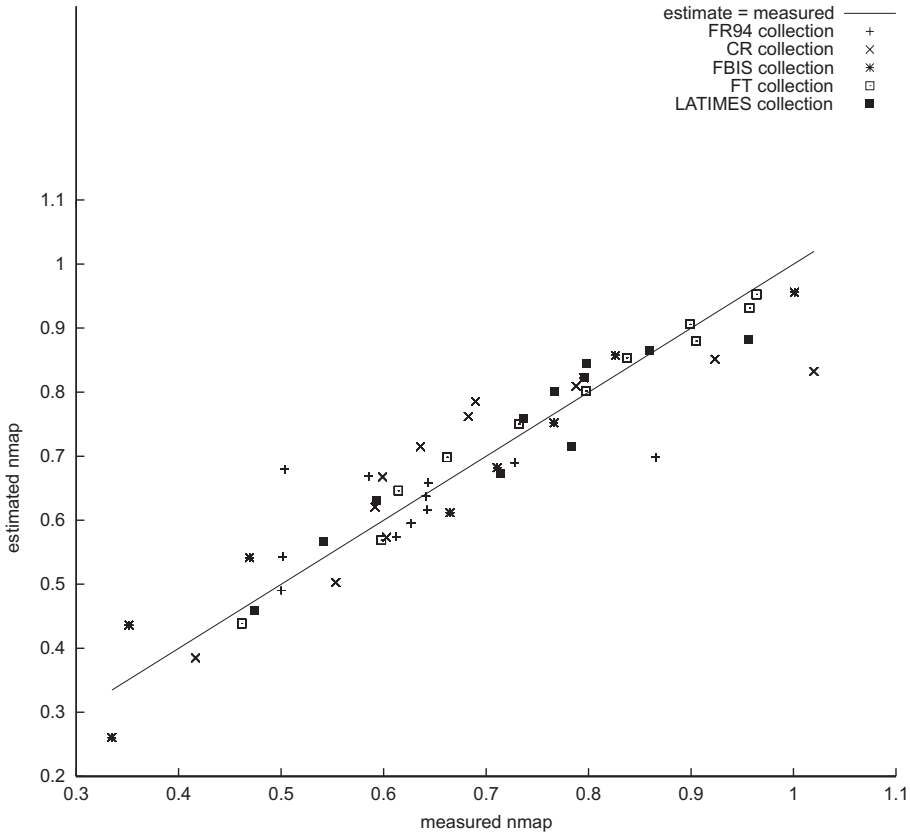

Figure 6.8: First order model test results, $\overline{nmap}_{C\,f}$ vs. $nmap_{C\,f}$, for all collections $C \in \mathcal{C}$

As one can see, all the data points are nicely grouped along the ideal line for all collections. This observation is supported by the relative errors plotted in Figure 6.9.

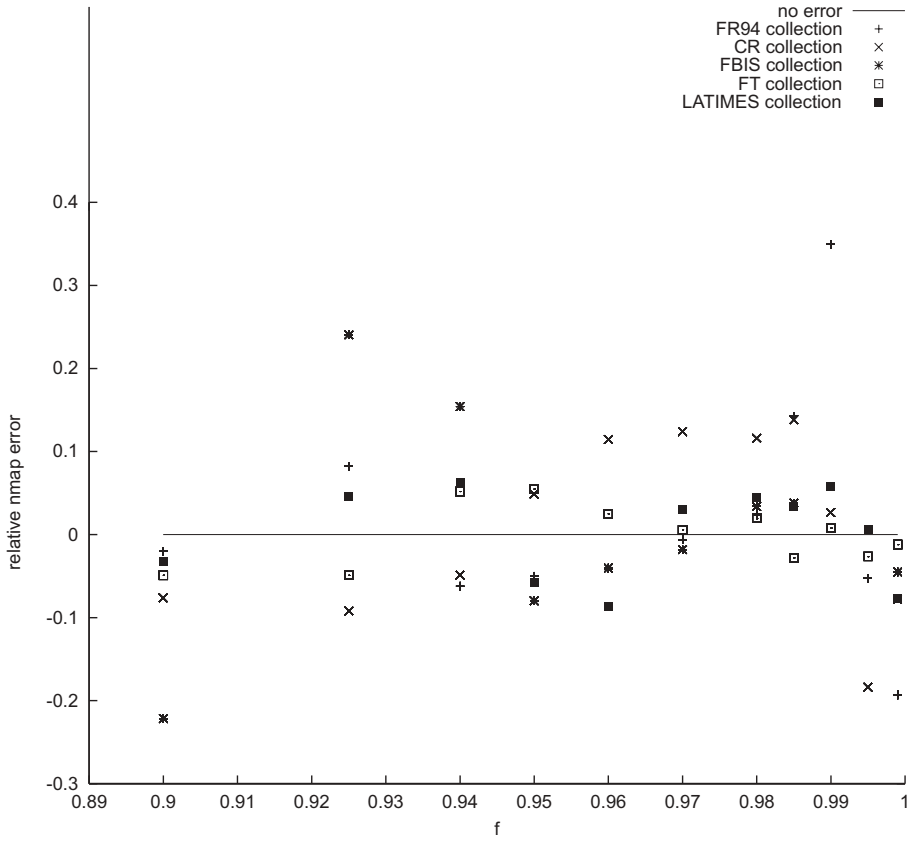

Figure 6.9: First order model relative error, $\epsilon_{\overline{nmap}_{C\,f}}$ vs. f, for all collections $C \in \mathcal{C}$

Note, both the values on the axes in Figure 6.8 are relative entities. This means they can range from 0 to 1, and in practice actually do occupy this entire range indeed. Theoretically this is also the case for the *ap* however in practice the *ap* hardly ever exceeds 0.4 for typical $tf \cdot idf$ based retrieval systems. Also note that a relative error $\epsilon_{\overline{nmap}_{C\,f}} = e$ corresponds to a relative error $\epsilon_{\overline{map}_{C\,f}} = e$ (see Lemma 6.1).

6.5 Second order approach

In the previous section, we trained the model parameters on the same collection we wanted to predict the *nmap* for. In reality one wants to predict the *nmap* for a collection for which one has no relevance judgments available. This means that it is impossible to measure the *nmap* in such a case, taking away the possibility to train on the same collection as one wants to predict the *nmap* for.

In other words, one would like to train the model parameters on some collection (or even collections) for which the *nmap* can be measured, and then use this trained model on another collection to predict *nmap* values.

In this section, we use two extra parameters, which also follow nicely from our theoretical model. These two parameters are used to add a second regression model to predict the model parameters of the previous model. At a first glance, this might seem very awkward, but we demonstrate that it is in fact a quite intuitive approach that also provides quite reasonable results.

This section has been divided in three parts, similar to the previous section. First, we use our mathematical model to derive an elegant second regression model as an extension to the basic first model. Then, we present the extensions to the original experimental setup used to validate this second order model. Finally, we present the results of these experiments.

6.5.1 Model

In a real world situation it is usually impossible to train the first order model on the same collection C as one wants to predict the quality for, due to the absence of $nmap_C$ and $nmap_{Cf}$ information on that collection C. So, one would like to train the model on some other collection $C' \neq C$.

Unfortunately, Expression 6.46 in Section 6.4 appears not to work in this case. A closer look learns that the estimated coefficients $\overline{\psi}_{C0}$ and $\overline{\psi}_{C1}$ differ significantly per collection. It appears that these coefficients contain collection dependent information, as is quite obvious from Expression 6.47 and 6.48.

In these two parameter definitions, we do not know γ_C, η_{C0}, and η_{C1}. Also, we do not know φ'_{C0} and φ'_{C1} exactly. But, we know that φ'_{C0} and φ'_{C1} are mainly determined by n_C and τ_C (see Expressions 6.44 and 6.45), so we approximate the formulas 6.47 and 6.48 by:

$$\omega_{00} + \omega_{01} \cdot \tau_C + \omega_{02} \cdot n_C = \psi_{C0} \quad (6.49)$$

$$\omega_{10} + \omega_{11} \cdot \tau_C + \omega_{12} \cdot n_C = \psi_{C1} \quad (6.50)$$

This completes the construction of our second order model, which clearly captures the collection dependencies in $\psi_{C\,0}$ and $\psi_{C\,1}$ (and therefore in $\overline{\psi}_{C\,0}$ and $\overline{\psi}_{C\,1}$).

Note that the Expressions 6.49 and 6.50 are linear whereas the Expressions 6.44 and 6.45 are not. We acknowledge that this might be a very crude approximation. The main reason to choose this approximation is the practical advantage in the estimation of the parameters, since we can use the LMS method. The use of two regression models in a nested manner is not an uncommon statistical technique given the type of situation we use it in (see [HATB98]). However, as approximation this model, of course, has to prove its usefulness in practice.

Recall from the definitions in Section 6.3 that $\tau_C = |TF_C|$ and $n_C = |T_C|$, meaning we only need to perform count operations to get this information. Since this requires practically no database accesses this fits in our requirement to use this model during database design and query optimization. Using these two cheap collection statistics, we can use the Expressions 6.49 and 6.50, once their own coefficients have been determined, to estimate $\overline{\psi}_{C\,0}$ and $\overline{\psi}_{C\,1}$.

In the remainder of this section we demonstrate that the two Expressions 6.49 and 6.50 indeed do allow training of our model on collections $C \in \mathcal{C}^{train}$ such that $\mathcal{C}^{train} \cap \mathcal{C}^{test} = \emptyset$.

6.5.2 Experimental setup

To evaluate our second order approach we extend our first order experimental setup as described in Subsection 6.4.2. We now also split up our set of collections $\mathcal{C}$ in a set of training collections $\mathcal{C}^{train}$ and test collections $\mathcal{C}^{test}$.

Due to the number of parameters to be estimated in our second order model, we need that $|\mathcal{C}^{train}| \geq 3$. Otherwise, the system is underdetermined. For the special case $|\mathcal{C}^{train}| = 3$ the problem reduces to a system of three equations with three variables that either has a unique deterministic solution or no solution at all. Since this latter case might cause trouble, though chances that this happens are very low and the LMS method is in fact perfectly able to determine the solution if one exists, we require that $|\mathcal{C}^{train}| > 3$.

To allow the most accurate training we looked only at cases where $|\mathcal{C}^{test}| = 1$ leaving 4 collections to train our model on (which we do need anyway, as we just argued). Consequently, we have 5 possible ways to divide $\mathcal{C}$ in a $\mathcal{C}^{train}$ and $\mathcal{C}^{test}$, by subsequentially taking each $C \in \mathcal{C}$ as test collection ($\mathcal{C}^{test} = \{C\}$) and the remaining 4 as training collections ($\mathcal{C}^{train} = \mathcal{C} \setminus \{C\}$). For a given partitioning of $\mathcal{C}$ in $\mathcal{C}^{train}$ and $\mathcal{C}^{test}$ that way, we can train, and subsequently test, our model. Figure 6.10 describes the training procedure we followed.

Figure 6.11 describes the corresponding test procedure we followed, using the

Step 1 Perform the first order training procedure (see Figure 6.6, Subsection 6.4.2) to compute $\overline{\psi}_{C\,0}$ and $\overline{\psi}_{C\,1}$ for each collection $C \in \mathcal{C}^{train}$.

Step 2 Get τ_C and n_C for each collection $C \in \mathcal{C}^{train}$.

Step 3a Use the results from **Step 1** ($\overline{\psi}_{C\,0}$) and **Step 2** (τ_C and n_C) in combination with the LMS method on equation 6.49 to estimate $\overline{\omega}_{0\,0}$, $\overline{\omega}_{0\,1}$, and $\overline{\omega}_{0\,2}$.

Step 3b Use the results from **Step 1** ($\overline{\psi}_{C\,1}$) and **Step 2** (τ_C and n_C) in combination with the LMS method on equation 6.50 to estimate $\overline{\omega}_{1\,0}$, $\overline{\omega}_{1\,1}$, and $\overline{\omega}_{1\,2}$.

Figure 6.10: Second order training procedure (for a given partitioning of $\mathcal{C}$ in $\mathcal{C}^{train}$ and $\mathcal{C}^{test}$ where $|\mathcal{C}^{test}| = 1$)

$(\overline{\omega}_{0\,0}, \overline{\omega}_{0\,1}, \overline{\omega}_{0\,2})$ and $(\overline{\omega}_{1\,0}, \overline{\omega}_{1\,1}, \overline{\omega}_{1\,2})$ vectors that were determined according to the procedure described in Figure 6.10.

Step 1 Get τ_C and n_C for the test collection $C \in \mathcal{C}^{test}$.

Step 2a Substitute $\overline{\omega}_{0\,0}$, $\overline{\omega}_{0\,1}$, $\overline{\omega}_{0\,2}$, τ_C, and n_C in Expression 6.49 to compute $\overline{\overline{\psi}}_{C\,0}$.

Step 2b Substitute $\overline{\omega}_{1\,0}$, $\overline{\omega}_{1\,1}$, $\overline{\omega}_{1\,2}$, τ_C, and n_C in Expression 6.50 to compute $\overline{\overline{\psi}}_{C\,1}$.

Step 3 Perform the first order test procedure (see Figure 6.7, Subsection 6.4.2) for test collection $C \in \mathcal{C}^{train}$. However, now use $\overline{\overline{\psi}}_{C\,0}$ and $\overline{\overline{\psi}}_{C\,1}$ in **Step 6** instead of $\overline{\psi}_{C\,0}$ and $\overline{\psi}_{C\,1}$, respectively.

Figure 6.11: Second order test procedure (for a given partitioning of $\mathcal{C}$ in $\mathcal{C}^{train}$ and $\mathcal{C}^{test}$ where $|\mathcal{C}^{test}| = 1$)

6.5.3 Experimental results

In Figure 6.12 we combined all five test cases, each represented by their respective test collection. As before, we plotted the $\overline{\overline{nmap}}_{C\,f}$ vs. $nmap_{C\,f}$. We also included the ideal line where $\overline{\overline{nmap}}_{C\,f} = nmap_{C\,f}$.

As expected the point clouds do not group as nicely along the ideal line as in the first order case (Subsection 6.4.3). However, as we can see in Figure 6.13, the relative error stays within 25% in most of the cases. We find this quite acceptable, given the fact we use only very little information in our model (f, τ_C, and n_C). Furthermore, we stress on the fact that the number of training collections is very low from a statistical point of view.

A closer review of the log files of the first and second order training and testing runs also learned us that the number of 50 queries in total, is very low as well (recall that we had to do with these 50 queries for both the training and testing).

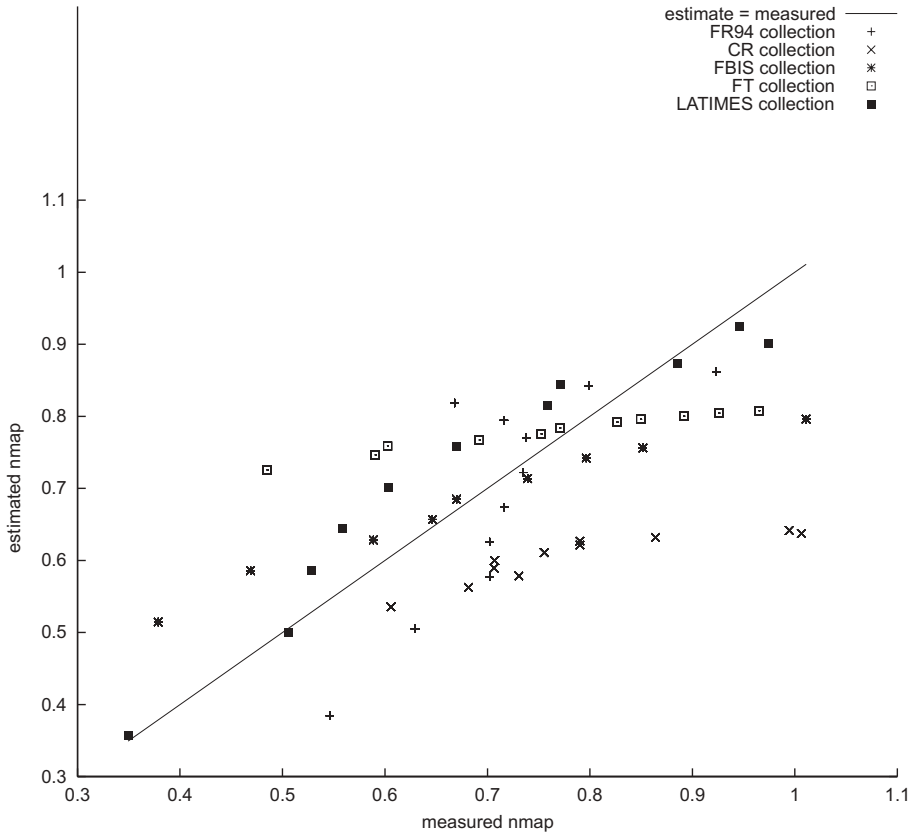

Figure 6.12: Second order model test results, $\overline{nmap}_{C\,f}$ vs. $nmap_{C\,f}$, for all collections $C \in \mathcal{C}^{test}$ and all $\mathcal{C}^{test} \subset \mathcal{C}$ where $|\mathcal{C}^{test}| = 1$

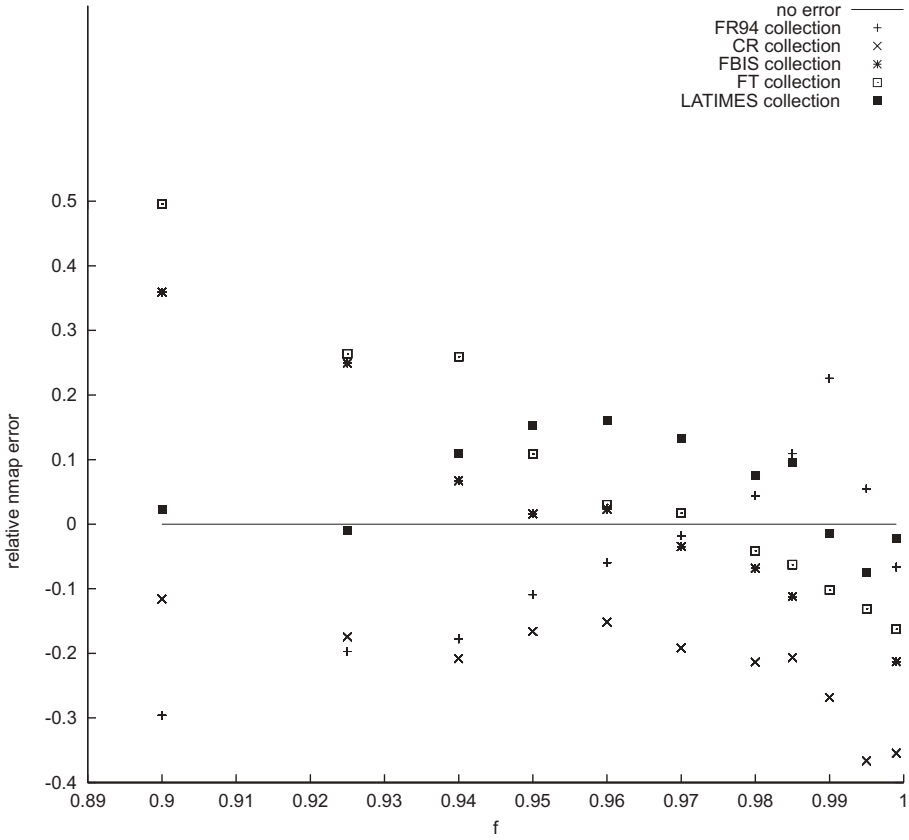

Figure 6.13: Second order model relative error, $\epsilon_{\overline{nmap}_{C\,f}}$ vs. f, for all collections $C \in \mathcal{C}^{test}$ and all $\mathcal{C}^{test} \subset \mathcal{C}$ where $|\mathcal{C}^{test}| = 1$

6.6 Concluding remarks

In this chapter, we derived a mathematical model to estimate the expected decrease in retrieval answer quality given that we use only the fraction f of the terms with the lowest document frequencies.

We distinguished two major approaches, of which the second was an extension of the first. We think both these models provide a good step in the right direction to answer the research question this chapter started with:

Q2 *Can we estimate the quality behavior as a function of the relative size of the used fragments?*

The performed experiments demonstrated that for the first approach, in which we tested our model on the same collection as we trained its parameters on, our model predicts the quality implications of decreasing f very well. The second approach attempted to overcome the major drawback of the first approach. There we trained the model on different collections than we wanted to test it on, i.e., predict the quality for. We observed a significant increase in the (relative) estimation error, which we expected beforehand. But the results are still interesting, since the relative error stayed mostly within the 25% range.

However, we still question whether these models, and in particular the second one, can be used in practice. In Chapter 7 we elaborate a bit more on possible directions to improve the accuracy and the use of the models in a real world application setting.

Furthermore, we only performed experiments for the case where we used the first fragment of two fragments in total. Note that for the quality model the number of fragments does not really matter. Only after which term we cut off processing is important. Suppose we have a first fragment of relative size f. Now suppose we split this first fragment in several smaller fragments. As long as all sub-fragments of the original first fragment containing query terms are used during query processing, the quality of the resulting answer does not change. By providing the quality model with the original fragmenting coefficient f, being the sum of the fragmenting coefficients of the sub-fragments, the quality prediction will be unchanged as well.

So, we can conclude that our quality model does meet its first and third requirement we stated in Chapter 3: it is fast and takes our fragmenting approach into account. However, on the second requirement, the capability to generalize over different collections, is not yet met. We see this as topic for further research.

Also, in the construction of our first-order model, we assumed any change in

$nEscore_{C\,fij}$, i.e., the normalized expected score of a term on a document, to effect $nmap_{C\,f}$ proportionally (Expression 6.33). Given the rather good experimental results obtained with our first-order model, we have no reason to question this assumption on its practical effectiveness. However, we certainly are interested in a more formal derivation of this relation, so it is certainly a candidate for future research. Likewise, we have no reason to withdraw the approximation of $ntf_{C\,ij}$ by a constant γ_C. But for the approximation of the formulas 6.47 and 6.48 (which in fact imply approximations of the Expressions 6.44 and 6.45) by the formulas 6.49 and 6.50 we are not so certain. The results of the second order model can be considered quite reasonable, given the fact that we wanted to use only very little information, and that it was a first attempt to predict retrieval quality using a model trained on other collections. However, the results are not that good that we are willing to accept the approximation blindly. Furthermore, we did approximate a non-linear mapping by a simple linear one. Mathematically speaking, such an approximation has a high chance of miss-fitting the original mapping quite a bit. So, we certainly think this approximation should be investigated in more detail in the future.

References

[BHC+01a] H.E. Blok, D. Hiemstra, R.S. Choenni, F.M.G. de Jong, H.M. Blanken, and P.M.G. Apers, *Predicting the cost-quality trade-off for information retrieval queries: Facilitating database design and query optimization*, Tenth International Conference on Information and Knowlegde Management (ACM CIKM'01), ACM SIGIR/SIGMIS, ACM Press, November 2001.

[BHC+01b] H.E. Blok, D. Hiemstra, R.S. Choenni, F.M.G. de Jong, H.M. Blanken, and P.M.G. Apers, *Predicting the cost-quality trade-off for information retrieval queries: Facilitating database design and query optimization*, Technical Report 01-33, Centre for Telematics and Information Technology (CTIT), University of Twente, Enschede, The Netherlands, September 2001.

[CG99] S. Chaudhuri and L. Gravano, *Evaluating Top-k Selection Queries*, Proceedings of the 25th VLDB Conference (M.P. Atkinson, M.E. Orlowska, P. Valduriez, S.B. Zdonik, and M.L. Brodie, eds.), VLDB, Morgan Kaufmann, September 1999, pp. 397–410.

[CK98] M.J. Carey and D. Kossmann, *Reducing the Braking Distance of an SQL Query Engine*, In Gupta et al. [GSW98], pp. 158–169.

[FSGM+98] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J.D. Ullman, *Computing Iceberg Queries Efficiently*, In Gupta et al. [GSW98], pp. 299–310.

[GSW98] A. Gupta, O. Shmueli, and J. Widom (eds.), *Proceedings of the 24th VLDB Conference*, VLDB, Morgan Kaufmann, 1998.

[HATB98] J. Hair, R. Anderson, R. Tatham, and W. Black, *Mutlivariate Data Analysis*, 5th ed., Prentice Hall, Inc, New Jersey, 1998, ISBN 0-13-930587-4.

[Hie00] D. Hiemstra, *A probabilistic justification for using tf · idf term weighting in information retrieval*, International Journal on Digital Libraries **3** (2000), no. 2, 131–139.

[HK98] D. Hiemstra and W. Kraaij, *Twenty-One at TREC-7: Ad-hoc and Cross-language track*, Proceedings of the Seventh Text Retrieval Conference (TREC-7) (Gaithersburg, Maryland), NIST Special publications, November 1998, pp. 227–238.

[PC98] J.M. Ponte and W.B. Croft, *A Language Modeling Approach to Information Retrieval*, SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM SIGIR, ACM, August 1998, pp. 275–281.

[SB88] G. Salton and C. Buckley, *Term-Weighting Approaches in Automatic Text Retrieval*, Information Processing & Management **24** (1988), no. 5, 513–523.

[SJ72] K. Sparck-Jones, *A Statistical Interpretation of Term Specifity and its Application in Retrieval*, Journal of Documentation **28** (1972), no. 1, 11–20.

[Zip49] G.K. Zipf, *Human Behavior and the Principle of Least Effort*, Addison-Wesley, Reading, MA, USA, 1949.

Chapter 7

Conclusions & future work

7.1 Introduction

The final chapter of this dissertation consists of two parts. In the first part, Section 7.2, we present our conclusions, both for our three derived research questions, as well as our main objective. In Section 7.3 we present some suggestions for future work.

7.2 Conclusions

First, we revisit the conclusions concerning our three derived research questions in the Section 7.2.1, 7.2.2, and 7.2.3. Next, we address our main research objective as stated in Chapter 1.

7.2.1 Fragmentation & set-based IR top-N query optimization

In Chapter 4 we investigated research question number 3:

Q3	*Can horizontal fragmentation facilitate the trading of quality for speed to support set-based IR top-N query optimization?*

We performed four series of experiments. For all the experiments we measured both the execution time as well as two quality measures: recall and average

precision. In the case where we used only the first of two fragments, we saw a clear reduction in execution costs as well as answer quality for smaller first fragments. Allowing the use of the second fragment when necessary showed a clear improvement in answer quality but also a corresponding drop in speed.

We ported both safe and unsafe element-at-a-time IR optimization tricks to a fragment-at-a-time, i.e., set-based, way of processing. Next to that we introduced a heuristic unsafe variant of the normal unsafe optimization method. We tested these optimization methods for the case where we split up the *tf* data in 25 equally sized fragments. All techniques showed a lot of overhead in administrative work resulting in no significant improvement, or sometimes even loss, of execution speed while keeping maximum retrieval quality. So, it is yet unclear whether these techniques can be of help when optimizing IR query processing in a DBMS. Furthermore, the experiments showed that the length of the query and the size of the requested top-N, i.e., N, also play an important role in the processing speed and, to some point, the result quality.

All experiments, but in particular those concerning the two fragment cases, showed clearly that ignoring more highly frequent terms, i.e., using a smaller first fragment or cutting off query processing after less fragments, reduced both execution costs and result quality. So, we can conclude that Chapter 4 has shown sufficient evidence to answer our research question of interest positively.

7.2.2 Estimating selectivity

Our research concerning the third question showed a clear trend in the relation between the used amount of data and the execution speed. We expected that it would be possible to construct a quite accurate selectivity model given the smoothness of this relation.

In Chapter 5 we indeed were able to construct a quite good model. The resulting model is a linear function of the form $l\alpha\beta$, where α and β are parameters that depend on the document frequency distribution, i.e., df, of the terms and the size of the fragment for which the selectivity is to be estimated, and l is length of the query for which the selectivity is to be estimated. The parameters α and β can be determined easily at database design time, by a single scan over the data. Also, after each update α and β can be updated incrementally. The selectivity can then be estimated at query evaluation time with almost no costs since only one scalar multiplication is required. Furthermore, the experiments showed that our model is quite accurate, as well. So, our selectivity model does adhere to the three requirements we posed in Chapter 3.

Now recall our first research question:

Q1 *Can we estimate selectivity as a function of the used fragments?*

In our research we only looked at the case for two fragments. For this case our model is able to estimate the selectivity on the first TFstats fragment. Strictly speaking, this only answers our research question partially and not for the general case of predicting the selectivity for any fragment in the case of an arbitrary number of fragments.

However, we think it is quite straightforward to extend the model to predict the selectivity for a case with more than two fragments and other than the first fragment via leaving out preceding fragments and renumbering of the terms. The actual work to check this is future work, which we describe in Section 7.3.

7.2.3 Estimating quality

The work in Chapter 4 also showed a clear trend in the relation between the used fraction of the terms and quality of the resulting answers.

In Chapter 6 we looked at the second research question:

Q2 *Can we estimate the quality behavior as a function of the relative size of the used fragments?*

to see to what point we could predict this quality behavior, like we were able to for the selectivity.

We constructed two models, a first order model for predicting quality behavior for one single collection, and a second-order model that attempted to port a first order model, once trained, to other collections. Our experiments showed that the first order model was able to predict quality behavior quite well. The second-order model was able to deliver predictions as well but with a much larger error margin. We find this error too large for practical use of this model. But given the fact it was a first attempt to build such a generalizing model the results were quite promising.

Like for the selectivity model, we again looked at the two fragment case, for which our models can predict the quality implications of shrinking the first fragment. A generalization to more than two fragments is, however, quite straightforward by using the sum of the fragmenting coefficients of all fragments before the cut off.

7.2.4 Main objective revisited: Efficient set-based IR query processing

Now that we have gathered the results for our three derived objectives we return to our main objective:

Q *Can IR top-N queries be optimized in a database manner?*

We saw that porting IR optimization techniques to the set-based database context is not trivial. We showed that it is possible to predict at query optimization time what the selectivity of the query will be for a given first fragment. We also mentioned that the same prediction very likely can be made for other fragments as well. Furthermore, we showed a first attempt to predict the quality implications of using a first fragment of a certain size, relative to the entire data set. Here too, we mentioned that a generalization to many fragments is very likely to be possible. Combining these results, we think we showed that our idea to decide on the cut off boundary before query execution, thus getting rid of the overhead in administrative work that makes the porting of IR optimization techniques less profitable, might be promising indeed.

So, we may conclude that the answer to our main research questions is most likely *YES*. However, we do suggest additional research to provide a more thorough basis for this conclusion. In the next section we discuss some of the issues for future work in that direction.

7.3 Future work

In this section we present our suggestions for future work. We have divided this in two parts. First, in Section 7.3.1, we address future work that more or less directly follows from the experiments we performed in the previous three chapters. Secondly, Section 7.3.2 addresses some architectural issues concerning future query optimizers and cost models to allow the exploitation of the results presented in this dissertation.

7.3.1 Experimental extensions

In Section 7.2 we presented our conclusions concerning our three research questions. For each of these three research questions we discovered new problems while doing our research or we found that additional experiments would be interesting. In this section we address some suggestions for future work related

to these findings. Again, we structured things according to our three research questions.

Improved exploitation of fragments

The experiments in Chapter 4 showed no significant benefit of using the fragment-at-a-time version of the safe, unsafe, and heuristic unsafe IR top-N optimization methods. The main reason was the great additional overhead in administrative work involved in updating the ranking after having processed each fragment. We hoped on better results given the performance gains reported in literature for the safe and unsafe techniques when used in a dedicated IR system. Maybe we could reduce the additional overhead when Monet would be extended with top-N aware operators, similar to the `BREAK` or `STOP AFTER` clauses proposed for SQL, as we described in Chapter 2. It would be interesting to repeat the experiments with such operators.

One of the reasons to horizontally fragment the TFstats relation could be to allow batch-wise processing in main-memory. An interesting question would be what the most optimal fragment size would be given a particular main-memory size.

Also, we are interested in the possible applications of our fragmentation approach for scaling the indexed text collection and computing in parallel, maybe even in a shared-nothing environment. This also means we are interested in distributing the data over several systems. We already have been doing some first experiments in this direction using the 10 and 100 GB WebTREC collections. These collections contain actual web documents and, like for the normal TREC collections, come with queries and accompanying relevance judgments. Also, we have been doing some first experiments with setting up web search engine facilities using a cluster of four identical PCs, similar to the one we used for several of our experiments as described in Chapter 3. But these experimental results are too premature yet, to be reported here. Also, we experienced memory problems with Monet, which caused some problems with our experimental system, making scaling experiments difficult to execute for now. Note that Monet is a topic of ongoing research itself, which typically suffers from such instability problems. We expect that the Monet development team at CWI [CWI] will cope with the problems in the near future.

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 Chapter 4 are a first step in the

right direction but we are certainly interested to see how close we can get to the execution times of a state-of-the-art dedicated IR system.

Improving insight in selectivity model

In Chapter 5 we derived a selectivity model which actually does not require the data distribution to be Zipfian as we have with the text retrieval case for which we evaluated it. So, we are interested whether our model indeed can be used for other distributions as well.

However, we did assume that the query is distributed similarly to the data in the database itself. We are also interested to find out whether we can adapt our model for the case where we drop this requirement. Currently, experimental work is being done that eventually will provide us with some answers concerning these two issues.

Furthermore, a mathematical analysis of the error of our selectivity model is needed to allow better comparison with other methods. We have already been working on such an analysis but still have to take care of some mathematical issues to complete it.

Improving insight in quality model

In the construction of our two models in Chapter 6 we made several assumptions. The main reason for these assumptions was to keep the models simple. For a first model this is an acceptable approach. However, the still too poor generalization capabilities of the second model, in particular, suggest we should construct more formally derived models and more experiments to validate our current and such new models. Also, we used the least mean square regression technique. This may be a too crude approximation since in our model derivation we actually are not dealing with linear functions.

Finally, we want to stress that the statistical stability of our experiments in Chapter 6 seemed to have suffered somewhat from the lack of data. This holds in particular for the experiments we performed for the second order model. We recommend to repeat these experiments with more and different document collections and more queries. Due to several practical reasons, we are not able to report any results on that here. However, we are working on evaluation of our models on the WebTREC datasets (10 and, 100 GB). Furthermore, the current queries can be extended with the topics of other TREC conferences, whereas we only used the topics of TREC-6 in this dissertation.

7.3.2 Query optimizer architecture

Next to the suggestions for future work that are an almost direct extension of our experimental efforts, we also are interested in how the results of this dissertation can be used in tomorrows query optimizers and cost models. This section describes three suggestions for future research in this direction.

A cost model that accepts selectivity hints

The selectivity model we presented in Chapter 5 of this dissertation is intended to be used in a cost model eventually. However, this selectivity model, once its parameters have been determined, is only applicable for the data those parameters reflect. So, a cost model that has to use this model has to be told somehow when it is allowed to use this selectivity model and when not. Though our selectivity model might be applicable to many different domains, our first interest was the development of a good model for our particular case, and we can imagine that it might be interesting to develop special-purpose selectivity models for other special domains, as well.

One could imagine that such special-purpose selectivity models are incorporated in the extension module that houses the accompanying special-purpose data management facilities in the DBMS for that particular application domain. So, for instance, our selectivity model could be incorporated in the `DOCREP` extension of Moa. To allow the cost model to deal with any arbitrary selectivity model provided by some extension, we propose to setup a generic interface that allows the extension to give selectivity 'hints' to the cost model regarding operators provided by that extension.

In the Moa context this would mean we require a cost model for Monet which can be told by a Moa extension to use a particular selectivity model instead of some standard model. Like the E-ADT principle to enhance a query optimizer with new capabilities concerning some extension, this could be seen as a selectivity enhancement to provide the general selectivity estimation facilities in the cost model with application specific information.

A generalized cost-quality trade-off optimizer approach

In this dissertation we presented both a selectivity model, being a crucial part of any cost model, and a quality model. Both models were developed in the IR context with the underlying idea in mind to allow trading quality for speed. The IR field is not the only area where this trade-off might be interesting.

For instance, in the border area of GIS and telecommunications, applications exist where a moving user asks location dependent questions about route infor-

mation via a GSM phone, for instance route information to a nice place to stop for lunch. Then a better answer can take longer, but at the same time part of the answer might become outdated. For instance, the user passed a certain exit of the highway relevant to the answer in the meantime. Transmission of these answers are a waste of dialing minutes. However, a quicker answer, that allows inclusion of that nearby exit, might contain more answers for which the route info is still valid, but it will also contain more answers that do not match the user interest well enough. Since these answers are useless these also are a waste of dialing minutes.

We think more areas exist like these, for which a certain trade-off between speed and quality is interesting.

It would be interesting to investigate if this trading can be observed in a more abstract sense. And if so, it would be interesting to see how modern day database technology, and in particular the query optimizer, could be adapted to handle such a trade-off, maybe in cooperation with the user, to the satisfaction of the user.

An intra-/inter-object optimizer in Moa

In Chapter 3 we argued that the E-ADT, or intra-object, optimization approach does not suffice if we want to perform 'top-N push down' query optimization. We introduced the notion of inter-object optimization to capture the concept of performing query optimization across the typical orthogonality boundaries of extension modules, required to handle such optimization problems.

We think that Moa provides some additional benefits to implement inter-object optimization, due to its particular way of handling of nested structures at the physical level, combined with its uniform physical interface, i.e., each extension is implemented in terms of a common physical language.

Currently, Moa has no query optimizer. We are interested in implementing a new type of query optimizer in Moa to fill this gap, providing us a nice opportunity to realize inter-object optimization facilities, next to the commonly used optimizer techniques known in literature, such as push-select-down, but also intra-object optimization techniques. This would neatly fit in the overall Moa vision of being a flexible platform for both standard as well as non-standard database applications.

References

[CWI] *National Research Institute for Mathematics and Computer Science (CWI)*, URL: http://www.cwi.nl/.

Appendix A

Derivations and proofs

In this appendix we present some derivations and proofs related to expressions used in Chapter 5.

A.1 Proof of Expression 5.34

This appendix concerns the proof of Expression 5.34 in Chapter 5 on page 113:

$$\frac{\partial}{\partial p_{i'}} h_k(\vec{p}) \quad = \quad \sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \setminus i' \\ |Z|=k-1}} \prod_{i \in Z} p_i.$$

Recall Expression 5.27 in Chapter 5 on page 112:

$$h_k : \vec{p} \quad \mapsto \quad \sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \\ |Z|=k}} \prod_{i \in Z} p_i.$$

We can divide $h_k(\vec{p})$ in two parts, of which one contains all terms *without* $p_{i'}$:

$$h_{k,0}(\vec{p}) = \sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \\ |Z|=k \\ i' \notin Z}} \prod_{i \in Z} p_i,$$

and the other contains all terms *with* $p_{i'}$:

$$h_{k,1}(\vec{p}) = \sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \\ |Z|=k \\ i' \in Z}} \prod_{i \in Z} p_i,$$

such that $h_k(\vec{p}) = h_{k,0}(\vec{p}) + h_{k,1}(\vec{p})$.
Since $h_{k,0}$ does not contain any terms with $p_{i'}$ we know that $\frac{\partial}{\partial p_{i'}} h_{k,0}(\vec{p}) = 0$, which means:

$$\frac{\partial}{\partial p_{i'}} h_k(\vec{p}) = \frac{\partial}{\partial p_{i'}} h_{k,1}(\vec{p}).$$

With some simple formula manipulations we then get:

$$\begin{aligned}
&\frac{\partial}{\partial p_{i'}} h_k(\vec{p}) \\
&= \frac{\partial}{\partial p_{i'}} \sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \\ |Z|=k \\ i' \in Z}} \prod_{i \in Z} p_i \\
&= \frac{\partial}{\partial p_{i'}} \sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \setminus i' \\ |Z|=k-1}} p_{i'} \prod_{i \in Z} p_i \\
&= \frac{\partial}{\partial p_{i'}} p_{i'} \sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \setminus i' \\ |Z|=k-1}} \prod_{i \in Z} p_i \\
&= \left(\frac{\partial}{\partial p_{i'}} p_{i'} \right) \sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \setminus i' \\ |Z|=k-1}} \prod_{i \in Z} p_i \\
&= 1 \cdot \sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \setminus i' \\ |Z|=k-1}} \prod_{i \in Z} p_i
\end{aligned}$$

$$= \sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \setminus i' \\ |Z| = k-1}} \prod_{i \in Z} p_i$$

which completes our proof. □

A.2 Proof of Expression 5.36

In this appendix we present the proof of Expression 5.36 in Chapter 5 on page 113:

$$\frac{\partial}{\partial p_{i'}} g_k(\vec{p}) = k g_{k-1}(\vec{p})$$

Recall Expression 5.28 in Chapter 5 on page 112:

$$g_k : \vec{p} \mapsto \left(\sum_{i=1}^{m} p_i \right)^k .$$

This means that:

$$g_1(\vec{p}) = \sum_{i=1}^{m} p_i .$$

So we can write g_k as follows:

$$g_k(\vec{p}) = \left(g_1(\vec{p}) \right)^k .$$

This means that, using the chain rule for differentiation,

$$\begin{aligned}
&\frac{\partial}{\partial p_{i'}} g_k(\vec{p}) \\
&= \frac{\partial}{\partial p_{i'}} \left(g_1(\vec{p}) \right)^k \\
&= k \cdot \frac{\partial}{\partial p_{i'}} g_1(\vec{p}) \cdot \left(g_1(\vec{p}) \right)^{k-1} \\
&= k \cdot \left(\sum_{i=1}^{m} \frac{\partial}{\partial p_{i'}} p_i \right) \cdot \left(g_1(\vec{p}) \right)^{k-1} .
\end{aligned}$$

Note that $\frac{\partial}{\partial p_{i'}} p_i = 1$ for $i' = i$ and $\frac{\partial}{\partial p_{i'}} p_i = 0$ for all $i' \neq i$, so

$$\sum_{i=1}^{m} \frac{\partial}{\partial p_{i'}} p_i = 1$$

which means that

$$\frac{\partial}{\partial p_{i'}} g_k(\vec{p}) = k \left(g_1(\vec{p})\right)^{k-1} = k g_{k-1}(\vec{p})$$

which completes our proof. □

A.3 Equality of Expressions 5.38 and 5.39

In this appendix we explain the equality of Expression 5.38

$$E(X) = \sum_{k=1}^{l} \sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \\ |Z|=k}} \left(\prod_{i \in Z} p_i\right) \left(\sum_{i \in Z} df_i\right) P(K = k).$$

and Expression 5.39

$$E(X) = \sum_{k=1}^{l} \sum_{i'=1}^{m} p_{i'} \, df_{i'} \sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \setminus i' \\ |Z|=k-1}} \prod_{i \in Z} p_i P(K = k)$$

in Chapter 5.

This means we have to show that

$$\sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \\ |Z|=k}} \left(\prod_{i \in Z} p_i\right) \left(\sum_{i \in Z} df_i\right) = \sum_{i'=1}^{m} p_{i'} \, df_{i'} \sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \setminus i' \\ |Z|=k-1}} \prod_{i \in Z} p_i \tag{A.1}$$

for $k > 1$, and

$$\sum_{\substack{Z \subset \{1,2,3,\ldots,m\} \\ |Z|=k}} \left(\prod_{i \in Z} p_i \right) \left(\sum_{i \in Z} df_i \right) = \sum_{i'=1}^{m} p_{i'} \, df_{i'} \tag{A.2}$$

for $k = 1$.

Since the case for $k = 1$ is trivial, we concentrate on the case $k > 1$.

Consider the following example for $m = 5$ and $k = 2$. This example illustrates the important properties that we need to explain for an arbitrary m and k.

$$\begin{aligned}
&\sum_{\substack{Z \subset \{1,2,3,4,5\} \\ |Z|=2}} \left(\prod_{i \in Z} p_i \right) \left(\sum_{i \in Z} df_i \right) \\
&= \left(\prod_{i \in \{1,2\}} p_i \right) \left(\sum_{i \in \{1,2\}} df_i \right) + \left(\prod_{i \in \{1,3\}} p_i \right) \left(\sum_{i \in \{1,3\}} df_i \right) + \\
&\quad \left(\prod_{i \in \{1,4\}} p_i \right) \left(\sum_{i \in \{1,4\}} df_i \right) + \left(\prod_{i \in \{1,5\}} p_i \right) \left(\sum_{i \in \{1,5\}} df_i \right) + \\
&\quad \left(\prod_{i \in \{2,3\}} p_i \right) \left(\sum_{i \in \{2,3\}} df_i \right) + \left(\prod_{i \in \{2,4\}} p_i \right) \left(\sum_{i \in \{2,4\}} df_i \right) + \\
&\quad \left(\prod_{i \in \{2,5\}} p_i \right) \left(\sum_{i \in \{2,5\}} df_i \right) + \left(\prod_{i \in \{3,4\}} p_i \right) \left(\sum_{i \in \{3,4\}} df_i \right) + \\
&\quad \left(\prod_{i \in \{3,5\}} p_i \right) \left(\sum_{i \in \{3,5\}} df_i \right) + \left(\prod_{i \in \{4,5\}} p_i \right) \left(\sum_{i \in \{4,5\}} df_i \right) \\
&= p_1 p_2 \, df_1 + p_1 p_2 \, df_2 + p_1 p_3 \, df_1 + p_1 p_3 \, df_3 + \\
&\quad p_1 p_4 \, df_1 + p_1 p_4 \, df_4 + p_1 p_5 \, df_1 + p_1 p_5 \, df_5 + \\
&\quad p_2 p_3 \, df_2 + p_2 p_3 \, df_3 + p_2 p_4 \, df_2 + p_2 p_4 \, df_4 + \\
&\quad p_2 p_5 \, df_2 + p_2 p_5 \, df_5 + p_3 p_4 \, df_3 + p_3 p_4 \, df_4 + \\
&\quad p_3 p_5 \, df_3 + p_3 p_5 \, df_5 + p_4 p_5 \, df_4 + p_4 p_5 \, df_5
\end{aligned}$$

$$
\begin{aligned}
= \quad & p_1p_2\,df_1 + p_1p_3\,df_1 + p_1p_4\,df_1 + p_1p_5\,df_1 + \\
& p_1p_2\,df_2 + p_2p_3\,df_2 + p_2p_4\,df_2 + p_2p_5\,df_2 + \\
& p_1p_3\,df_3 + p_2p_3\,df_3 + p_3p_4\,df_3 + p_3p_5\,df_3 + \\
& p_1p_4\,df_4 + p_2p_4\,df_4 + p_3p_4\,df_4 + p_4p_5\,df_4 + \\
& p_1p_5\,df_5 + p_2p_5\,df_5 + p_3p_5\,df_5 + p_4p_5\,df_5
\end{aligned}
$$

$$
\begin{aligned}
= \quad & p_1df_1(p_2 + p_3 + p_4 + p_5) + \\
& p_2df_2(p_1 + p_3 + p_4 + p_5) + \\
& p_3df_3(p_1 + p_2 + p_4 + p_5) + \\
& p_4df_4(p_1 + p_2 + p_3 + p_5) + \\
& p_5df_5(p_1 + p_2 + p_3 + p_4)
\end{aligned}
$$

It is obvious that we have performed a simple regrouping of the terms in the sum, followed by a reordering on terms with a $p_{i'}\,df_{i'}$ factor and collection of that common $p_{i'}\,df_{i'}$ factor.

The same regrouping principle can be applied to the general case. In the left hand side of Expression A.1 we have $\binom{m}{k}$ groups of k terms. Each of these terms is a $df_{i'}$ value multiplied with a k-product of p_i values. Since each distinct $df_{i'}$ value always comes with a corresponding $p_{i'}$ value, we can regroup these terms by a common $p_{i'}\,df_{i'}$ factor. Since we have m unique $p_{i'}\,df_{i'}$ factors, we have

$$
\begin{aligned}
\binom{m}{k}\frac{k}{m} &= \frac{m!}{k!(m-k)!}\frac{k}{m} = \frac{(m-1)!}{(k-1)!(m-k)!} \\
&= \frac{(m-1)!}{(k-1)!((m-1)-(k-1))!} = \binom{m-1}{k-1}
\end{aligned}
$$

product terms per $p_{i'}\,df_{i'}$ factor. These $\binom{m-1}{k-1}$ product terms are the $(k-1)$-products of remaining, so $i \neq i'$, p_i values. The right hand side of Expression A.1 follows directly from this observation.

References

[ACM] *ACM Digital Library*, URL: http://www.acm.org/dl/.

[ACM86] *Proceedings of the 1986 ACM SIGMOD International Conference on the Management of Data*, ACM Press, June 1986.

[ACM96] *Proceedings of the 1996 ACM SIGMOD International Conference on the Management of Data*, ACM Press, 1996.

[ACM97] *Proceedings of the 1997 ACM SIGMOD International Conference on the Management of Data*, ACM Press, 1997.

[ACM98a] *Proceedings of the 1998 ACM SIGMOD International Conference on the Management of Data*, ACM Press, 1998.

[ACM98b] *Proceedings of the 1998 ACM SIGMOD International Conference on Principles of Database Systems (PODS'98)*, ACM Press, 1998.

[AOV+99] M.P. Atkinson, M.E. Orlowska, P. Valduriez, S.B. Zdonik, and M.L. Brodie (eds.), *Proceedings of the 25th VLDB Conference*, VLDB, Morgan Kaufmann, September 1999.

[BA94] K. Böhm and K. Aberer, *Storing HyTime Documents in an Object-Oriented Database*, In Nicholas and Mayfield [NM94], pp. 26–33.

[BCBA01] H.E. Blok, R.S. Choenni, H.M. Blanken, and P.M.G. Apers, *A selectivity model for fragmented relations in information retrieval*, Technical Report 01-02, Centre for Telematics and Information Technology (CTIT), University of Twente, Enschede, The Netherlands, January 2001.

[BCC94] J. Broglio, J.P. Callan, and W.B. Croft, *INQUERY System Overview*, TIPSTER Text Program (Phase I) (San Francisco, CA), Morgan Kaufmann, 1994, pp. 47–67.

[BH96] C.A. van den Berg and van der A. Hoeven, *Monet meets OO7*, Proceedings of OODS'96, January 1996.

[BHC+01a] H.E. Blok, D. Hiemstra, R.S. Choenni, F.M.G. de Jong, H.M. Blanken, and P.M.G. Apers, *Predicting the cost-quality trade-off for information retrieval queries: Facilitating database design and query optimization*, Tenth International Conference on Information and Knowlegde Management (ACM CIKM'01), ACM SIGIR/SIGMIS, ACM Press, November 2001.

[BHC+01b] H.E. Blok, D. Hiemstra, R.S. Choenni, F.M.G. de Jong, H.M. Blanken, and P.M.G. Apers, *Predicting the cost-quality trade-off for information retrieval queries: Facilitating database design and query optimization*, Technical Report 01-33, Centre for Telematics and Information Technology (CTIT), University of Twente, Enschede, The Netherlands, September 2001.

[BK95] P.A. Boncz and M.L. Kersten, *Monet: An Impressionist Sketch of an Advanced Database System*, Basque International Workshop on Information Technology, Data Management Systems (BIWIT'95), IEEE Computer Society Press, July 1995.

[BK99] P.A. Boncz and M.L. Kersten, *MIL Primitives For Querying A Fragmented World*, VLDB Journal **8** (1999), no. 2.

[BL85] C. Buckley and A.F. Lewit, *Optimisation of Inverted Vector Searches*, SIGIR '85: Proceedings of the 8th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM SIGIR, ACM, August 1985, pp. 97–110.

[BMK99] P.A. Boncz, S. Manegold, and M.L. Kersten, *Database Architecture Optimized for the new Bottleneck: Memory Access*, In Atkinson et al. [AOV+99].

[BMN94] K. Böhm, A. Müller, and E. Neuhold, *Structured Document Handling - a Case for Integrating Databases and Information Retrieval*, In Nicholas and Mayfield [NM94], pp. 147–154.

[BP98] S. Brin and L. Page, *The Anatomy of a Large-Scale Hypertextual Web Search Engine*, Proceedings of the Seventh International World Wide Web Conference (WWW7), WWW7 Consortium, April 1998.

[BRG88] E. Bertino, F. Rabitti, and S. Gibbs, *Query Processing in a Multimedia Document System*, ACM Transactions on Office Information Systems **6** (1988), no. 1, 1–41.

[Bro95] E.W. Brown, *Execution Performance Issues in Full-Text Information Retrieval*, Ph.D.T./Technical Report 95-81, University of Massachusetts, Amherst, October 1995.

[Bur92] F.J. Burkowski, *Retrieval Activities in a Database Consisting of Heterogeneous Collections of Structured Text*, SIGIR '92: Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM SIGIR, ACM, June 1992, pp. 112–125.

[BVB99] H.E. Blok, A.P. de Vries, and H.M. Blanken, *Top N MM query optimization: The best of both IR and DB worlds*, Conferentie Informatiewetenschap 1999 [Eng.: Information Science Conference 1999] (CIW'99) (Amsterdam) (P. De Bra and L. Hardman, eds.), Werkgemeenschap Informatiewetenschap, November 1999.

[BVBA01] H.E. Blok, A.P. de Vries, H.M. Blanken, and P.M.G. Apers, *Experiences with IR TOP N Optimization in a Main Memory DBMS: Applying 'the Database Approach' in New Domains*, Advances in Databases, 18th British National Conference on Databases (BNCOD 18) (Chilton, UK) (B. Read, ed.), Lecture Notes in Computer Science, vol. 2097, CLRC Rutherford Appleton Laboratory, Springer, July 2001.

[BWK98] P.A. Boncz, A.N. Wilschut, and M.L. Kersten, *Flattening an Object Algebra to Provide Performance*, Proceedings of the 14th International Conference on Data Engineering (ICDE'98), IEEE Transactions on Knowledge and Data Engineering, IEEE Computer Society, February 1998.

[BWZ+01] H.E. Blok, M. Windhouwer, R. van Zwol, M. Petkovic, P.M.G. Apers, M.L. Kersten, and W. Jonker, *Flexible and scalable digital library search*, Proceedings of the 27th VLDB Conference, VLDB, September 2001.

[Car75] A.F. Cardenas, *Analysis and performance of inverted data base structures*, Communications of the ACM **18** (1975), no. 5, 253–263.

[CCH92] J.P. Callan, W.B. Croft, and S.M. Harding, *The INQUERY Retrieval System*, 3rd International Conference on Database and Expert Systems Applications (DEXA'92) (A.M. Tjoa and I. Ramos, eds.), 1992, pp. 78–83.

[CG96a] S. Chaudhuri and L. Gravano, *Optimizing Queries over Multimedia Repositories*, IEEE Data Engineering Bulletin **19** (1996), no. 4, 45–52, Also see [CG96b].

[CG96b] S. Chaudhuri and L. Gravano, *Optimizing Queries over Multimedia Repositories*, In *Proceedings of the 1996 ACM SIGMOD International Conference on the Management of Data* [ACM96], pp. 91–102.

[CG99] S. Chaudhuri and L. Gravano, *Evaluating Top-k Selection Queries*, In Atkinson et al. [AOV+99], pp. 397–410.

[Cha98] S. Chaudhuri, *An Overview of Query Optimization in Relational Systems*, In *Proceedings of the 1998 ACM SIGMOD International Conference on Principles of Database Systems (PODS'98)* [ACM98b], pp. 34–42.

[CHS99] F. Chu, J.Y. Halpern, and P. Seshadri, *Least Expected Cost Query Optimization: An Exercise in Utility*, Proceedings of the 1999 ACM SIGMOD International Conference on Principles of Database Systems (PODS'99), ACM Press, 1999, pp. 138–147.

[ÇJF+98] U. Çetintemel, B.Th. Jónsson, M.J. Franklin, C.L. Giles, and D. Srivastava, *Evaluating Answer Quality/Efficiency Tradeoffs*, Proceedings of the 5th International Workshop on Knowledge Represenation Meets Databases (KRDB '98): Innovative Application Programming and Query Interfaces (A. Borgida, V.K. Chaudhri, and M. Staudt, eds.), CEUR Workshop Proceedings, vol. 10, May 1998, pp. 19.1–19.4.

[CK97a] M.J. Carey and D. Kossmann, *On Saying "Enough Already!" in SQL*, In *Proceedings of the 1997 ACM SIGMOD International Conference on the Management of Data* [ACM97], pp. 219–230.

[CK97b] M.J. Carey and D. Kossmann, *Processing Top and Bottom N Queries*, IEEE Bulletin of the Technical Commitee on Data Engineering **20** (1997), no. 3, 12–19.

[CK98] M.J. Carey and D. Kossmann, *Reducing the Braking Distance of an SQL Query Engine*, In Gupta et al. [GSW98], pp. 158–169.

[CMN98] S. Chaudhuri, R. Motwani, and V. Narasayya, *Random Sampling for Histogram Construction: How much is enough?*, In *Proceedings of the 1998 ACM SIGMOD International Conference on the Management of Data* [ACM98a], pp. 436–447.

[CMN99] S. Chaudhuri, R. Motwani, and V. Narasayya, *On Random Sampling over Joins*, Proceedings of the 1999 ACM SIGMOD International Conference on the Management of Data, ACM Press, 1999, pp. 263–274.

[CP86] S. Ceri and G. Pelagatti, *Distributed Databases: Principles and Systems*, McGraw-Hill computer science series, International Student Editions, McGraw-Hill, Inc., 1986.

[CP97] D.R. Cutting and J.O. Pedersen, *Space Optimizations for Total Ranking*, Proceedings of RAIO'97, Computer-Assisted Information Searching on Internet, June 1997, pp. 401–412.

[CR94] C.M. Chen and N. Roussopoulos, *Adaptive Selectivity Estimation Using Query Feedback*, Proceedings of the 1994 ACM SIGMOD International Conference on the Management of Data, ACM Press, May 1994, pp. 161–172.

[CWI] *National Research Institute for Mathematics and Computer Science (CWI)*, URL: http://www.cwi.nl/.

[Dat95] C.J. Date, *An Introduction to Database Systems*, 6th. ed., 1995, ISBN 0-201-54329-X.

[DKA+86] P. Dadam, K. Kuespert, F. Andersen, H.M. Blanken, R. Erbe, J. Guenauer, V. Lum, P. Pistor, and G. Walch, *A DBMS Prototype to Support NF^2 Relations: An Integrated View on Flat Tables and Hierarchies*, In *Proceedings of the 1986 ACM SIGMOD International Conference on the Management of Data* [ACM86], pp. 356–367.

[DR99a] D. Donjerkovic and R. Ramakrishnan, *Probabilistic Optimization of Top N Queries*, In Atkinson et al. [AOV+99], pp. 411–422.

[DR99b] D. Donjerkovic and R. Ramakrishnan, *Probabilistic Optimization of Top N Queries*, Technical Report CR-TR-99-1395, Department of Computer Sciences, University of Wisconsin-Madison, 1999.

[Exc99] *Excalibur Text Search Datablade*, Informix Tech Brief, 1999, See also [Inf].

[Fag98] R. Fagin, *Fuzzy Queries in Multimedia Database Systems*, In *Proceedings of the 1998 ACM SIGMOD International Conference on Principles of Database Systems (PODS'98)* [ACM98b], pp. 1–10.

[Fag99] R. Fagin, *Combining fuzzy information from multiple systems*, Journal on Computer and System Sciences **58** (1999), no. 1, 83–99, Special issue for selected papers from the 1996 ACM SIGMOD PODS Conference.

[FGCF00] O. Frieder, D.A. Grossman, A. Chowdhury, and G. Frieder, *Efficiency Considerations for Scalable Information Retrieval Servers*, Journal of Digital Information **1** (2000), no. 5.

[FM00] R. Fagin and Y.S. Maarek, *Allowing users to weight search terms*, Proceedings of RAIO'00, Computer-Assisted Information Searching on Internet, 2000, pp. 682–700.

[FR97] N. Fuhr and T. Rölleke, *A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems*, ACM Transactions on Information Systems **15** (1997), no. 1, 32–66.

[FSGM+98] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J.D. Ullman, *Computing Iceberg Queries Efficiently*, In Gupta et al. [GSW98], pp. 299–310.

[Fuh96] N. Fuhr, *Models for Integrated Information Retrieval and Database Systems*, IEEE Data Engineering Bulletin **19** (1996), no. 1, 3–13.

[Gan98] S. Ganguly, *Design and Analysis of Parametric Query Optimization Algorithms*, In Gupta et al. [GSW98], pp. 228–238.

[GBS99] T. Grabs, K. Böhm, and H.-J. Schek, *A Document Engine on a DB Cluster*, High Performance Transaction Processing Systems Workshop (Asilomar, California, USA), September 1999.

[GF98] D.A. Grossman and O. Frieder, *Information retrieval: algorithms and heuristics*, The Kluwer international series in engineering and computer science, Kluwer Academic, Boston, 1998, ISBN 0-7923-8271-4.

[GFHR97] D.A. Grossman, O. Frieder, D.O. Holmes, and D.C. Roberts, *Integrating Structured Data and Text: A Relational Approach*, Journal of the American Society of Information Science **48** (1997), no. 2, 122–132.

[GGMS96] S. Ganguly, P.B. Gibbons, Y. Matias, and A. Silberschatz, *Bifocal Sampling for Skew-Resistant Join Size Estimation*, In *Proceedings of the 1996 ACM SIGMOD International Conference on the Management of Data* [ACM96], pp. 271–281.

[GHF94] D.A. Grossman, D.O. Holmes, and O. Frieder, *A Parallel DBMS Approach to IR in TREC-3*, In *Proceedings of the Third Text Retrieval Conference (TREC-3)* [TRE94], pp. 279–288.

[Gra93] G. Graefe, *Query Evaluation Techniques for Large Databases*, ACM Computing Surveys **25** (1993), no. 2, 73–170.

[GSW98] A. Gupta, O. Shmueli, and J. Widom (eds.), *Proceedings of the 24th VLDB Conference*, VLDB, Morgan Kaufmann, 1998.

[Har94] D.K. Harman, *Evaluation Techniques and Measures*, In *Proceedings of the Third Text Retrieval Conference (TREC-3)* [TRE94], pp. A5–A13.

[HATB98] J. Hair, R. Anderson, R. Tatham, and W. Black, *Mutlivariate Data Analysis*, 5th ed., Prentice Hall, Inc, New Jersey, 1998, ISBN 0-13-930587-4.

[HCL+90] L.M. Haas, W. Chang, G.M. Lohman, J. McPherson, P.F. Wilms, G. Lapis, B. Lindsay, H. Pirahesh, M.J. Carey, and E. Shekita, *Starburst Mid-Flight: As the Dust Clears*, IEEE Transactions on Knowledge and Data Engineering **2** (1990), no. 1, 143–160.

[Hel98] J.M. Hellerstein, *Optimization Techniques for Queries with Expensive Methods*, ACM Transactions on Database Systems **23** (1998), no. 2, 113–157.

[Hie98] D. Hiemstra, *A Linguistically Motivated Probabilistic Model of Information Retrieval*, Proceeding of the second European Conference on Research and Advanced Technology for Digital Libraries (ECDL'98) (C. Nicolaou and C. Stephanidis, eds.), Springer-Verlag, 1998, pp. 569–584.

[Hie00] D. Hiemstra, *A probabilistic justification for using tf · idf term weighting in information retrieval*, International Journal on Digital Libraries **3** (2000), no. 2, 131–139.

[Hie01] D. Hiemstra, *Using language models for information retrieval*, Ph.D. thesis, University of Twente, Enschede, The Netherlands, January 2001.

[HK98] D. Hiemstra and W. Kraaij, *Twenty-One at TREC-7: Ad-hoc and Cross-language track*, Proceedings of the Seventh Text Retrieval Conference (TREC-7) (Gaithersburg, Maryland), NIST Special publications, November 1998, pp. 227–238.

[HS93] J.M. Hellerstein and M. Stonebreaker, *Predicate Migration: Optimizing Queries with Expensive Predicates*, Proceedings of the 1993 ACM SIGMOD International Conference on the Management of Data, ACM Press, May 1993, pp. 267–276.

[IB86] A. IJbema and H.M. Blanken, *Estimating bucket accesses: A practical approach*, Proceedings of the Second International Conference on Data Engineering (ICDE'86), IEEE Computer Society, IEEE, February 1986, pp. 30–37.

[IC91] Y.E. Ioannidis and S. Christodoulakis, *On the Propagation of Errors in the Size of Join Results*, Proceedings of the 1991 ACM SIGMOD International Conference on the Management of Data (J. Clifford and R. King, eds.), ACM Press, June 1991, pp. 268–277.

[Inf] *Informix Corporation*, URL: http://www.informix.com/.

[Ioa96] Y.E. Ioannidis, *Query Optimization*, The Computer Science and Engineering Handbook, CRC Press, Boca Raton, Florida, 1996, ISBN 0-8493-2909-4, ch. 45, pp. 1038–1057.

[IP95] Y.E. Ioannidis and V. Poosala, *Balancing Histogram Optimality and Practicality for Query Result Size Estimation*, Proceedings of the 1995 ACM SIGMOD International Conference on the Management of Data, ACM Press, 1995, pp. 233–244.

[JFS98] B.Th. Jónsson, M.J. Franklin, and D. Srivastava, *Interaction of Query evaluation and Buffer management for IR*, In *Proceedings of the 1998 ACM SIGMOD International Conference on the Management of Data* [ACM98a].

[JK84] M. Jarke and J. Koch, *Query Optimization in Database Systems*, ACM Computing Surveys **16** (1984), no. 2, 111–152.

[JN95] K. Järvelin and T. Niemi, *An NF^2 Relational Interface for Document Retrieval, Restructuring and Aggregation*, SIGIR '95: Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM SIGIR, ACM, June 1995, pp. 102–110.

[KN98] M.L. Kersten and N.J. Nes, *Fitness Joins in the Ballroom*, Cwi report, CWI, Amsterdam, The Netherlands, July 1998.

[LMS94] A.Y. Levy, I.S. Mumick, and Y. Sagiv, *Query Optimization by Predicate Move-Around*, Proceedings of the 20th VLDB Conference, VLDB, 1994.

[LNS90] R.J. Lipton, J.F. Naughton, and D.A. Schneider, *Practical Selectivity Estimation through Adaptive Sampling*, Proceedings of the 1990 ACM SIGMOD International Conference on the Management of Data (H. Garcia-Molina and H.V. Jagadish, eds.), ACM Press, June 1990, pp. 1–11.

[Lyn88] C. Lynch, *Selectivity estimation and query optimization in large databases with highly skewed distributions of column values*, Proceedings of 14th VLDB Conference, August 1988, pp. 240–251.

[Mut85] B. Muthuswamy, *A Detailed Statistical Model for Relational Query Optimization*, Proceedings of the 1985 ACM SIGMOD International Conference on the Management of Data, ACM Press, June 1985, pp. 439–448.

[NBY97] G. Navarro and R. Baeza-Yates, *Proximal Nodes: A Model to Query Document Databases by Content and Structure*, ACM Transactions on Information Systems **15** (1997), no. 4, 400–435.

[NIS] *National Institute of Standards and Technology (NIST)*, URL: http://www.nist.gov/.

[NM94] C.K. Nicholas and J. Mayfield (eds.), *Third International Conference on Information and Knowlegde Management (CIKM'94)*, ACM SIGIR/SIGMIS, ACM Press, November 1994.

[PC98] J.M. Ponte and W.B. Croft, *A Language Modeling Approach to Information Retrieval*, SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM SIGIR, ACM, August 1998, pp. 275–281.

[Per84] M. Persin, *Document filtering for fast ranking*, SIGIR '84: Proceedings of the 7th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM SIGIR, ACM, August 1984, pp. 339–349.

[PIHS96] V. Poosala, Y.E. Ioannidis, P.J. Haas, and E.J. Shekita, *Improved Histograms for Selectivity Estimation of Range Predicates*, In *Proceedings of the 1996 ACM SIGMOD International Conference on the Management of Data* [ACM96], pp. 294–305.

[PSS+87] H.-B. Paul, H.-J. Schek, M.H. Scholl, G. Weikum, and U. Deppisch, *Architecture and Implementation of the Darmstadt Database Kernel System*, Proceedings of the 1987 ACM SIGMOD International Conference on the Management of Data, ACM Press, June 1987, pp. 196–207.

[RDR+98] R. Ramakrishnan, D. Donjerkovic, A. Ranganathan, K.S. Beyer, and M. Krishnaprasad, *SRQL: Sorted Relational Query Language*, Proceedings of SSDBMS'98 (Piscataway, NJ), IEEE, IEEE, July 1998.

[RH96] E. Riloff and L. Hollaar, *Text Databases and Information Retrieval*, The Computer Science and Engineering Handbook, CRC Press, Boca Raton, Florida, 1996, ISBN 0-8493-2909-4, ch. 50, pp. 1125–1141.

[Rij79] C.J. van Rijsbergen, *Information Retrieval*, 2nd. ed., Butterworths, London, 1979.

[Roc71] J.J. Rocchio, *Relevance feedback in information retrieval*, The SMART retrieval system: Experiments in automatic document processing (G. Salton, ed.), Series in automatic computation, Prentice-Hall, 1971, ISBN 0-13-814525-3, pp. 313–323.

[SB88] G. Salton and C. Buckley, *Term-Weighting Approaches in Automatic Text Retrieval*, Information Processing & Management **24** (1988), no. 5, 513–523.

[Sch93] P. Schäuble, *SPIDER: A Multiuser Information Retrieval System for Semistructured and Dynamic Data*, SIGIR '93: Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM SIGIR, ACM, June 1993, pp. 318–327.

[Ses98] P. Seshradri, *Enhanced abstract data types in object-relational databases*, The VLDB Journal **7** (1998), no. 3, 130–140.

[SJ72] K. Sparck-Jones, *A Statistical Interpretation of Term Specifity and its Application in Retrieval*, Journal of Documentation **28** (1972), no. 1, 11–20.

[SLR96] P. Seshadri, M. Livny, and R. Ramakrishnan, *E-ADTs: Turbo-Charging Complex Data*, IEEE Data Engineering Bulletin **19** (1996), no. 4, 11–18.

[SLR97] P. Seshadri, M. Livny, and R. Ramakrishnan, *The Case for Enhanced Abstract Data Types*, Proceedings of the 23th VLDB Conference, VLDB, 1997, pp. 66–75.

[Son96] G. Sonnenberg, *Exploiting the Functionality of Object-Oriented Database Management Systems for Information Retrieval*, IEEE Data Engineering Bulletin **19** (1996), no. 1, 14–23.

[SP82] H.-J. Schek and P. Pistor, *Data Structures for an Integrated Data Base Management and Information Retrieval System*, Proceedings of 8th VLDB Conference, September 1982, pp. 197–207.

[SP97] P. Seshradri and M. Paskin, *PREDATOR: An OR-DBMS with Enhanced Data Types*, In *Proceedings of the 1997 ACM SIGMOD International Conference on the Management of Data* [ACM97], pp. 568–571.

[SR86] M. Stonebraker and L.A. Rowe, *The Design of Postgres*, In *Proceedings of the 1986 ACM SIGMOD International Conference on the Management of Data* [ACM86], pp. 340–355.

[SSM96] D. Simmen, E. Shekita, and T. Malkemus, *Fundamental Techniques for Order Optimization*, In *Proceedings of the 1996 ACM SIGMOD International Conference on the Management of Data* [ACM96], pp. 57–67.

[Ste95] H. Steenhagen, *Optimization of Object Query Languages*, Ph.D. thesis, University of Twente, Enschede, The Netherlands, October 1995.

[TC90] H. Turtle and W.B. Croft, *Inference Networks for Document Retrieval*, Computer and Information Science Technical Report 90-07, University of Massachusetts, Amherst, Massachusetts, February 1990.

[TRE] *Text REtrieval Conference (TREC)*, URL: http://trec.nist.gov/.

[TRE94] *Proceedings of the Third Text Retrieval Conference (TREC-3)*, NIST Special publications, Gaithersburg, Maryland, November 1994.

[TRE97] *Proceedings of the Sixth Text Retrieval Conference (TREC-6)*, NIST Special publications, Gaithersburg, Maryland, November 1997.

[Ull88] J.D. Ullman, *Principles of database and knowledgebase systems*, vol. 2, Principles of computer science series, no. 14, Computer Science Press, Rockville, Maryland, 1988, ISBN 0-7167-8162-X.

[VB98a] A.P. de Vries and H.M. Blanken, *Database technology and the management of multimedia data in miRRor*, Proceedings of SPIE, Multimedia Storage and Archiving Systems III, vol. 3527, November 1998.

[VB98b] A.P. de Vries and H.M. Blanken, *The Relationship between IR and Multimedia Databases*, IRSG'98, the 20th BCS Colloquium on Information Retrieval (M.D. Dunlop, ed.), 1998.

[VCC96] S.R. Vasanthakumar, J.P. Callan, and W.B. Croft, *Integrating INQUERY with an RDBMS to Support Text Retrieval*, IEEE Data Engineering Bulletin **19** (1996), no. 1, 24–33.

[VDBA99] A.P. de Vries, M.G.L.M. van Doorn, H.M. Blanken, and P.M.G. Apers, *The miRRor MMDBMS Architecture*, In Atkinson et al. [AOV+99], pp. 758–761.

[VH99] A.P. de Vries and D. Hiemstra, *The miRRor DBMS at TREC*, Proceedings of the Eigth Text Retrieval Conference (TREC-8) (Gaithersburg, Maryland), NIST Special publications, November 1999, pp. 725–734.

[Vri98] A.P. de Vries, *MiRRor: Multimedia Query Processing in Extensible Databases*, 14th Twente Workshop on Language Technology, Language Technology in Multimedia Information Retrieval (TWLT 14) (Enschede, The Netherlands), University of Twente, December 1998, pp. 37–47.

[Vri99] A.P. de Vries, *Content and Multimedia Database Management Systems*, Ph.D. thesis, University of Twente, Enschede, The Netherlands, December 1999.

[VW99] A.P. de Vries and A.N. Wilschut, *On the Integration of IR and Databases*, 8th IFIP 2.6 Working Conference on Data Semantics 8, January 1999, pp. 16–31.

[WB00] R. Weber and K. Böhm, *Trading Quality for Time with Nearest-Neighbor Search*, Proceedings of the 7th Conference on Extending Database Technology (EDBT 2000), Lecture Notes in Computer Science, Springer, March 2000.

[WSK99] M.A. Windhouwer, A.R. Schmidt, and M.L. Kersten, *Acoi: A System for Indexing Multimedia Objects*, International Workshop on Information Integration and Web-based Applications & Services (Yogyakarta, Indonesia), November 1999.

[Yao77a] S.B. Yao, *An Attribute Based Model for Database Access Cost Analysis*, ACM Transactions on Database Systems **3** (1977), no. 1, 45–67.

[Yao77b] S.B. Yao, *Approximating Block Accesses in Database Organizations*, Communications of the ACM **20** (1977), no. 4, 260–261.

[Yao79] S.B. Yao, *Optimization of Query Evaluation Algorithms*, Proceedings of the 1979 ACM SIGMOD International Conference on the Management of Data, ACM Press, June 1979, pp. 133–155.

[ZA00] R. van Zwol and P.M.G. Apers, *The webspace method: On the integration of database technology with information retrieval*, In proceedings of CIKM'00 (Washington, DC.), November 2000.

[Zip49] G.K. Zipf, *Human Behavior and the Principle of Least Effort*, Addison-Wesley, Reading, MA, USA, 1949.

Index

E

F

G

H

I

L

M

N

S

T

U

V

W

X

Z

Summary

Database optimization aspects for information retrieval

The combined querying of highly structured data and content data is becoming more and more important. We focus on the combined querying of structured data and text. Several ways exist to realize query processing capabilities required for this. Continuing on previous research done in the Database Group in Twente, we are mainly interested in the option of integrating IR in a DBMS.

Database technology provides several interesting data management properties such as data independence, flexibility, and efficient and state-of-the-art query processing. Each of these properties would be welcome in the IR field as well.

One of the key components in a DBMS is the query optimizer. As part of the data independence principle queries are stated in a declarative manner at the top level of the layered architecture typical to a DBMS. The query optimizer exploits the properties of the set algebra at the middle level, the logical layer, to change the order of the operators into a more efficient one. Usually finding the most optimal query plan is far too time consuming. However, a quick heuristic that can find a good solution is often enough to achieve acceptable performance. At the bottom level, the physical layer, the query optimizer exploits a cost model to determine which physical operators implementations to use to realize the query plan. This cost model takes several aspects of the database into account, such as the cardinalities of the involved relations, availability of index structures, ordering, etcetera.

IR research has mainly focussed on the question how text data can be retrieved as effective as possible for an acceptable performance. Since generic and flexible data management never has been a prime target of IR research, IR techniques and data storage are often closely tied to each other. In other words, IR algorithms are typically data bound in contrast with the typical data independency incorporated in the architecture of a DBMS. Furthermore, IR queries are typically fixed in structure with varying parameters, e.g., the query

terms, in contrast with the typical wide variety of database query expressions due to their ad hoc nature. As a consequence, IR optimization typically involves the replacement of the entire algorithm for a new, faster one, discarding the old algorithm from then on.

A major class of IR queries are the top-N queries. In the IR field work has been done to optimize these queries by cutting off query processing as soon as mathematically derived requirements show that the top-N is stable. Unfortunately, these techniques typically exploit the element-at-a-time, data bound processing nature used by IR systems.

When integrating IR processing facilities in a DBMS the optimization of the IR part of queries should be adapted to the database manner of query optimization or the DBMS should support new primitives to support IR optimization techniques. Due to the element-at-a-time nature of the IR optimization techniques, which conflicts with the set-based processing of a DBMS, porting to the database approach is not trivial. We propose to fragment the IR metadata in the database to allow subset-at-a-time, i.e., fragment-at-a-time, processing as an attempt to combine the best of both worlds.

Given this context we are interested in the following main research object:

- Can IR top-N queries be optimized in a database manner?

More concretely we look at the following three derived research questions:

- Can we estimate selectivity as a function of the used fragments?
- Can we estimate the quality behavior as a function of the relative size of the used fragments?
- Can horizontal fragmentation facilitate the trading of quality for speed to support set-based IR top-N query optimization?

In Chapter 1 this context and basic problem definitions are described. Chapter 2 concerns the related work regarding database technology, IR technology, and the integration of IR in databases.

We analyze the three derived research questions in more detail in Chapter 3. This chapter also provides an overview of the followed approach. Since fragmentation is a basic tool that we use throughout this thesis, the third research question is handled first, instead of last, from this chapter on.

So, in Chapter 4 we start with the third research question. We fragment the metadata by sorting it ascendingly on the document frequency of the terms and then partitioning it. We perform several experiments that demonstrate that porting IR optimization techniques to the set-based database environment

does not result in performance gains. We also show that by ignoring more of the less interesting terms in the query, i.e., the terms with higher document frequency, speed does increase and quality degrades, as one would expect.

In Chapter 5 we derive and evaluate a mathematical model to estimate the selectivity of a query on the most interesting fragment(s) of the metadata stored in the database. Though more experiments have to be done, and an error analysis is not available yet, the results seem to be quite accurate.

Chapter 6 concerns the derivation of a pair of statistical models, one being a generalization of the other. The first model allows, once trained, prediction of quality implications of ignoring the less interesting query terms for the training collection. The second model, allows training of the first model on one set of collections, and then using it on another collection. The first model performs quite nice. The second model performs nice for a first approach but is not accurate enough for practical use, yet.

We conclude this thesis with Chapter 7. We find that our three derived research questions can be answered positively, though with some remarks for future work. Our results also present good evidence in favor of answering our main research question with yes as well. However, several issues need to be investigated to obtain more certainty on this. More concretely, we mention six issues for future work. The first three concern extensions of the experiments for the three derived research questions, respectively. The second group of three mostly concerns architectural issues that need to be taken care of to allow the results presented in this theses to be exploited in DBMSs with integrated IR in practice.

Samenvatting

Database optimalisatie aspecten voor information retrieval

Het gecombineerd bevragen van gestructureerde gegevens en content gegevens wordt steeds belangrijker. Wij richten ons op het gecombineerd bevragen van gestructureerde gegevens en tekst. Er bestaan diverse manieren om de hiervoor benodigde *query*-verwerking te realiseren. We zijn hoofdzakelijk geïnteresseerd in de mogelijkheid van de integratie van IR in een DBMS, voortbouwend op onderzoek dat in het verleden is gedaan in de Database Groep in Twente.

Database technologie voorziet in een aantal interessante eigenschappen ten behoeve van gegevens-beheer, zoals: data onafhankelijkheid, flexibiliteit en efficiënt en *state-of-the-art query*-verwerking. Deze eigenschappen zouden in het IR-vakgebied ook van pas komen.

Een van de sleutel-onderdelen in een DBMS is de *query optimizer*. Als onderdeel van het principe van dataonafhankelijkheid worden *queries* op een declaratieve manier gesteld op het top niveau van de typische gelaagde architectuur van een DBMS. De *query optimizer* gebruikt de eigenschappen van de verzamelingsalgebra van het middelste laag, het logische niveau, om de volgorde van de operatoren te veranderen in een efficiëntere. Normaliter is het vinden van het meest optimale *query plan* veel te tijdrovend. Echter, een snelle heuristiek die een goede oplossing vindt, is vaak voldoende om acceptabele prestaties te krijgen. Op de onderste laag, het fysieke niveau, gebruikt de *query optimizer* een kostenmodel om te bepalen welke fysieke operator-implementaties gebruikt moeten worden om het *query plan* te realiseren. Dit kostenmodel houdt rekening met diverse aspecten van de database, zoals de cardinaliteiten van de desbetreffende relaties, de beschikbaarheid van index-structuren, ordening, etcetera.

IR-onderzoek heeft zich vooral gericht op de vraag hoe tekstuele gegevens zo effectief mogelijk, en met een acceptabele snelheid, kunnen worden doorzocht. Omdat generiek en flexibel gegevens-beheer nooit een hoofddoel was voor IR-

onderzoek, zijn de gebruikte technieken en gegevens-opslag vaak sterk met elkaar verweven. Met andere woorden, IR-algorithmen zijn typisch *data bound* in tegenstelling tot de typische dataonafhankelijkheid die is verwerkt in de architectuur van een DBMS. Bovendien hebben IR-*queries* doorgaans een vaste structuur en variëren alleen de parameters, d.w.z. de zoektermen. Database *query* expressies vertonen daarentegen een grote variëteit vanwege hun ad hoc karakteristiek. Het gevolg is dat IR-optimalisatie meestal neerkomt op het vervangen van het gehele algorithme door een nieuw, sneller algorithme. Het oude algorithme wordt vervolgens weggegooid.

Een grote groep van IR-*queries* wordt gevormd door de top-N-queries. In het IR-vakgebied is er veel werk gedaan om deze *queries* te optimaliseren door het afkappen van de *query*-verwerking. Zodra wiskundig afgeleide criteria aangeven dat de top-N een stabiele toestand heeft bereikt wordt de verwerking gestopt. Helaas gaan deze technieken uit van de *element-at-a-time*, *data bound* wijze van verwerken in IR-systemen.

Als we IR-verwerkingsfaciliteiten willen integreren in een DBMS moet de optimalisatie van het IR-deel van de *queries* worden aangepast aan de manier waarop een *query optimizer* van de DBMS werkt. Een andere optie is om de DBMS uit te rusten met ondersteuning voor nieuwe primitieven ter ondersteuning van IR-optimalisatie. Echter, de *element-at-a-time* karakteristiek van IR-optimalisatie-technieken botst met de verzamelinggebaseerde verwerkingswijze van een DBMS. Daarom is het overzetten van die technieken niet triviaal. Ons voorstel is om de IR-metadata in de database te fragmenteren. Hierdoor wordt *subset-at-a-time*, ofwel *fragment-at-a-time*, verwerking mogelijk. We pogen hiermee om het beste van beide gebieden te combineren.

Gegeven deze context zijn we geïnteresseerd in het volgende hoofd-onderzoeksdoel:

- Kunnen IR-top-N-*queries* worden geoptimaliseerd op een databasemanier?

Meer concreet zijn we geïnteresseerd in de volgende drie afgeleide onderzoeksvragen:

- Kunnen we de selectiviteit schatten als functie van de gebruikte fragmenten?
- Kunnen we de gevolgen voor de antwoord-kwaliteit schatten als functie van de relatieve grootte van de gebruikte fragmenten?
- Kan horizontale fragmentatie helpen bij het uitwisselen van snelheid en kwaliteit ter ondersteunening van verzamelinggebaseerde IR-top-N-*query*-optimalisatie?

In hoofdstuk 1 beschrijven we deze context en het basisprobleem. Hoofdstuk 2 betreft het gerelateerde onderzoek aangaande database technologie, IR-technologie en de integratie van IR in databases.

We analyseren de drie afgeleide onderzoeksvragen in meer detail in hoofdstuk 3. Dit hoofdstuk bevat ook een overzicht van de gevolgde aanpak. Omdat fragmentatie een middel is dat we gebruiken in dit hele proefschrift behandelen we de derde onderzoeksvraag als eerste, in plaats van laatste vanaf dit hoofdstuk.

In hoofdstuk 4 beginnen we dus met de derde onderzoeksvraag. We fragmenteren de metadata door deze oplopend op de document-frequentie van de termen te sorteren. Daarna partitioneren we de metadata. We voeren diverse experimenten uit en laten zien dat het overzetten van IR-optimalisatietechnieken naar de verzamelinggebaseerde database-omgeving niet resulteert in prestatieverbeteringen. Ook laten we zien dat, zoals verwacht, de snelheid toe- en de kwaliteit afneemt bij het negeren van een groter aantal van de minder interessante termen in de *query*, d.w.z. de termen met een hogere document-frequentie.

Hoofdstuk 5 gaat over de afleiding en evaluatie van een wiskundig model om de selectiviteit te schatten van een *query* op het/de interessantste fragment(en) van de metadata in de database. Er moeten nog meer experimenten gedaan en een fout-analyse gemaakt worden. Maar desalniettemin lijken de resultaten behoorlijk nauwkeurig.

In hoofdstuk 6 leiden we een tweetal statistische modellen af. Hiervan is het ene model een generalisatie van het andere. Het eerste model maakt het mogelijk om, zodra het getrained is, de gevolgen voor de antwoord-kwaliteit te voorspellen voor de trainingscollectie als minder interessante *query*-termen worden genegeerd. Het tweede model maakt het mogelijk het eerste model te trainen op een andere verzameling collecties dan waarvoor de kwaliteitsvoorspellingen gemaakt moeten worden. Het eerste model werk redelijk goed. Het tweede model presteert goed voor een eerste opzet maar is nog niet voldoende nauwkeurig om praktisch toepasbaar te zijn.

We sluiten dit proefschrift af met hoofdstuk 7. We concluderen dat onze drie afgeleide onderzoeksvragen positief beantwoord kunnen worden, zei het met wat kanttekeningen voor vervolgonderzoek. Onze resultaten leveren ook goede argumenten om onze hoofd-onderzoeksvraag ook met ja te beantwoorden. Echter, er zijn nog een aantal zaken die verder onderzocht moeten worden om hier echt zekerheid over te krijgen. We noemen zes concrete onderwerpen voor verder onderzoek. De eerste drie betreffen nadere experimenten voor de drie respectievelijke afgeleide onderzoeksvragen. De twee groep van drie gaat over architecturele aanpassingen die doorgevoerd zouden moeten worden om de resultaten van dit proefschrift in de praktijk te kunnen gebruiken in DBMS-en met geïntegreerde IR.

Curriculum Vitae

Henk Ernst Blok was born on May 23, 1974 in Hoofddorp, in the Netherlands. He completed secondary school (VWO) at 'De Waezenburg' in Leek in 1992.

In the same year he started as student at the University of Twente. Under supervision of prof.dr. C. Hoede he obtained his M.Sc. in Applied Mathematics, with an endorsement for knowledge engineering, in September 1997 after having completed his thesis entitled *Knowledge Graph Framing and Exploitation*.

In October 1997 he started as a Ph.D. student of prof.dr. P.M.G. Apers and dr. H.M. Blanken on the Advanced Multimedia Indexing and Searching (AMIS) project, funded by the Netherlands Organization for Scientific Research (NWO), in the Database group of the Faculty of Computer Science of the University of Twente. His research was situated in the area of integration of information retrieval technology in database systems. The focus of his research was on query optimization, exploiting the trade-off between speed and quality typical to the area of information retrieval optimization. His dissertation is entitled *Database Optimization Aspects for Information Retrieval*.

Next to his research activities he was also actively involved in a couple of organizations related to the university or the database world.

He hopes to continue doing research in the area of core database technology for new application domains with a special interest in applying mathematical modeling and analysis techniques.

SIKS Dissertation Series

[98-1] Johan van den Akker (CWI), *DEGAS – An Active, Temporal Database of Autonomous Objects*

[98-2] Floris Wiesman (UM), *Information Retrieval by Graphically Browsing Meta-Information*

[98-3] Ans Steuten (TUD), *A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective*

[98-4] Dennis Breuker (UM), *Memory versus Search in Games*

[98-5] E.W. Oskamp (RUL), *Computerondersteuning bij Straftoemeting*

[99-1] Mark Sloof (VU), *Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products*

[99-2] Rob Potharst (EUR), *Classification using decision trees and neural nets*

[99-3] Don Beal (Queen Mary and Westfield College), *The Nature of Minimax Search*

[99-4] Jacques Penders (KPN Research), *The practical Art of Moving Physical Objects*

[99-5] Aldo de Moor (KUB), *Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems*

[99-6] Niek J.E. Wijngaards (VU), *Re-design of compositional systems*

[99-7] David Spelt (UT), *Verification support for object database design*

[99-8] Jacques H.J. Lenting (UM), *Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.*

[2000-1] Frank Niessink (VU), *Perspectives on Improving Software Maintenance*

[2000-2] Koen Holtman (TUE), *Prototyping of CMS Storage Management*

[2000-3] Carolien M.T. Metselaar (UvA), *Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief.*

[2000-4] Geert de Haan (VU), *ETAG, A Formal Model of Competence Knowledge for User Interface Design*

[2000-5] Ruud van der Pol (UM), *Knowledge-based Query Formulation in Information Retrieval.*

[2000-6] Rogier van Eijk (UU), *Programming Languages for Agent Communication*

[2000-7] Niels Peek (UU), *Decision-theoretic Planning of Clinical Patient Management*

[2000-8] Veerle Coup (EUR), *Sensitivity Analyis of Decision-Theoretic Networks*

[2000-9] Florian Waas (CWI), *Principles of Probabilistic Query Optimization*

[2000-10] Niels Nes (CWI), *Image Database Management System Design Considerations, Algorithms and Architecture*

[2000-11] Jonas Karlsson (CWI), *Scalable Distributed Data Structures for Database Management*

[2001-1] Silja Renooij (UU), *Qualitative Approaches to Quantifying Probabilistic Networks*

[2001-2] Koen Hindriks (UU), *Agent Programming Languages: Programming with Mental Models*

[2001-3] Maarten van Someren (UvA), *Learning as problem solving*

[2001-4] Evgueni Smirnov (UM), *Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets*

[2001-5] Jacco van Ossenbruggen (VU), *Processing Structured Hypermedia: A Matter of Style*

[2001-6] Martijn van Welie (VU), *Task-based User Interface Design*

[2001-7] Bastiaan Schonhage (VU), *Diva: Architectural Perspectives on Information Visualization*

[2001-8] Pascal van Eck (VU), *A Compositional Semantic Structure for Multi-Agent Systems Dynamics.*

[2001-9] Pieter Jan 't Hoen (RUL), *Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes*

[2001-10] Maarten Sierhuis (UvA), *Modeling and Simulating Work Practice – BRAHMS: a multiagent modeling and simulation language for work practice analysis and design*

[2001-11] Tom M. van Engers (VU), *Knowledge Management: The Role of Mental Models in Business Systems Design*

[2002-01] Nico Lassing (VU), *Architecture-Level Modifiability Analysis*

[2002-02] Roelof van Zwol (UT), *Modelling and searching web-based document collections*

[2002-03] Henk Ernst Blok (UT), *Database Optimization Aspects for Information Retrieval*

Explanation of used institute acronyms:

CWI – Centrum voor Wiskunde en Informatica / National Research Institute for Mathematics and Computer Science, Amsterdam
EUR – Erasmus Universiteit Amsterdam / Erasmus University Amsterdam
KUB – Katholieke Universiteit Brabant / Tilburg University
RUL – Universiteit Leiden / Leiden University
TUD – Technische Universiteit Delft / Delft University of Technology
TUE – Technische Universiteit Eindhoven
UM – Universiteit Maastricht
UT – Universiteit Twente / University of Twente
UU – Universiteit Utrecht / Utrecht University
UvA – Universiteit van Amsterdam
VU – Vrije Universiteit Amsterdam