INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

DOCUMENT RANKING ON WEIGHT-PARTITIONED SIGNATURE FILES

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the
Graduate School of The Ohio State University

By

Liming Ren, M.S.

* * * * *

The Ohio State University

1998

Dissertation Committee:

Professor Mukesh Singhal, Adviser

Professor Douglas S. Kerr

Professor Harvey Friedman

Approved by

A handwritten signature in black ink, appearing to read 'Mukesh Singhal', written over a horizontal line.

Adviser

Department of Computer
and Information Science

UMI Number: 9834053

UMI Microform 9834053
Copyright 1998, by UMI Company. All rights reserved.

**This microform edition is protected against unauthorized
copying under Title 17, United States Code.**

UMI
300 North Zeeb Road
Ann Arbor, MI 48103

ABSTRACT

With the rapid development and spread of information technologies, especially the Internet, information retrieval applications are no longer just limited to library systems and office automation systems. They become an integral part of human life in the information society. Information retrieval research has received great attention in both the traditional and new application domains because large amount of information demands efficient and accurate information retrieval techniques. In this thesis, we study methods to implement document ranking using the signature file, one of the main information retrieval models. Our main goals are to improve retrieval effectiveness and system response time.

A signature file organization, called the weight-partitioned signature file, for supporting document ranking is proposed. It employs multiple signature files, each of which corresponds to one term frequency, to represent terms with different term frequencies. Words with the same term frequency in a document are grouped together and hashed into the signature file corresponding to that term frequency. This eliminates the need to explicitly record the term frequency for each word.

We investigate the effect of false drops on retrieval effectiveness if they are not eliminated in the search process. We have shown that false drops introduce insignificant degradation on precision and recall when the false drop probability is below

a certain threshold. This is an important result since false drop elimination could become the bottleneck in systems using fast signature file search techniques.

We perform an analytical study on the performance of the weight-partitioned signature file under different search strategies and configurations. An optimal formula is obtained to determine for a fixed total storage overhead the storage to be allocated to each partition in order to minimize the effect of false drops on document ranks. Experiments were performed using two document collections to support the analytical results.

We also study two approaches to improve the response time of weight-partitioned signature files. In the first approach, we propose several fast heuristic search algorithms to reduce the percentage of signatures that needs to be searched. On average they achieve search reduction ratios in the range of 40% to 65% without sacrificing any precision and recall for top documents. In the second approach, we try to reduce the total size of the signature files, since space is speed. The main idea is that we establish a coarse ranking by searching signature files first: then we do exact text matching on the top documents obtained. Our results show that comparable precision can be achieved with at least 37% less storage overhead. For long document collection, we achieve the same precision and recall with 75% less of storage overhead.

In the second part of this dissertation, we propose a new key-based partitioning method, variable-prefix partitioning, to improve signature file search speed. We obtain several fast and accurate analytical functions for estimating search space reduction ratio, page fill factor etc. for the variable-prefix method. We also estimate how much space can be saved by storing only one copy of the partition keys.

ACKNOWLEDGMENTS

I wish to express gratitude to my advisor, Prof. Mukesh Singhal, for guiding me through this dissertation.

I owe a great debt to Prof. Dik Lee and Prof. Singhal, who guided me expertly in the transition from study to research of computer science. Most results in this dissertation, together with less tangible aspects of my research, are a product of their influences. Their encouragements, understandings and patience are greatly appreciated.

I also wish to thank Prof. Doug Kerr and Prof. H. Friedman for serving as members of my reading committee and reviewing the manuscript.

I wish to thank my wife, Shuying Li, for statistical consulting. I wish to thank my parents for their constant encouragements.

Finally, I wish to thank my daughter, Kelly, being there through this entire ordeal. The love and support you have given me these past years are beyond words.

VITA

June, 13, 1963	Born - Shanxi, China
September, 1979 - July 1983	B.S., Mathematics. Nankai University, Tianjin, China
September, 1983 - July 1986	M.S., Mathematics, Nankai University, Tianjin, China
September, 1991 - March 1992	M.S., Department of Computer and In- formation Science, The Ohio State Uni- versity, Columbus, Ohio
September, 1992 - August, 1994	Graduate Teaching Associate, Depart- ment of Computer and Information Science, The Ohio State University. Columbus, Ohio
July 1995-present	Software Design Engineer. Microsoft Corporation. Redmond. Washington

PUBLICATIONS

Research Publications

D. Lee, L. Ren. Document Ranking on Weight-Partitioned Signature Files, ACM Transactions on Information Systems, 14(2):107-137, 1996

FIELDS OF STUDY

Major Field: Computer and Information Science

Studies in:

Information Retrieval and Database Systems	Prof. Mukesh Singhal
Information Retrieval and Database Systems	Prof. Dik L. Lee
Mathematics Logic	Prof. Tim Carlson
Parallel Computing	Prof. C. H. Huang

TABLE OF CONTENTS

	Page
Abstract	ii
Acknowledgments	iv
Vita	v
List of Tables	ix
List of Figures	x
Chapters:	
1. Introduction	1
1.1 Overview	1
1.2 Document Indexing Methods	6
1.2.1 Document Ranking Using Inverted Files	6
1.2.2 Document Ranking Using Signature Files	7
1.3 Description of the Problems	10
1.4 Organization of the Dissertation	13
2. Current IR research	15
3. Weight-partitioned Signature Files	24
3.1 Term Weights and Similarity Measure	24
3.2 Weight-partitioned Signature File	26
3.2.1 Encoding Term Frequency in Signature File	26
3.2.2 Search Strategies	30
3.2.3 Signature File Configuration	32
3.3 Analytical Results	32
3.3.1 HL Versus LH Search Strategies	33

3.3.2	Optimal Signature File Configuration	40
3.3.3	Effect of False Drops on Document Ranks	48
3.4	Experimental Results	52
4.	Fast Heuristic Search Algorithms	70
4.1	Search Space Reduction in HL Search Method	70
4.2	Upperbound Search Algorithm in Inverted File	75
4.3	Upperbound Search Algorithms for Weight-partitioned Signature File	76
4.4	Search Reduction Ratios of Heuristic Algorithms	82
4.5	Weight-Partitioned Signature File Under Low Storage	85
4.5.1	Exact Search for Top Documents	86
4.5.2	High Frequency Term Indexing	87
5.	Key Partitioned Signature Files	94
5.1	Variable-prefix Partitioning	95
5.2	Floating Key Partitioning	112
5.3	Storage Saving	119
6.	Conclusion and Future Work	125
6.1	Summary and Conclusion	125
6.2	Future Research	127
	Bibliography	131

LIST OF TABLES

Table	Page
3.1 Symbols for TF-partitioned Signature Files	25
3.2 Characteristics of the document collections	54
3.3 Characteristics of the query collections	54
5.1 Symbols for Variable-prefix Partitioned Signature Files	96

LIST OF FIGURES

Figure	Page
1.1 Keywords distribution in documents	6
1.2 An example of inverted file	6
1.3 Signature generation and comparison based on superimposed coding .	9
3.1 An example of weight-partitioned signature files	28
3.2 Retrieval system based on weight-partitioned signature files	29
3.3 Effects of search strategies	30
3.4 Distribution of true matches in CACM	56
3.5 Distribution of true matches in TREC	56
3.6 Performance of LH versus HL search strategies on CACM	59
3.7 Performance of LH versus HL search strategies on TREC	60
3.8 Comparison of signature length assignment methods (CACM, HL) . .	61
3.9 Comparison of signature length assignment methods (TREC subset, HL)	62
3.10 Retrieval accuracy for top 10, 20 and 50 documents on CACM	65
3.11 Retrieval accuracy for top 10, 20 and 50 documents on TREC	66
3.12 Number of false drops (FDs) and weighted false drop probability (FDP) for CACM (HL)	67

3.13	Number of false drops (FDs) and weighted false drop probability (FDP) for TREC (HL)	68
4.1	Search reduction ratio (CACM, HL)	73
4.2	Search reduction ratio (TREC, HL)	73
4.3	Search reduction ratio for top 20 documents (CACM, HL)	84
4.4	Search reduction ratio for top 20 documents (TREC, HL)	84
4.5	Exact match for top 50 and 100 documents (CACM, uniform)	88
4.6	Exact match for top 50 and 100 documents (CACM, optimal)	88
4.7	Exact match for top 50 and 100 documents (TREC, uniform)	89
4.8	Exact match for top 50 and 100 documents (TREC, optimal)	89
4.9	Performance of high TF term indexing (CACM, uniform)	92
4.10	Performance of high TF term indexing (CACM, optimal)	92
4.11	Performance of high TF term indexing (TREC, uniform)	93
4.12	Performance of high TF term indexing (TREC, optimal)	93
5.1	Pages generated (Variable-prefix)	110
5.2	Pages searched (Variable-prefix)	110
5.3	Signature search reduction ratio	112
5.4	Number of pages generated for $m = 512$ and $k = 32$	117
5.5	Percentage of signatures searched for $m = 512$, $k = 32$ and $w_Q = 0.1172$	117
5.6	Maximal space saving by not storing Keys	122
5.7	Space saved by not storing keys	122

5.8	Optimal key length	123
-----	------------------------------	-----

CHAPTER 1

Introduction

1.1 Overview

Information retrieval (IR) research has received a lot of attention recently, especially with the tremendous success of the World Wide Web. An unprecedented amount of information in the form of written texts, speeches, graphics is becoming available daily around the world. The need for efficient and effective information retrieval systems is ever increasing. Information retrieval systems have been used in office automation, software engineering and to provide access to books, journals, and other online documents. In other words, IR systems are used in any areas which use documents. It is no surprise that the most popular and useful web sites over the Internet are search engine sites. Even with the amount of multimedia data available, text retrieval is still the most important function in any information retrieval systems, because a large portion of data stored online is still in text. In this dissertation, we study new techniques to make IR systems fast and accurate.

In the late 1950s, Luhn [35] first suggested that the automatic text retrieval systems could be designed based on a comparison of content identifiers from the stored texts and user queries. Over the years, information retrieval systems have improved gradually. Various mathematical models have been proposed to represent content

identifier based information retrieval systems in the last 40 years. Several approaches have received the most attention, notably the Boolean model [3, 46, 49] which compares Boolean query statements with the term sets used to identify document content. the vector space model [46] which represents both queries and documents by vectors and computes similarities between queries and documents. and the probabilistic model based on the computation of relevance for the documents of a collection. Of these models, the vector space model has been the most popular one because of its simplicity and effectiveness.

The Boolean model and vector space model [46] represent both documents and queries as term vectors in the form

$$D = \langle w_1, w_2, \dots, w_V \rangle$$

$$Q = \langle w_{Q,1}, w_{Q,2}, \dots, w_{Q,V} \rangle,$$

where V is the size of the dictionary (or vocabulary), w_j and $w_{Q,j}$ are term weights representing the importance of term j in document D and query Q , respectively.

For Boolean model, the term weight w_j (or $w_{Q,j}$) is set to 1 when term j appears in document D (or in query Q) and 0 otherwise. In other words, Boolean model ignores much of the syntactic, semantic and contextual information. Furthermore it cannot support document ranking effectively. The retrieval effectiveness of Boolean systems has been demonstrated to be unsatisfactory [3, 46, 49]. User needs to know how to formulate complex boolean queries using logical operators like “AND”, “OR” and “NOT” etc. Boolean systems are hard to use.

To improve the effectiveness of IR systems, a term weight in the vector space model could take on any numeric values, which, when selected properly, represents the relative importance of the term in the respective document and query.

If we identify each term in the vocabulary with a vector and span a linear space from those V vectors over real number space, we get the document space. Each vector in this space, theoretically, can be viewed as a document. Hence, A document D in this space can be written as:

$$D = \sum_{j=1}^V w_j T_j$$

where T_j is the vector associated with term j , w_j is interpreted as the component of document D along the vector T_j or term j . The similarity between a document D and Q is measured by their inner product

$$\langle D, Q \rangle = \sum_{i,j=1}^V w_i w_{Q,j} \langle T_i, T_j \rangle$$

which geometrically is $|D||Q| \cos \alpha$, where α is the angle between the two vectors and $|D|$ and $|Q|$ the lengths of D and Q , respectively, in the vector space.

It is clear that the vectors $\{T_i, i < V\}$ form a basis for the generated vector space. It is not necessary that the T_i 's are linearly independent with respect to each other. For example, if terms t_j and t_k are synonym in a specific text database, it is possible to assume that T_j and T_k are linearly dependent and may even be identical. In general, $\langle T_i, T_j \rangle, i, j < V$, the term-correlations, are not available. In practice, the term vectors T_i 's are assumed to be linearly independent and orthogonal in the vector space model for simplicity. The document space is a V dimensional real linear space with normalized basis $\{T_1, T_2, \dots, T_V\}$. The cardinality of the space is clearly uncountable. But only a small and finitely many vectors are representations of real

documents. Under this assumption, the similarity between a document D and Q becomes their inner product

$$\langle D, Q \rangle = \sum_{j=1}^V w_j w_{Q,j}$$

The angle between D and Q , $\alpha = \arccos(\frac{\langle D, Q \rangle}{|D| \times |Q|})$, in some degree, represents the relevance of document D with respect to query Q . By using the angle, instead of the inner product, we effectively normalize the similarity value to numbers in interval $[0, 1]$. For this reason, the angle similarity is the most widely used similarity measure. By this similarity measure, documents can be sorted according to decreasing order and only the desired top documents are returned to the user. Terms in the top documents, which are most similar to the queries, may help generate improved queries using relevance feedback. This model has been used as the basis for many ranking retrieval experiments, in particular the SMART system [44] developed by G. Salton and his group. It is also used extensively in our research and in this dissertation.

The original vector space model did not specify how to select the term weights. In the IR community, most researchers use some combination of term frequency and document frequency, which will be elaborated in the next section.

An IR system can be evaluated in terms of many criteria including response time, retrieval effectiveness, storage overhead and the features it offers. Response time is the time it takes a system to return results to the users. Slow response time will affect the usefulness of the system. Storage overhead is measured by the storage needed to store the index file. It can be calculated as the ratio of the size of the index file over the size of the document files.

To reduce the storage overhead, most IR systems perform stemming and stop word removal before indexing. Common terms which carry little or no information

like “as”, “of” and “the” are not indexed. We will use single term identifier approach in our research.

Most IR research focuses on retrieval effectiveness. Two main parameters of retrieval effectiveness, precision and recall, have been used over the years by IR community to evaluate retrieval systems. Mathematically, they can be expressed as:

$$\text{Precision} = \frac{\text{Number of relevant documents retrieved}}{\text{Total number of documents retrieved.}}$$

$$\text{Recall} = \frac{\text{Number of relevant documents retrieved}}{\text{Total number of relevant documents in collection}}$$

Recall and precision seem to be inversely related. That is, when precision goes up, recall typically goes down and vice versa. The ideal and perfect retrieval system should achieve 100% precision and 100% recall. However, in reality, most systems are far from reaching that goal. They have a limited recall and precision, meaning that users may retrieve a few relevant documents mixed with some irrelevant ones in response to their queries, but never all and only the relevant documents.

Most IR researchers use test collections to evaluate their IR systems. The test collections usually consist of a document database and a set of queries for the database for which relevance judgments (precision and recall) are available. More and more researchers are endorsing the use of the TREC (Text REtrieval Conference) text corpuses, a large document collection from NIST and DARPA, to provide a means of evaluation of text search strategies. Currently, Unix is still the main platform for information retrieval research.

	term 1	term 2	term 3	term 4
Doc 1	1.3	0.0	0.4	0.0
Doc 2	0.0	0.0	0.21	1.90
Doc 3	0.7	0.5	0.21	0.341

Figure 1.1: Keywords distribution in documents

term 1	→	Doc 1, 1.3; Doc 3, 0.7;
term 2	→	Doc 3, 0.5;
term 3	→	Doc 1, 0.4; Doc 2, 0.21; Doc 3, 0.21;
term 4	→	Doc 2, 1.90; Doc 3, 0.341

Figure 1.2: An example of inverted file

1.2 Document Indexing Methods

Among many different information retrieval techniques, two methods stand out and have received the most attention. One is inverted files and the other is signature files. In this section, we give a detailed description of inverted files and signature files.

1.2.1 Document Ranking Using Inverted Files

The concept of inverted file method is as follows. Each document in the document collection is assigned a list of keywords or attributes, with optional relevance weights associated with each keyword. An inverted file is then a sorted list (or index) of keywords, with each keyword having links to the documents containing that keyword.

Figure 1.1 presents a document collection of 3 documents with a dictionary of 4 terms. The decimal numbers are weights carried by the terms in their hosting documents. Transposing the document representation matrix, we have Figure 1.2. Each row in the second matrix is called a postings list. The inverted file consists of all postings lists. It is clear that other information like term frequency, byte offset and line number can also be stored in the postings list to facilitate queries with location requirements. Inverted file indexing can be found in most commercial library systems and search engine sites. The use of an inverted file improves search efficiency by several orders of magnitude compared to full text scanning, a necessity for large text databases. The penalty paid for this efficiency is the need to store a data structure that ranges from 100 to 300 percents of the size of the text itself, and a need to update the index as the data set changes, which is expensive. The storage overhead of inverted file sometime deems the system unusable. For example, it is impractical to ask average users to double their disk space in order to use an IR system to manage their documents.

An inverted file usually is organized as a sorted list or a B-tree. The most straightforward way to search an inverted file is by brute force. All postings lists related to the query have to be retrieved and processed. There have been attempts to avoid exhaustive search of the postings lists. We will present a brief review on this subject in later sections.

1.2.2 Document Ranking Using Signature Files

The signature file approach has been investigated quite extensively in the last decade. It has been applied to a large variety of applications, including textual

databases such as news databases [Stanfill and Kahle 1986], multi-attribute retrieval in relational databases [Aho and Ullman 1979; Lee and Leng 1990], multimedia office filing [Christodoulakis et al. 1986], and chemical databases for DNA matching [Lipman and Pearson 1985].

Superimposed coding is the most common way for generating signatures from a text. In superimposed coding, a text is divided into text blocks containing the same number of unique, nontrivial words. Each word in a text block is hashed into a word signature. A block signature is generated by superimposing all word signatures generated from the block. Given a query, the query terms are hashed and superimposed into a query signature in the same way. Then the query signature is matched against each signature in the signature file. Figure 1.3 is an example showing the generation of the block signature from a text block.

The signature file is a filtering mechanism which eliminates most, but not all, of the text blocks which will not match the query. The first case shown in Fig. 1.3 illustrates this point. The query signature doesn't "match" with the text signature in that some of the bits in the text signature are zero while the corresponding bits in the query signature are set to one. If the query term "retrieval" is indeed in the text, the query signature would be one of the word signatures forming the text signature: thus, every bit set in the query signature will be set in the text signature. The second case shows a match, where for each bit in the query signature set to one, the corresponding bit in the block signature is also set to one. The third case is a false drop. False drops are text blocks which the signature file identifies as containing the query terms (i.e., a match) but indeed they don't. They can be eliminated by further comparing the query terms with the text blocks.

text block	<div>... text database ...</div>		
word signatures:			
text	001	000	110 010
database	000	010	101 001
<hr/>			
block signature (V)	001	010	111 011
Queries	Query Signatures	Results	
1) retrieval	010 001 000 011	← no match	
2) database	000 010 101 001	← match	
3) information	001 000 111 000	← false drop	

Figure 1.3: Signature generation and comparison based on superimposed coding

C. S. Roberts [43] proved that the false drop probability is minimized when the probability of any bit position set to 1 is 0.5. Compared to inverted files, the signature file approach has two major advantages:

- Its storage overhead can be controlled easily and in general is very low. Therefore, if only a limited space for indexes is available, the signature length can be set so that the whole signature file will fit into the given space.
- Its structure is very simple. Thus, it has low processing overhead on insertion. Since signatures are conceptually organized as a sequential file, it is easy to partition a signature file for parallel processing [Stanfill and Kahle 1986].

It is generally believed [14, 41] that a signature file usually incurs 10-20% storage overhead. In this dissertation, we will supply quantitative data to confirm this belief.

1.3 Description of the Problems

However, the signature file method has some serious drawbacks which prevent it to be used in large and advanced IR systems effectively. In this section, we identify and discuss these problems. The goal of this dissertation is to implement document ranking strategy using signature file in a way which supports document ranking and fast retrieval.

Encoding Term Frequency in Signature Files

First it is not easy to implement document ranking in signature files. A signature encodes the presence of terms in a text. Therefore, it can easily support the Boolean retrieval model. However, in order to support document ranking, term frequencies must be stored; this is difficult to do on signatures, which are simple bit vectors. Croft and Savino [9] provide a ranking technique that combines the inverse document frequency with an estimated term frequency. The retrieval effectiveness suffers because of the approximation of the term frequency.

Slow Response Time

Another problem with signature file is that its search speed is low compared to inverted files, since the signature file has to be searched exhaustively if signatures are organized in a single sequential file. Since the size of the signature file is directly proportional to the size of the database, retrieving all or some of the top documents satisfying a query requires exhaustive search of the whole signature file. Therefore, the time spent on searching the signature file is proportional to the database size, and performance becomes a major concern for large databases.

Problems of False Drops

Signature files introduce false drops which are expensive to eliminate. False drop is an inherent property of signature file. The false drop probability, defined as the ratio of the number of false drops to the number of unqualified signatures, is dependent on the size of the signatures and thus on the storage overhead. False drop probability can be reduced arbitrarily by using a longer signature length while keeping the number of distinct keywords in a text block the same. However, to eliminate false drops completely, it is necessary to retrieve the text and perform pattern matching between the query term(s) and the text after matching the signature file. This process imposes a great penalty on the speed of signature file. The cost of false drop elimination is particularly acute for large collections since the number of false drops is proportional to the number of signatures and thus to the size of the collection. Furthermore, when queries are not highly selective, a large number of matches will be returned and each of the matches has to be verified with pattern matching. The high cost raises a serious concern that false drop elimination will become the bottleneck in retrieval, especially in systems employing fast signature search techniques [Lee et al. 1994]. In other words, the search speed will be slow no matter how fast the signature file itself can be searched.

In this dissertation, we attempt to address all three drawbacks of signature files listed above. We first present a technique, called the weight-partitioned signature file, for implementing document ranking with signature files. The technique we propose does not represent term frequency values explicitly. Furthermore, it also avoids exhaustive search on the signature file. An analytical study is performed on various

performance issues of the weight-partitioned signature file. We also present an extensive evaluation on the degradation of retrieval effectiveness if false drops are not eliminated from the signature file. We also propose several fast heuristic search algorithms to improve the slow response time problem. In these algorithms, the search space is reduced by 50% to 75% with little or no sacrifice on precision and recall.

In this dissertation, we investigate the feasibility of retaining the false drops generated by the signature file. For Boolean retrieval, it is problematic since users are accustomed to the semantics of Boolean retrieval that each document retrieved will satisfy the Boolean condition that the user specifies. Compared to the Boolean model, retrieval in statistical models is less predictable in that documents are retrieved based on their relative ranking rather than their absolute scores. Therefore, given a query, it is difficult to tell why a document is retrieved without examining the other documents. This makes the effect of false drops less observable to the user even if the document ranks are slightly perturbed by the false drops. The main results of this thesis is based upon the conjecture that since document ranking is not a precise process (i.e., only some of the retrieved documents are relevant to the users), the addition of some “noise” or perturbation, due to false drops, to the result document set should have a small impact on retrieval effectiveness. If the conjecture is true, then the signature file approach will become a very attractive technique for implementing document ranking.

1.4 Organization of the Dissertation

In this dissertation, we develop new techniques to implement document ranking using signature files. We also study algorithms to facilitate fast and accurate retrieval for signature files. The organization of the dissertation is as follows. Chapter II reviews IR research literatures. It summarizes current research results from IR community.

In Chapter III, we introduce the weight-partitioned signature file, a new signature file organization to support document ranking. Various properties of the new file structure are studied in depth. We also propose a new scheme to allocate storage among different partitions to minimize false drops. The new weight-partitioned signature file based retrieval method achieves comparable precision and recall with significantly less storage overhead than inverted file based systems.

In Chapter IV, we propose two sets of heuristic algorithms to attack the slow response problem of weight-partitioned signature file from two different angles. The first set of algorithms, based on upperbound search algorithm, reduces the percentage of signatures needed to be searched. The second set of algorithms tries to achieve fast response time by reducing the generated signature file size. Various experimental results are presented to support our conclusions.

In Chapter V, we continue the research started by Lee and Leng [30, 31]. A new partition method, called variable-prefix partitioning, is introduced. It achieves better search space reduction ratio. We also estimate the storage saving by storing only one copy of the partition keys.

Finally in Chapter VI, we summarize the research results from this paper and contemplate future research directions.

Some important symbols used in our study of weight-partitioned signature files are summarized in Table 3.1.

CHAPTER 2

Current IR research

There are many active research areas in information retrieval. In this section, we review some recent IR research results with emphasis on signature files.

Document Indexing

When designing a retrieval system, decisions have to be made on what the base content units are and how the term weights are assigned to individual content identifier. The most obvious way to extract identifiers is to use single term identifiers. To improve the precision and recall, many different ways to extract document identifiers have been attempted over the years with little success. Examples are thesauruses [15], stemming [48, 18], related terms based statistical cooccurrence [61] and phrases [50] etc.

G. Salton and C. Buckley [47] surveyed results in the area of retrieval systems for the last twenty five years. They concluded that “judicious use of single term identifiers is preferable to the incorporation of more complex entities extracted from the text themselves or obtained from available vocabulary schedules”. Two main problems with producing complex text identifiers [47] are:

1. When stringent conditions are used for the construction of complex identifiers, typified by the use of restrictive frequency criteria and limited concurrence contexts for the recognition of term phrases, then few new identifiers are likely to become available, and the performance of the retrieval system with complex identifiers will differ only marginally from the results obtainable with single term indexing.
2. On the other hand, when the construction criteria for the complex entities are relaxed, then some good identifiers are obtained. However many identifiers that do not prove useful are also returned.

G. Salton and C. Buckley [47] experimented with 287 distinct combinations of term-weighting assignments. It was shown that appropriately weighted single term indexing systems consistently produce better results than other complex text representations. Among the many different term weighting schemes (to assign or calculate term weight $w_{i,j}$ and $w_{Q,j}$). G. Salton and C. Buckley [47] found that assignments based on some combinations of term frequency, document frequency and length normalization provide the best performance. The best document term weight for the j th term in i th document is given by:

$$w_{i,j} = \frac{tf_{i,j} \cdot \log \frac{N}{df_j}}{\sqrt{\sum_{k=1}^V \left(tf_{i,k} \cdot \log \frac{N}{df_k} \right)^2}}$$

and the best query term weight for the j th term is given by:

$$w_{Q,j} = \left(0.5 + \frac{0.5 * tf_{Q,j}}{\max\{tf_{Q,k} | t_k \in Q\}} \right) \cdot \log \frac{N}{df_j}$$

where N is the number of documents in the collection, $tf_{i,j}$ the term frequency, how often term j occurs in the i^{th} document, and df_j the term frequency, how many

documents contain term j . In general, queries are very short. Most terms have term frequency equal to one in a query. The query term weight formula is basically

$$w_{Q,j} = tf_{Q,j} \cdot \log \frac{N}{df_j}.$$

In the literature, $\log \frac{N}{df_j}$ is often called the inverse document frequency of term j and denoted as idf_j .

Overall, Salton and Buckley suggested that the single term indexing should be preferred. We will use single term identifier approach in our research.

Term Correlations

In the vector space model, all terms in the dictionary are assumed to be mutually independent. This clearly is not true in a real document collection. On the other hand, it is not simple to generate useful term associations either. However, we cannot make the problem go away by just ignoring it. A related question is how effective is stemming and text normalization. With the rapid advance of IR system and computer power, some researchers begin to study ways to generate term correlations and the impact of stemming on text retrieval effectiveness. Church [8] studied the correlations between a term and its variants using statistical tools like Bahadur and Lazarsfeld (BL) expansion. He found that the correlation between a word and its variants tend to be small, but non-negligible. The correlation varies systematically depending on the words and the document collections involved. For examples, Church [8] found that $\langle "hostage", "hostages" \rangle = 0.5$ and $\langle "anytime", ".Anytime" \rangle = 0$ in a collection of 1988 AP articles.

Upperbound search algorithm on an inverted file

An inverted file usually is organized as a sorted list or a B-tree. The most straightforward way to search an inverted file is the brute force algorithm. All postings lists related to the query have to be retrieved and processed. This works well for small or medium sized document collections. When the document collection contains too many documents, it is necessary to find ways to avoid exhaustive search of the postings lists. One of the techniques is the so-called upperbound search algorithm developed by Smeaton, A. F. and C. J. van Rijsbergen [51], Mohan, K. C. and Willett, P. [36], Weiss, S. F [55] and Wong, W. Y. P [56].

Here we present a brief review of the upperbound search algorithm on the inverted file. Detailed information can be found in [36, 40, 51, 55, 56]. To make the presentation concise, we use $S(D)$ to denote document D 's score and $S(T)$ the T^{th} document's score in a ranking when there is no confusion.

The upperbound search algorithm is used to search for certain prespecified, say top T , documents in a document collection. Given a query $Q = \{t_1, t_2, \dots, t_k\}$ of k terms, Q is first sorted by increasing length of the postings lists. In this way, terms with high *idf* values will be processed first, while long postings lists are processed at the end. The upperbound search algorithm works as follows. After a postings list is processed, partial document scores are sorted in descending order. The current top T documents are obtained. The upperbound of the $(T + 1)^{th}$ document is then computed with the assumption that it contains all of the remaining unsearched query terms. The search stops if the upperbound of the $(T + 1)^{th}$ document is smaller than the current score of the T^{th} document. If the search can be terminated early, we have

better response time since short postings lists are processed first and the unprocessed lists are long. The upperbound search algorithm can be described as:

Algorithm: Upperbound Search Algorithm in Inverted File

1. Given query Q , sort the query terms in ascending order according to the lengths of their postings lists.
2. **For** each query term q in Q in the sorted list **Do**
3. **Begin**
4. **For** each document D in q 's postings list **Do**
5. $S(D) = S(D) + tf(q) * idf(q)$
6. Sort the document scores in descending order of their current scores.
7. **If** $U(T + 1) < S(T)$ **Then**
8. Stop the search process and return the top documents to the user.
9. **End**

In the upperbound search algorithm listed above, document scores are calculated according to the $tf * idf$ term weighting (Line 5). It is clear that the algorithm is applicable to other similarity measures.

Some variations of the upperbound search algorithm have been proposed to improve retrieval efficiency by relaxing the stopping criterion. Buckley and Lewit [5] suggested that upperbound of the $(T + 1)^{th}$ document is compared with current score of the S^{th} document, where $S < T$. With this relaxation, the chance to meet the stopping criterion is enhanced because of a larger difference in scores. The drawback is that it only returns a subset of the top documents. Another approach [Perry, Willett 1983; Weiss 1981] applies probability on the upper bound computation. The process

may stop early if the probability for the upperbound of the $(T + 1)^{th}$ document to exceed the current score of the T^{th} document is small.

Search Space Reduction in Signature Files

Signature files, in particular, those using the superimposed coding method in information retrieval have been studied for a long time. It still receives great attention in many application domains. It is well known that the signature file approach suffers from slow response time compared to the inverted file method. Some recent research on signature files has addressed the improvement of search speed, as exemplified by the partitioned signature file approach. Lee and Leng [30, 31] and Zezula et al. [63, 41].

To reduce the number of signatures searched, Lee and Leng [30, 31] proposed to partition signatures into a number of partitions by a deterministic mapping algorithm. Three partitioning algorithms, namely, the fixed-prefix, extended-prefix, and floating-key methods, were presented. Each method specifies a different way to extract keys from signatures.

In all three methods, signatures with the same key are grouped together to form a partition. For a partition M , the common key shared by all signatures in M is called the key of partition M , which will be referred to as K_M . To illustrate the main idea, let K_S be the key portion for signature S and $\bar{S}$ be the non-key portion. We can view the partition M which contains S as

$$M = \{\bar{S} | K_S = K_M\}.$$

When a query signature S_Q is received, we first extract its key K_{S_Q} in the same way as we did for signature S , then compare the key to the keys of the partitions. A

partition M is activated when K_M includes K_{S_Q} :

$$K_{S_Q} \wedge K_M = K_{S_Q}$$

An activated partition is searched further for signatures S' with $\tilde{S}'$ including $\tilde{S}_Q$, where $\tilde{S}_Q$ denote the non-key part of S_Q . As indicated by Lee and Leng's research, these partitioning methods reduce the search space by 15 to 85 percents over the sequential signature file approach with very little overhead. This is because many partitions are filtered out by the key comparison. These methods can be used effectively in a sequential environment as well as in a parallel environment.

The first partition method proposed by Lee and Leng is called fixed-prefix partitioning. The key is selected by taking the first k bits of a signature as the key, where k is the prespecified key length. Signatures with the same key are grouped together to form a partition.

The second partition method is called extended-prefix partitioning. In extended-prefix partitioning, the key of a signature is chosen to be the shortest prefix which contains a prespecified number of zeros, say z . The main reason for doing so is for load balancing since it has been shown that the probability of a partition being searched for a query solely depends on the number of zeros in the partition key [30]. The drawback is that different partitions may have different key lengths.

The last partition method, floating-key method, selects keys differently. First, a key length, say k , is selected, and then the first k -substring of a signature which contains the least number of ones is selected as the key. The key in this method is composed of the key value K_S^V , the substring which forms the key, and the key position K_S^P , the starting bit position of the key in the signature. For simplicity, K_S^P was assumed only to take the values of $0, k, 2k, \dots$, etc.

Lee and Leng [30, 31] have shown that floating-key partitioning results in more search space reduction than the fixed-prefix partitioning and extended-prefix partitioning.

In[31], Lee and Leng proposed a general trie data structure to support these three partition schemes. The structure uses fixed size pages. Signatures in each trie are partitioned into the left or right branch according to the value of the current bit position. Linked lists are used when all key positions are exhausted.

The trie structure turns out to be a very efficient way to organize signature files. The time complexity for searching the trie structure is of the order of the number of partitions which actually match the query. The complexity for both insertion and deletion operations are $O(k + p)$ [31], where p is the page size and k the key length.

ZeZula et al. [41] proposed another partitioning scheme, named *QuickFilter*. The basic idea is the same as Lee and Leng's approach, namely, to put similar signatures in the same page. However, ZeZula et al. used linear hashing to partition signatures. Linear hashing allows a hash file to expand and shrink dynamically without using a directory. For each signature $S = \langle b_1, b_2, \dots, b_m \rangle$, the hashing function used by ZeZula et al. is defined as:

$$g(S, h, n) = \begin{cases} 0 & \text{if } h = 0, n = 1; \\ \sum_{r=0}^{h-1} b_{m-r} 2^r & \text{if } \sum_{r=0}^{h-1} b_{m-r} 2^r < n; \\ \sum_{r=0}^{h-2} b_{m-r} 2^r & \text{otherwise.} \end{cases}$$

where h is the hashing level and $2^{h-1} < n \leq 2^h$. A page pointer p is maintained to point to the page where new signature is inserted. Whenever there is an overflow, the page pointed at by a split pointer is split and one new page is added. From the definition of this hash function, we can see that the hash function uses a suffix of the signature to decide where to insert signatures. When a query signature S_Q arrives,

$p = g(S_Q, h, n)$ is calculated to access the page. The search reduction ratio achieved is between 15-80% depending on the level of hashing.

The performance of key-partitioned signature file organization [30, 31] and Quick Filter [41] suffers when the query signature has a low weight, i.e., only a small percentage of bits are set to 1. To solve this problem, Yoo et al. [23, 60] recently introduced a new dynamic signature file structure, the Hierarchical Signature (HS) file. The HS file works well in dynamic environments (frequent insertions, deletions and updates) and has significantly better retrieval efficiency. In simulation, they reported 180 ~ 360% performance gain over the Quick Filter in terms of disk block accesses with a document collection of 100,000 documents.

The HS file is a height balanced tree that is a hierarchy of nodes containing signatures. The leaf nodes contain document signatures and pointers to the real documents. Non-leaf nodes contain signature and pointer pairs, where the pointer points to a sub-branch and signature is constructed by superimposing all signatures in the child node. The signatures in the non-leaf node behave as filters to exclude branches from search process. However, HS file does not address document ranking at all.

CHAPTER 3

Weight-partitioned Signature Files

In this chapter, we introduce the weight-partitioned signature file organization. Section 3.1 defines the term weights and the similarity formula under the vector-space model. Section 3.2 discusses how to encode exact term frequency in weight-partitioned signature file and the design of the signature file for supporting document ranking. Section 3.3 presents an analytical study. The experimental results on the performance of the weight-partitioned signature file under different search strategies and storage configurations are discussed in section 3.4.

3.1 Term Weights and Similarity Measure

Term weights are determined by the $tf \times idf$ strategy [Salton and Buckley 1988]. Precisely, the weight of term j in document i is defined as:

$$w_{i,j} = tf_{i,j} \times idf_j,$$

where $tf_{i,j}$ is the frequency of occurrence of term j in document i and idf_j is the inverse document frequency of term j in the entire collection. The similarity between a document and a query is based on the cosine measure:

$$\Theta(D_i, Q) = \frac{\sum_{j=1}^V w_{Q,j} \times w_{i,j}}{\sqrt{\sum_{j=1}^V w_{Q,j}^2 \sum_{j=1}^V w_{i,j}^2}}, \quad (3.1)$$

C	Total storage constraint.
D	A document.
D_i	i^{th} document in the database.
d	Number of terms in document D .
d_i	Number of terms in document D_i .
I_i	Average number of terms in a document with term frequency i .
$I_i(D)$	Number of terms in D with term frequency i .
idf_j	Inverse document frequency of term j in the database.
l_i	Total number of signatures in SF_i .
$l_i(D)$	Number of signatures in SF_i for document D .
m	Signature length.
m_i	Signature length in SF_i .
$\vec{m}$	Abbreviation for $m_1, \dots, m_n$.
N	Total number of documents.
n	Number of tf groups.
$p(m, w, s)$	False drop probability given m , w and s .
p_i	False drop probability of signatures in SF_i .
Q	Query.
q_i	False drop probability of file SF_i .
s	Number of words hashed into a signature.
s_i	Number of words hashed into a signature in SF_i .
$\vec{s}$	Abbreviation for $s_1, \dots, s_n$.
$S(D)$	Score associated with document D .
SF_i	Signature file corresponding to term frequency i .
$S(T)$	Score of T^{th} document.
$\Theta(D_i, Q)$	Similarity between document D_i and query Q .
T_{es}	Number of documents to do exact matching.
$tf_{i,j}$	Term frequency of term j in document i .
uf	Update frequency, used in search reduction algorithms.
$U(D)$	Upperbound of $S(D)$.
V	Size of the document vocabulary.
w	Weight of a word signature (number of bits set to 1).
w_i	Weight of a word signature in tf_i (number of bits set to 1).
$\vec{w}$	Abbreviation for $w_1, \dots, w_n$.
$w_{i,j}$	Weight of term j in document i .
$w_{Q,j}$	Weight of term j in query Q .
W	General term ranking weight.

Table 3.1: Symbols for TF-partitioned Signature Files

where V is the vocabulary size, $w_{Q,j}$ is the weight of term j in the query, defined in the same way as document term weights.

It is clear from equation (3.1) that the factor $\sum_{j=1}^V w_{Q,j}^2$ in the denominator is a constant for each document, so it only affects the absolute scores but not the ranks of the documents. Therefore, this factor is not used in our study. The factor $\sum_{j=1}^V w_{i,j}^2$ is very expensive to compute, because the weight of every term in a document has to be computed, not just for the terms specified in the query. Note that this factor cannot be pre-computed for each document because the weight depends on *idf* which changes as documents are inserted into and deleted from a collection. Therefore, we approximate the normalization factor with the square root of the number of terms in a document (i.e., the term weights are set to one when the normalization factor is computed). The effect of the approximation has been shown to be negligible compared to the exact normalization factor [Lee and Chuang 1994]. In summary, the similarity is computed as:

$$\Theta(D_i, Q) = \frac{\sum_{j=1}^V w_{Q,j} \times w_{i,j}}{\sqrt{d_i}}, \quad (3.2)$$

where d_i is the number of terms in document D_i .

3.2 Weight-partitioned Signature File

3.2.1 Encoding Term Frequency in Signature File

One major problem of signature files is the difficulty in encoding term frequency information. Croft and Savino [1988] evaluated several approaches which range from not representing term frequencies at all to approximating term frequencies with the number of text blocks containing the term. None of the approaches allow exact representation of term frequencies. Wong and Lee [1990] proposed a method to represent

the exact values of term frequencies by using a separate signature file for terms with the same term frequency. It is done by first grouping terms with the same term frequency in a document. Then, signatures are generated from each group and stored in the signature file corresponding to that term frequency. In our research, we follow a similar approach. The main difference is that our approach uses superimposed coding whereas Wong and Lee uses an encoding scheme which, unlike superimposed codes, does not introduce any false drops.

Figure 3.1 is an example of the signature generation process and Figure 3.2 illustrates the general design of the retrieval system. A document first goes through a preprocessing stage where common words are removed and keywords stemmed. Then, the keywords of a document are grouped according to term frequency. Signatures are generated from keywords in each group as in a traditional signature file, but signatures generated from different groups are stored in different signature files. The group corresponding to term frequency i is denoted by tf_i and the corresponding signature file is denoted by SF_i . For this reason, we also refer weight-partitioned signature files as tf -partitioned signature files from time to time. These two names will be used interchangeably throughout this thesis. During a search, the query signature is compared to each of the SF_i s. If a match is found in SF_i , the term frequency of the term producing the match has a possible term frequency equal to i , which can be used in computing the document score.

The algorithms for generating and searching the signature files are given in Algorithms 1 and 2, where n and N denote, respectively, the number of tf -groups and the number of documents in the collection. They are shown at the conceptual level

Figure 3.1: An example of weight-partitioned signature files

for the purpose of illustration only, and are not necessarily implemented as described in real implementations.

Algorithm 1. signature file generation

1. sort keywords and tabulate term frequencies of keywords:
2. for $i = 1$ to n do
3. for every s keywords in tf -group i do
4. generate signature and insert into SF_i ;
5. end {for}
6. end {for}

Algorithm 2: signature file search

1. for all term j in Q do
2. obtain idf of term j ;
3. generate signature for term j ;
4. end {for}
5. for $i = 1$ to N do
6. for $j = 1$ to n do

Figure 3.2: Retrieval system based on weight-partitioned signature files

7. for all term k in Q do
8. if term k 's signature matches D_i 's signature in the j th partition then
9. increment D_i 's score by i by $j \times idf_k \times w_{Q,k}$
10. break;
11. end {if}
12. end {for}
13. end {for}
14. end {for}

In the weight-partitioned signature file, multiple occurrences of the same keyword in a document will generate only one word signature and thus are represented in the signature file only once. This is different from the traditional method where a

Figure 3.3: Effects of search strategies

keyword will be represented as many times as the number of text blocks containing it (although multiple occurrences of a keyword within the same text block will be collapsed). This collapsing of multiple occurrences into a single representation in the signature file significantly reduces the storage overhead for long documents; on the other hand, the distribution of a document's signatures into several signature files causes additional overhead. Furthermore, this method does not always search all of the signatures in the signature file: whenever a match is found, the search can proceed to the next document since a query signature will find *at most* one match (ignoring false drops) within a document. This explains the break statement in step 10 of Algorithm 2.

Since the signature file is partitioned, the partitions can be searched in different orders and may have different signature lengths. The effects of different choices are explained in the following two subsections.

3.2.2 Search Strategies

A search can start from the signature file with highest term frequency and move towards the signature file with the lowest term frequency, or vice versa. We refer to

these two approaches, respectively, as HL (high to low) and LH (low to high) methods. There are two contradicting factors affecting the merits of these two search orders, as explained in Figure 3.3. Suppose signatures x and z are false drops matching the query term being searched. If the search starts from the highest term frequency, z will be identified as a match. In this case, the effect of the false drop on the document score will be very high since z has a high term frequency. On the contrary, if the search starts from the lowest term frequency, the false drop x will have a small effect on the document score since the term frequency is low. This suggests that LH is better than HL. However, there is a contradicting factor due to the fact that the number of signatures at high term frequencies is much less than that at low term frequencies, owing to the Zipfian distribution of terms among term frequencies. If we further assume that the probability that a true match occurs at SF_i is the same for all i , then, on the average, the chance of encountering a false drop before a true match is found is smaller in the HL approach than in the LH approach. In Fig. 3.3, if y is a true match, then there will be much fewer signatures to the “right” than to the “left” of y . Consequently, the probability that the search will arrive at y without hitting any false drops is higher if the search starts from the “right” (i.e., high term frequency). Further, the search speed is improved since once a match is found, the search can proceed immediately to the next query term. This factor is magnified if the user has a tendency to specify query terms which have high term frequencies (i.e., true matches tend to skew towards high term frequencies). This suggests that the search should start from the highest term frequency. The performance of these two search strategies will be studied in Section 3.3.

3.2.3 Signature File Configuration

When the signatures are partitioned among several files, it is possible to use different configurations for different partitions so that the false drop probabilities of the partitions could be different. Since a false match at high term frequencies affects the document scores more than those at low term frequencies, it is desirable to have lower false drop probabilities for high-frequency partitions and higher for low-frequency partitions. However, while the former decreases the chance of getting false drops at high-frequency partitions, the latter increases the number of false drops for the entire signature file since there are more signatures at the lower frequency end. Therefore, there are conflicting factors affecting the determination of the signature lengths for the partitions. For a given storage overhead (i.e., a fixed total size for all the signature partitions), we want to obtain an optimal way of allocating the storage to the partitions so as to minimize the effect of false drops on precision and recall. This is investigated in Section 3.3.

3.3 Analytical Results

In superimposed coding, there are four major parameters related by the following formula [Roberts 1979]:

$$p = (1 - (1 - w/m)^s)^w.$$

where: p : false drop probability.
 w : weight of a word signature (number of bits set by a word).
 m : signature length.
 s : number of words hashed into a block signature.

It has been shown that in order to minimize false drop probability for a given storage overhead, the number of bits set in a block signature must be equal to half of the

signature length, and

$$w = (1/\ln 2)^2 s (1 - 2^{-\frac{1}{s}}) \ln(1/p) \quad (3.3)$$

$$m = (1/\ln 2)^2 s \ln(1/p). \quad (3.4)$$

Formula (3.3) can be approximated by:

$$w = (1/\ln 2) \log_e(1/p) \quad (3.5)$$

when $s \geq 5$. These formulas are used later sections to determine the values of m , s and w when two of these parameters are given.

To improve clarity, we sometimes use $p(m, w, s)$ instead of p to indicate the relationship between p, w, m and s . With this notation, the false drop probability for signatures in partition SF_i is given by $p(m_i, w_i, s_i)$, or p_i .

In the experiment, p is a variable which is varied for each experiment run. The signature length, m , is a design parameter which is fixed for each test collection. Choosing a length which fits into the native word length of the machine is most efficient, but the native word length is short and thus will limit the achievable minimum false drop probability. Further, a low false drop probability with a short signature length entails an extremely small number of words to be hashed in a signature, resulting in a large number of signatures and disk accesses (if buffering is not used). On the other hand, a long signature length is less efficient to manipulate and will generate under-utilized signatures for short documents.

3.3.1 HL Versus LH Search Strategies

In this subsection, we analyze the performance of the HL and LH search strategies. The performance of these two strategies depend on the distributions of false and true

drops. The distribution of false drops further depends on the signature lengths of the partitions. In order to compare these two methods, we assume in this subsection that the distributions and the signature lengths are uniform.

In the search algorithm, a signature file SF_i is treated as a single unit. A match with any signature in SF_i results in tf equal to i . We need to find the probability of false match when we compare a random signature with all of the signatures in SF_i . We denote the false drop probability against SF_i as q_i to distinguish it from the usual false drop probability, which is the probability for two signatures to match combinatorially.

Lemma 1 *The false drop probability against tf_i is given by:*

$$q_i = 1 - (1 - p(m_i, w_i, s_i))^{l_i} \approx l_i p(m_i, w_i, s_i).$$

where l_i is the number of signatures in SF_i .

Proof: This is clear. $\square$

We may use different m_i , w_i and s_i for different tf -groups. As long as

$$1 \geq p_1 \geq \dots \geq p_n \geq 0,$$

we have

$$0 \leq 1 - p_1 \leq \dots \leq 1 - p_n \leq 1.$$

From Zipf's Law, we know $l_1 \geq l_2 \geq \dots \geq l_{n-1}$ and $l_1 > l_n$. This implies

$$(1 - p_1)^{l_1} \leq \dots \leq (1 - p_{n-1})^{l_{n-1}} \quad \text{and} \quad (1 - p_1)^{l_1} \leq (1 - p_n)^{l_n},$$

or, equivalently, $q_1 \geq \dots \geq q_{n-1}$ and $q_1 \geq q_n$. These relations among q_i 's imply that false drop probabilities are higher at the low tf end than at the high tf end. This

turns out to be very important in our analysis. One thing we want to point out is that $p_1 \geq \dots \geq p_n$ does not necessarily mean $m_1 \geq \dots \geq m_n$ (neither does the other direction hold) since there are other parameters such as s_i 's and w_i 's involved.

Lemma 2 *Given a random word signature, the expected tf at which a match is found can be obtained by $S_n^{(n)}$ for HL and $T_1^{(n)}$ for LH, where the definitions of $S_i^{(n)}$ and $T_i^{(n)}$ are given as follows:*

(1) For HL:

$$\begin{aligned} S_n^{(i)} &= q_i i + (1 - q_i) S_{i-1}^{(n)} \quad \text{for } i = n, \dots, 2 \\ S_1^{(n)} &= q_1. \end{aligned}$$

(2) For LH:

$$\begin{aligned} T_i^{(n)} &= i q_1 + (1 - q_i) T_{i+1}^{(n)} \quad \text{for } i = 1, \dots, n-1 \\ T_n^{(n)} &= n q_n. \end{aligned}$$

Further, $S_n^{(n)} \geq T_1^{(n)}$.

Proof: In the HL method, the probability that there is a match in tf_n is q_n . If there is a match in tf_n , the expected tf value is n . Otherwise, we have to check the next lower tf group. Therefore, we get the following recurrence relation:

$$S_n^{(n)} = q_n n + (1 - q_n) S_{n-1}^{(n)}.$$

For tf_1 , there is no lower tf group, the expected tf value is given by q_1 . $T_n^{(n)}$ for the LH method can be derived in a similar way. That $S_n^{(n)} \geq T_1^{(n)}$ is clear from the definitions of $S_n^{(n)}$ and $T_1^{(n)}$. $\square$

Lemma 3 Suppose a word appears in a document with $tf = j$, then HL yields the following expected tf :

$$j + ((n - j)q_n + (n - j - 1)q_{n-1} + \cdots + q_{j+1})$$

and LH yields:

$$j - ((j - 1)q_1 + (j - 2)q_2 + \cdots + q_{j-1}).$$

when high order terms such as $q_i q_j$ and $q_i q_j q_k$ are dropped.

Proof: We will use the formulas in Lemma 2. The key in this Lemma is that search will stop at tf_k , where $k \geq j$ for HL and $k \leq j$ for LH.

(1) For HL:

$$\begin{aligned} S_n^{(i)} &= q_i i + (1 - q_i) S_{i-1}^n \quad \text{for } i = n, \dots, 2 \\ S_1^{(n)} &= q_1. \end{aligned}$$

By eliminating $S_i^{(n)}$ for $i = n - 1, \dots, j$ from the above equations, we have

$$\begin{aligned} S_n^{(n)} &= q_n n + (n - 1)q_{n-1}(1 - q_n) + \cdots + (j + 1)q_{j+1}(1 - q_n) \cdots (1 - q_{j+2}) \\ &\quad + j(1 - q_n)(1 - q_{n-1}) \cdots (1 - q_{j+1}). \end{aligned}$$

Multiplying out everything and dropping small high order terms, we get

$$\begin{aligned} S_n^{(n)} &\approx nq_n + \cdots + (j + 1)q_{j+1} + j - j(q_n + \cdots + q_{j+1}) \\ &= j + ((n - j)q_n + (n - j - 1)q_{n-1} + \cdots + q_{j+1}). \end{aligned}$$

(2) For LH: As in HL, we have

$$T_1^{(n)} \approx j - ((j - 1)q_1 + (j - 2)q_2 + \cdots + q_{j-1}).$$

Since false drops exist, it is clear that in the above two expressions $S_n^{(n)} \geq j$ is an over-estimation and that $T_1^{(n)} \leq j$ is an under-estimation. $\square$

Since we know the exact tf value is j , we can check the errors introduced by these two methods:

$$|S_n^{(n)} - j| \approx (n - j)q_n + (n - j - 1)q_{n-1} + \cdots + q_{j+1}.$$

$$|T_1^{(n)} - j| \approx (j - 1)q_1 + (j - 2)q_2 + \cdots + q_{j-1}.$$

It is easy to see that $|S_n^{(n)} - j| \leq |T_1^{(n)} - j|$ provided $n - j \leq j - 1$. Therefore, $S_n^{(n)}$ is a better approximation of the true tf value if $j \geq (n + 1)/2$.

The number $(n + 1)/2$ is a very conservative estimation. In fact, since q_1 (the false drop probability for SF_1) is typically much larger than q_n (the false drop probability for SF_n), the tf value above which the HL approach yields better results than the LH method is much smaller than $(n + 1)/2$ owing to the Zipfian distribution of terms among term frequencies. In the following discussion, $I_i(D)$ is used to denote the number of terms in D with term frequency i , and $l_i(D)$ the number of signatures in SF_i for document D . Given a document D and the parameters m_i , s_i , and w_i for each tf_i , we have the relation $l_i(D) \approx I_i(D)/s_i$. The Zipf Law can be mathematically approximated as [Salton 1989]:

$$I_i(D) = \frac{2}{i(i+1)} I_1(D) \quad \text{for } i \geq 1. \quad (3.6)$$

From formula (3.6), we can get

$$I_{\geq n}(D) \approx \frac{2}{n} I_1(D). \quad (3.7)$$

For simplicity, we will use I_n to denote $I_{\geq n}$.

Lemma 4 Suppose Zipf's Law holds for our document collection, $n = 30$ and

$$\frac{p_1}{s_1} \geq \dots \geq \frac{p_n}{s_n} \quad (3.8)$$

then for any term occurring in the document with $tf \geq 4$, $S_n^{(n)}$ is a better approximation of the tf value than $T_1^{(n)}$.

Proof: Without loss of generality, we assume that the average number of terms in tf_1 for our document collection is I_1 and that $tf = j$. Then using (3.6) and (3.7), we have

$$\begin{aligned} |S_n^{(n)} - j| &\approx (n-j)l_n p_n + (n-j-1)l_{n-1} p_{n-1} + \dots + l_{j+1} p_{j+1} \\ &\approx \frac{2(n-j)I_1}{n} \frac{p_n}{s_n} + \dots + \frac{2I_1}{(j+1)(j+2)} \frac{p_{j+1}}{s_{j+1}} \\ &\leq I_1 \frac{p_j}{s_j} \left(\frac{2(n-j)}{n} + \frac{2(n-j-1)}{(n-1)n} + \dots + \frac{2}{(j+1)(j+2)} \right) \\ |T_1^{(n)} - j| &\approx (j-1)q_1 + (j-2)q_2 + \dots + q_{j-1} \\ &\geq I_1 \frac{p_j}{s_j} \left((j-1) + \frac{2(j-2)}{2 \times 3} + \dots + \frac{2}{(j-1)j} \right) \end{aligned}$$

Note that $I_1 p_j / s_j$ is a common factor in both formulas. A simple computation will show that $|S_n^{(n)} - j| \leq |T_1^{(n)} - j|$ for any $j \geq 4$. $\square$

The condition (3.8) is not restrictive. Clearly, it is true when the signature lengths are uniform. We will show later that this condition also holds for our optimal signature configurations. From the proof, we can see that $tf \geq 4$ is still a conservative result. In our experiment with the TREC subset, HL is better than LH for $j \geq 2$ using the configuration $n = 30$, $m = 29$, $s = 2$ and $w = 10$. Lemma 3 can be used to find the exact j when the statistical data about the document collection is available.

Lemma 3 considers the situation where the query term indeed exists in the documents. We now consider the case when there is no such word in the document (i.e., the match is a false drop).

Lemma 5 *In the case of false drop, the expected tf value given by $S_n^{(n)}$ and $T_1^{(n)}$ all have the form*

$$\sum_{i=1}^n iq_i + \text{high order terms.}$$

Proof: Eliminating all $S_i^{(n)}$ for $i = n - 1, \dots, 1$ from part (1) of Lemma 2, we get

$$S_n^{(n)} = \sum_{i=1}^n iq_i + \text{high order terms.}$$

Similarly, we can obtain the expression for $T_1^{(n)}$. $\square$

Theorem 1 *If the false drop probabilities p_i 's and terms per signature s_i 's satisfy*

$$\frac{p_1}{s_1} \geq \dots \geq \frac{p_n}{s_n}.$$

the HL search strategy is superior than the LH method.

Proof: From Lemma 4, we can see that for true matches, HL produces *tf* values which are closer to the true values than LH does on the average.

On the other hand, Lemma 5 indicates that the expected errors created by these two methods on a false match all have the form: $\sum_{i=1}^n iq_i + \text{high order terms}$. The difference between the errors introduced by the two methods is negligible considering that the normal value of q_i is typically very small. Combining these two facts, we can claim that HL is better than LH. $\square$

We will see in the following subsection that in the optimal signature configuration, the signature lengths of the partitions are increasing and the false drop probabilities are decreasing from low to high term frequencies. Under this condition, it is clear that HL is even more desirable than LH.

3.3.2 Optimal Signature File Configuration

We have implicitly assumed so far that every signature partition uses the same signature length m , the same w and the same s . However, for the weight-partitioned signature file, different partitions may be assigned different signature lengths, different signature weights, etc. This implies that for a given total storage overhead, we can control the amount of storage allocated to each partition by changing the configuration of that partition. Thus, for a fixed total storage overhead, there is an optimal way of assigning signature lengths (or storage) to each partition to minimize the impact of false drops on precision and recall. This section derives the optimal signature configurations.

Since different SF_i 's may have different signature configurations and hence different false drop probabilities, the weighted average of the false drop probabilities of all SF_i 's is used to characterize the false drop probability of the whole signature file. The effective false drop probability can be written as

$$\begin{aligned} p(\vec{m}, \vec{w}, \vec{s}) &= \sum_{i=1}^n \frac{l_i(D)}{\sum_{j=1}^n l_j(D)} p(m_i, w_i, s_i) \\ &= \sum_{i=1}^n \frac{I_i(D)}{\sum_{j=1}^n I_j(D)} p(m_i, w_i, s_i). \end{aligned} \quad (3.9)$$

where $\vec{m}$ denotes $m_1, m_2, \dots, m_n$. In this formula, $l_i(D)$ and $I_i(D)$, for $i = 1, \dots, n$, are document dependent variables. We will use Zipf's law to eliminate them. Applying (3.6) and (3.7) to (3.9), we get a formula for the document independent effective

false drop probability:

$$p = \sum_{i=1}^{n-1} \frac{1}{i(i+1)} p(m_i, w_i, s_i) + \frac{1}{n} p(m_n, w_n, s_n). \quad (3.10)$$

When $m_1 = \dots = m_n$, $s_1 = \dots = s_n$ and $w_1 = \dots = w_n$ (i.e., all SF_i 's use the same signature configuration), we can easily check that

$$p(\vec{m}, \vec{w}, \vec{s}) = p(m_1, w_1, s_1) = \dots = p(m_n, w_n, s_n).$$

In other words, the uniform signature structure is just a special case of the non-uniform signature structure. We now present a result about the relation between the uniform signature length and the non-uniform signature length approaches.

Lemma 6 *For a signature file with parameters m, s , and w , reducing s by a factor of λ has the same effect on false drop probability as increasing m by a factor of λ , where $\lambda \geq 1$. Furthermore, increasing or decreasing m and s by the same factor does not change the false drop probability.*

Proof: We will use the false drop probability formula:

$$p(m, w, s) = (1 - (1 - w/m)^s)^k.$$

In fact, for small w and s (relative to m), we have

$$\begin{aligned} (1 - w/m)^s &= \left[\left(1 - \frac{1}{\frac{m}{w}}\right)^{\frac{m}{w}} \right]^{\frac{ws}{m}} \\ &\approx e^{-\frac{ws}{m}}. \end{aligned}$$

Thus, we have $p(m, w, s) \approx (1 - e^{-\frac{ws}{m}})^w$. Now increasing m by a factor of λ , we get

$$p(\lambda m, w, s) \approx (1 - e^{-\frac{ws}{\lambda m}})^w.$$

Reducing s by a factor of λ , we get

$$p(m, w, s/\lambda) \approx (1 - e^{-\frac{w(s/\lambda)}{m}})^s = (1 - e^{-\frac{ws}{\lambda m}})^w.$$

We can reach the second conclusion in a similar way. $\square$

To obtain formulas to determine the optimal signature configurations for the partitions, we first obtain a formula for the general case, where each partition is associated with a general ranking weight W .¹ Then, we can apply the formula to the case where tf is used as the ranking weight (we called this the TF-partitioned signature file). The theorem presented below determines the optimal signature configurations for the signature partitions to minimize the effect of false drops on retrieval effectiveness. The condition under which the theorem applies is similar to that for the false drop probability formula [Roberts 1979] (i.e., when w_i , s_i , and l_i are small relative to m_i , which is true for most applications).

It is well known that for the signature file approach, the space needed is proportional to $\ln(1/p)$. Given the space constraint, we will first compute the optimal false drop probabilities for all SF_i 's. From the false drop probabilities, we can compute the other parameters such as signature lengths, bits per word, etc.

We assume that document terms are partitioned into different groups, say, G_1 , G_2 , $\dots$, G_n and each term in G_i carries a weight W_i . Without loss of generality, we can assume that $W_1 \leq \dots \leq W_n$. The corresponding signature files are denoted as SF_1 , SF_2 , $\dots$, SF_n as before. Under these assumptions, the storage overhead for a document is $\sum_{i=1}^n l_i m_i$, where l_i is the number of signatures in SF_i and m_i is the length of signatures in SF_i .

¹In order to distinguish this weight, which is used to determine the importance of a match in the vector space model, from the signature weight, which is the number of ones in a signature, the former is hereafter called *ranking weight*.

Lemma 7 *For false matches, the expected weight given by $S_n^{(n)}$ and $T_1^{(n)}$ all have the form*

$$\sum_{i=1}^n W_i q_i + \text{high order terms.}$$

Proof: Similar to Lemma 5. $\square$

Lemma 8 *Suppose the storage constraint is $\sum_{i=1}^n l_i m_i = C$. In order to minimize the effect of false drops, the relations between different false drop probabilities p_i are:*

$$\begin{aligned} \frac{W_i}{s_i} p(m_i, w_i, s_i) &= \frac{W_j}{s_j} p(m_j, w_j, s_j) \quad \text{for } 1 \leq i \leq j \leq n \\ p(m_n, w_n, s_n) &= \exp \left(- \frac{C(\ln 2)^2 + \sum_{k=1}^n s_k l_k \ln \left(\frac{W_n s_k}{W_k s_n} \right)}{\sum_{k=1}^n l_k s_k} \right) \end{aligned}$$

Proof: We know from the previous result that the expected tf value is

$$\sum_{i=1}^n W_i q_i$$

for a false drop. We will minimize the function $\sum_{i=1}^n W_i q_i$ under the storage constraint $\sum_{i=1}^n l_i m_i = C$.

The existence of a minimum value is clear since the constraint defines a compact set in space R^n and the function is continuous. It is also not difficult to see that the minimum value cannot be achieved on boundaries. First, from Lemma 1, we have

$$\sum_{i=1}^n W_i q_i \approx \sum_{i=1}^n W_i l_i p_i.$$

There are too many variables to handle. We first eliminate all m_i 's. From formula (3.4), we have

$$m_i = (1/\ln 2)^2 s_i \log_e(1/p_i).$$

The constraint can be rewritten as

$$C = \sum_{i=1}^n l_i m_i = \sum_{i=1}^n (1/\ln 2)^2 s_i l_i \log_e(1/p_i).$$

The Lagrange's method can be used for obtaining the minimum. Let

$$F(\lambda, \vec{m}, \vec{w}, \vec{s}) = \lambda \left(\sum_{i=1}^n (1/\ln 2)^2 s_i l_i \log_e(1/p_i) - C \right) + \sum_{i=1}^n W_i l_i p_i.$$

The partial derivatives with respect to p_i 's and λ are

$$\begin{aligned} \frac{\partial F}{\partial p_i} &= \lambda \frac{l_i s_i}{p_i} + W_i l_i \quad i = 1, \dots, n. \\ \frac{\partial F}{\partial \lambda} &= \sum_{i=1}^n (1/\ln 2)^2 s_i l_i \log_e(1/p_i) - C. \end{aligned}$$

Now set all the partial derivatives to zero and cancel l_i from the i^{th} equation for $i = 1, \dots, n$, we have

$$\frac{W_i}{s_i} p_i + \lambda = 0 \quad i = 1, \dots, n. \quad (3.11)$$

$$\sum_{i=1}^n (1/\ln 2)^2 s_i l_i \log_e(p_i) + C = 0 \quad (3.12)$$

Eliminating λ from the i^{th} and j^{th} equations, we get

$$\frac{W_i}{s_i} p(m_i, w_i, s_i) = \frac{W_j}{s_j} p(m_j, w_j, s_j)$$

for $1 \leq i \leq j \leq n$. Setting $j = n$, we get

$$p_i = \frac{W_n s_i}{W_i s_n} p_n.$$

Substituting these equations into equation (3.12), we have

$$\sum_{i=1}^n s_i l_i \ln \frac{W_n s_i}{W_i s_n} + \left(\sum_{i=1}^n s_i l_i \right) \ln p_n + (\ln 2)^2 C = 0.$$

Solving for p_n , we get

$$p_n = \exp \left(- \frac{C(\ln 2)^2 + \sum_{k=1}^n s_k l_k \ln \left(\frac{W_n s_k}{W_k s_n} \right)}{\sum_{k=1}^n l_k s_k} \right).$$

□.

There are several things worth pointing out. First, since $W_1 \leq \dots \leq W_n$, we have

$$\frac{p_1}{s_1} \geq \dots \geq \frac{p_n}{s_n}.$$

These relations will ensure that HL is better than LH from Lemma 4. Secondly, it is a little surprising that the signature file size l_i disappeared in

$$\frac{W_i}{s_i} p(m_i, w_i, s_i) = \frac{W_j}{s_j} p(m_j, w_j, s_j).$$

This can be explained intuitively by the fact that there are two conflicting factors: large l_1 means more false drops, thus requiring more storage to be assigned to tf_1 ; on the other hand, with a fixed space, we cannot assign too much storage to tf_1 . The optimal method we are presenting will give us an optimal way of allocating storage among all tf -groups.

Theorem 2 *Under the same assumption as in the previous Lemma, to minimize the effect of false drops, the signature length m_i , bits per word w_i and words per signature s_i must satisfy:*

$$m_i = \frac{1}{\ln^2 2} s_i \left(\ln \left(\frac{W_i s_n}{W_n s_i} \right) + \frac{C(\ln 2)^2 + \sum_{k=1}^n s_k l_k \ln \left(\frac{W_n s_k}{W_k s_n} \right)}{\sum_{k=1}^n l_k s_k} \right) \quad (3.13)$$

$$w_i = \frac{1}{\ln 2} s_i (1 - 2^{-\frac{1}{s_i}}) \left(\ln \left(\frac{W_i s_n}{W_n s_i} \right) + \frac{C(\ln 2)^2 + \sum_{k=1}^n s_k l_k \ln \left(\frac{W_n s_k}{W_k s_n} \right)}{\sum_{k=1}^n l_k s_k} \right), \quad (3.14)$$

where $i = 1, \dots, n$.

Proof: From Lemma 8, we know for $1 \leq i \leq j \leq n$:

$$\begin{aligned}\frac{W_i}{s_i} p_i &= \frac{W_j}{s_j} p_j \\ p_n &= \exp \left(-\frac{C(\ln 2)^2 + \sum_{k=1}^n s_k l_k \ln \left(\frac{W_n s_k}{W_k s_n} \right)}{\sum_{k=1}^n l_k s_k} \right).\end{aligned}$$

If we let $j = n$, we get $p_i = \frac{W_n s_i}{W_i s_n} p_n$. This implies that

$$p_i = \frac{W_n s_i}{W_i s_n} \exp \left(-\frac{C(\ln 2)^2 + \sum_{k=1}^n s_k l_k \ln \left(\frac{W_n s_k}{W_k s_n} \right)}{\sum_{k=1}^n l_k s_k} \right).$$

From formulas (3.3), (3.4) and (3.15), we arrive at

$$\begin{aligned}m_i &= \frac{1}{\ln^2 2} s_i \ln \frac{1}{p_i} \\ &= \frac{1}{\ln^2 2} s_i \left(\ln \left(\frac{W_i s_n}{W_n s_i} \right) + \frac{C(\ln 2)^2 + \sum_{k=1}^n s_k l_k \ln \left(\frac{W_n s_k}{W_k s_n} \right)}{\sum_{k=1}^n l_k s_k} \right) \\ w_i &= \frac{1}{\ln 2} s_i (1 - 2^{-\frac{1}{s_i}}) \ln \frac{1}{p_i} \\ &= \frac{1}{\ln 2} s_i (1 - 2^{-\frac{1}{s_i}}) \left(\ln \left(\frac{W_i s_n}{W_n s_i} \right) + \frac{C(\ln 2)^2 + \sum_{k=1}^n s_k l_k \ln \left(\frac{W_n s_k}{W_k s_n} \right)}{\sum_{k=1}^n l_k s_k} \right)\end{aligned}$$

with some simple algebraic operations. $\square$

In order to use Theorem 2, we need to decide C , s_i , and l_i . Note that only C and s_i 's are design parameters, since $l_i = \lceil I_i/s_i \rceil$, which is determined by the document collection. C can be determined from the storage overhead desired. When C is fixed, the other parameters, namely, s_i 's, need to be determined. The selection of s_i 's needs some consideration. Suppose in a document collection, each document has on the average I_i terms with term frequency i . Then, a factor of I_1 should be used as s_1 to make sure we hash approximately the same number of words into a signature. For example, the TREC subset has on the average 105 terms in I_1 . Then, $s_1 = 5$ is a good choice since each signature in SF_1 will contain 5 terms. On the other hand, $s_1 = 8$ will

be a bad choice since one of the signatures will contain only one word. After we select $s_1 = s$, $s_i = \lceil \min(s, I_i) \rceil$ is the logical choice for SF_i , where $i = 2, \dots, n$. Finally, since we have $s_i l_i = \lceil I_i \rceil$, the optimal formulas (3.13) and (3.14) can be simplified as

$$m_i = \frac{1}{\ln^2 2} s_i \left(\ln\left(\frac{W_i s_n}{W_n s_i}\right) + \frac{C(\ln 2)^2 + \sum_{k=1}^n \lceil I_k \rceil \ln\left(\frac{W_n s_k}{W_k s_n}\right)}{\sum_{k=1}^n \lceil I_k \rceil} \right) \quad (3.15)$$

$$w_i = \frac{1}{\ln 2} s_i (1 - 2^{-\frac{1}{s_i}}) \left(\ln\left(\frac{W_i s_n}{W_n s_i}\right) + \frac{C(\ln 2)^2 + \sum_{k=1}^n \lceil I_k \rceil \ln\left(\frac{W_n s_k}{W_k s_n}\right)}{\sum_{k=1}^n \lceil I_k \rceil} \right). \quad (3.16)$$

Although it is natural to use different signature lengths in the weight-partitioned signature file method, it is difficult in practice (e.g., in terms of programming effort) to use different signature lengths for different SF_i 's because the signature lengths must be memorized for signature generation. A better alternative is to use the same signature length for all signature partitions but different s_i values. The next Corollary gives an optimal signature file configuration in which all signatures have the same length.

Corollary 1 *Suppose the storage constraint is C , then the following signature file configuration will minimize the effect of false drops.*

$$\begin{aligned} m &= \min\{\bar{m}_1, \dots, \bar{m}_n\} \\ s_i &= \left\lceil \frac{m}{\bar{m}_i} * \bar{s}_i \right\rceil \quad \text{for } i = 1, \dots, n \\ w_i &= \frac{1}{\ln 2} s_i (1 - 2^{-1/s_i}) \left(\ln\left(\frac{W_i s_n}{W_n s_i}\right) + \frac{C(\ln 2)^2 + \sum_{k=1}^n s_k l_k \ln\left(\frac{W_n s_k}{W_k s_n}\right)}{\sum_{k=1}^n l_k s_k} \right) \quad \text{for } i = 1, \dots, n \end{aligned}$$

where $l_i = \lceil I_i / s_i \rceil$ and w_i , $\bar{m}_i$ and $\bar{s}_i$ satisfy Theorem 2.

Proof: We know from Lemma 6 that increasing or decreasing m and s by the same factor simultaneously will not change the false drop probability. We first using Theorem 2 to compute all $\bar{m}_i$'s and $\bar{s}_i$'s. Then, we select the shortest signature length

(or whichever we prefer) as our new uniform signature length m and obtain s_i by scaling down $\bar{s}_i$ by the factor $m/\bar{m}_i$. The weight of each term, w_i , is the same as in Theorem 2. $\square$

We can see that Theorem 2 and Corollary 1 both depend on $I_1, \dots, I_n$, the sizes of the term frequency groups. These values can be determined by taking the averages of the term frequency group sizes over the document collection. Alternatively, we can use the Zipf's Law to get an approximation of $I_2, \dots, I_n$ from I_1 . From (3.6) and (3.7), we get

$$s_i = \lceil \min(s, I_i) \rceil = \left\lceil \min\left(s, \frac{I_1}{i(i+1)}\right) \right\rceil.$$

Then, we can use Theorem 2 to determine m_i 's and w_i 's and generate the signature files.

The general result can now be applied to the TF-partitioned signature file using the term frequencies as the ranking weights. That is, $W_i = i$ in eqns. 3.13–3.16. This amounts to assigning weights to a match in proportion to the term frequencies. Since false drops occurring at high term frequencies will have a large negative impact on the retrieval effectiveness, we want to allocate more storage to high term frequency partitions by increasing their weights.

3.3.3 Effect of False Drops on Document Ranks

Another application of Lemma 3 is on false drop eliminations. False drop elimination is expensive to perform. When false drop elimination is not performed, we want to verify analytically that reducing the number of false drops beyond a certain threshold yields no gain in precision and recall. Furthermore, if p denotes the exact value of a parameter, p' denotes the approximate value of the parameter derived from

the weight-partitioned signature file (i.e., with false drops). For example, $tf'(t)$ denotes the tf value of term t obtained in the search process and $tf(t)$ for the true tf value of t in the document: $\Theta'(D, Q)$ denotes the similarity obtained from the weight-partitioned signature file and $\Theta(D, Q)$ to denote the actual similarity if no false drop exist. $\Theta'(D, Q)$ is also used to denote the expected value of $\Theta(D, Q)$ for simplicity. The expected $\Theta'(D, Q)$ is a theoretical representation of $\Theta'(D, Q)$ on the average.

Lemma 9 *Given a query Q , the difference between the true similarity and the expected similarity of a document D obtained using the weight-partitioned signature file is a linear combination of q_i 's.*

Proof: We divide the query terms into two groups $Q = Q_1 \cup Q_2$, where $Q_1 = D \cap Q$ and $Q_2 = Q \setminus D$. The true similarity between document D and query Q is given by

$$\Theta(D, Q) = \frac{\sum_{t \in Q_1} tf(t) \times idf(t)}{\sqrt{d}}.$$

and the expected similarity obtained by the weight-partitioned signature file method is:

$$\Theta'(D, Q) = \frac{\sum_{t \in Q_1} tf'(t) \times idf(t) + \sum_{t \in Q_2} tf'(t) \times idf(t)}{\sqrt{d}}.$$

Due to errors caused by false drops, the tf value obtained by the HL method may be larger than the true tf value, although the difference is typically small. The formula for the expected tf values of the terms in Q_1 is given by Lemma 3. That is,

$$tf'(t) - tf(t) \approx ((n - tf(t))q_n + (n - tf(t) - 1)q_{n-1} + \cdots + q_{tf(t)+1}).$$

And the expected tf-value for elements of Q_2 is given by

$$tf'(t) = \sum_{i=1}^n iq_i.$$

The right hand side of these two expectations are linear combinations of q_i 's for a fixed query Q and document D . From the above formulas, we have

$$\begin{aligned} & \Theta'(D, Q) - \Theta(D, Q) \\ &= \frac{\sum_{t \in Q_1} (tf'(t) - tf(t)) \times idf(t)}{\sqrt{d}} + \frac{\sum_{t \in Q_2} (tf'(t)) \times idf(t)}{\sqrt{d}} \\ &= \sum_{t \in Q_1} \frac{idf(t)}{\sqrt{d}} (tf'(t) - tf(t)) + \sum_{t \in Q_2} \frac{idf(t)}{\sqrt{d}} (tf'(t)), \end{aligned}$$

which is clearly a linear combination of q_i 's since $tf'(t) - tf(t)$ for $t \in Q_1$, and $tf'(t)$ for $t \in Q_2$ are for every term t in Q . $\square$

From Lemma 9, we can see that the expected $\Theta'(D, Q)$ approaches $\Theta(D, Q)$ as q_i 's approach zero. For query Q , the vector-space model assigns a score $\Theta(D, Q)$ to each document. For simplicity, let's assume that no two documents receive the same score. In this way, a complete ranking of all the documents in $\mathcal{D}$ is produced. Under this assumption, the following theorem can be obtained.

Theorem 3 *For a document collection $\mathcal{D}$ and a set of queries $\mathcal{Q}$, there is a constant ϵ such that when the false drop probabilities for all SF_i 's are smaller than ϵ , the expected precision and recall resulted from searching the weight-partitioned signature file for $Q \in \mathcal{Q}$ will be the same as that obtained by inverted file model. In other words, reducing the false drop probabilities beyond this ϵ yields no gain in precision and recall.*

Proof: For each query $Q \in \mathcal{Q}$, the vector space model assigns a unique score $\Theta(D, Q)$ to each document D . Sorting all documents by decreasing order of their scores, we

obtain a complete ranking. Let's use δ_Q to denote the minimum difference between two adjacent documents. In other words,

$$\delta_Q = \min\{\Theta(D_i, Q) - \Theta(D_{i+1}, Q) \mid i \geq 1\}.$$

Finally, let $\delta = \min\{\delta_Q \mid Q \in \mathcal{Q}\}$. Since no two documents receive the same score, δ is a positive number. From Lemma 9, for each document D and query Q , we have

$$\lim_{q_i \rightarrow 0} (\Theta'(D, Q) - \Theta(D, Q)) = 0.$$

Thus, there must be a positive number ϵ such that when all of the false drop probabilities are less than ϵ ,

$$0 \leq \Theta'(D, Q) - \Theta(D, Q) < \delta.$$

This means that the expected ranking order given by $\Theta'(D, Q)$ will be identical to the ranking order given by $\Theta(D, Q)$. Reducing false drop probabilities beyond this ϵ will not alter the precision and recall values. $\square$

It is not always the case that no two documents receive the same score, but we can modify Theorem 3 to imply there is a number ϵ such that if all false drop probabilities are less than ϵ .

$$\Theta(D_1, Q) > \Theta(D_2, Q) \implies \Theta'(D_1, Q) > \Theta'(D_2, Q).$$

Theorem 3 does not give any method to compute the threshold ϵ . The main difficulty is we don't have a prior collection of queries. Even with this knowledge, obtaining this number is not simple. Constant update of the collection makes it

even more complicated to predict. Sampling or experimentation may be a practical method to obtain this threshold ϵ .

3.4 Experimental Results

The criteria for retrieval effectiveness are precision and recall. The goal of the experiments is:

1. to observe the extent to which false drops can be tolerated without causing unacceptable degradation to recall and precision.
2. to confirm the analytical results on the effectiveness of the LH and HL search strategies.
3. to observe the performance of various signature length assignment methods.

Experiments were performed with the CACM collection and a document subset extracted from the TREC collection [Harman 1993a]. CACM is a small collection of 3,204 bibliographic records from the *Communication of ACM* with a raw size of about 1.6 Mbytes (not counting index and comment fields). This collection (together with CISI, TIMES, CRAN, and MED collections, all of which are available on SMART) has been used extensively in the literature for evaluating the effectiveness of retrieval methods. The full TREC collection supplied by NIST consists of about 2 Gbytes worth of articles. Since the full collection will exceed the resource of our computing environment, our experiments use a subset, hereafter called the TREC subset, of 10,000 Wall Street Journal articles in 1987 with a raw size of about 27 Mbytes.² The TREC subset, which is 15 times the size of CACM, will show if our results can scale

²The subset was extracted at Online Computer Library Center (OCLC), Dublin, Ohio.

up and provide an additional data point to study our results. In addition, both the documents and the queries in the TREC subset are more diversified in contents and have longer lengths than their counterparts in the CACM collection.

The CACM collection and the TREC subset have, respectively, 64 and 25 standard queries, each of which comes with a standard relevance judgment. The precision and recall values are obtained as follows. First, the standard queries are run on the collections. The ranked outputs of the queries are then compared to their respective list of relevant documents. Finally, the precision and recall values are calculated and averaged over all of the queries. In the computation, the recall values are rounded to the nearest 0.05 increment and interpolation is used to obtain the precision values for all 21 recall points (from 0 to 1, at 0.05 increments). Note that owing to interpolation, the precision at the zero recall point is actually the precision of the recall point which is nearest to zero. The 21 pairs of values can then be plotted in a precision and recall graph. This procedure is commonly used in evaluating text retrieval methods [Harman 1993b; Salton and McGill 1983]. For clarity and conciseness reasons, we further average the 21 precision values to obtain the *average precision* for each set of parameters used in the experiments. In order to observe the effects of false drops on precision and recall and storage overhead, we use a very wide range of false drop probability in the experiments. In practice, the false drop probability is typically below 1%.

Some characteristics of the document collections and queries are given in Table 3.2 and Table 3.3 respectively. Note tf_1 terms outnumber any other groups in both collections. On average, TREC subset documents are five times longer than CACM documents.

	CACM	Percentage	TREC	Percentage
tf_1 terms	73,294	78.08%	1,035,521	72.78%
tf_2 terms	11,841	12.62%	208,121	14.63%
tf_3 terms	4,252	4.53%	75,908	5.34%
tf_4 terms	2,026	2.12%	36,482	2.56%
tf_5 terms	1,083	1.15%	20,324	1.43%
tf_6 terms	582	0.62%	12,452	0.875%
Terms with $tf \geq 7$	786	0.837%	33,964	2.3872%
Total terms	93,867		1,422,772	
collection size	3,204		10,000	
Average number of terms	29.29		142.3	
Average document length	462.1		2716.08	

Table 3.2: Characteristics of the document collections

	CACM Queries	TREC Queries
Number of queries	64	25
Number of terms	798	621
Number of unique terms	430	403

Table 3.3: Characteristics of the query collections

We also tabulate the distributions of true matches between the queries and documents among different tf groups. The results are plotted in Figure 3.4 for CACM and Figure 3.5 for TREC subset.

From Figure 3.4 and Figure 3.5, it is clear that the true matches overwhelmingly concentrate on low tf end. In fact, more than two thirds of the true matches occur in tf one group for both CACM and TREC subset. There are not many matches at the high tf -end for CACM due to the lengths of documents in that collection. Terms with $tf = 1$ is a very important term frequency group. Any fast search algorithm has to pay extra attention to this group of terms. On the other hand, matches (false or true) in higher tf groups will likely contribute more to the final ranking of the documents. Group wise, tf_1 terms are the most important, whereas tf_l terms are the least important ones term wise. Balancing these two factors will be central to achieve good performance.

Three different signature length assignment methods are tested in the experiments. The first two are derived from the optimal formulas (Eqn. 3.13) and (Eqn. 3.14) using (1) term frequencies as ranking weights of the signature partitions (denoted as TF in the graphs) and (2) uniform weights (denoted as U). The third method assigns signature lengths based upon the following table:

term frequency	multiplying factor
1	1
2	2
3-29	4
30	8

This method is based on the intuition that longer signature lengths should be used for higher term frequencies to reduce false drops occurred at high TF partitions. However, the multiplying factors are arbitrarily chosen. The inclusion of this method

Figure 3.4: Distribution of true matches in CACM

Figure 3.5: Distribution of true matches in TREC

provides a data point with which the other two methods can compare. This method is denoted as EXP in the graphs.

Note that in all experiments, the maximum term frequency is set to 30. For large collections (used in experiments described later), there may be many terms with term frequencies greater than 30 and thus are mapped to $tf = 30$. This may result in a larger number of terms for $tf = 30$ (or whatever ceiling was chosen).

The storage overhead is used as the main controlling parameter in the experiments. Storage overhead is measured by the size of the signature file divided by the total size of the documents *after stemming and common word removal*. This measures the true storage overhead attributable to the signature file mechanism (as opposed to the space reduction achieved by stemming and common word removal). When the raw collection size is used, the storage overhead is much lower. For instance, in the TREC subset, the storage overhead calculated using the raw collection size is typically only half of the storage overhead calculated using our method.

For a fixed storage overhead, the signature lengths and the other related parameters are computed based on the three methods described above.

Performance of HL and LH strategies

Figures 3.6 and 3.7 compare the average precisions of the HL and LH search strategies using the CACM collection and the TREC subset. It is clear that the HL method is consistently better than the LH method, as predicted by our analysis. The difference is large for high false drop probabilities (i.e., low storage overheads) but narrows down as the false drop probability decreases. In general, the gap between the HL and LH methods increases when the number of false drops increases. This explains the large gap in the EXP method, because the EXP method produces the

largest number of false drops given a fixed storage overhead. Since the HL method is much better than the LH method, we will only show the performance of the HL method in subsequent discussion.

Comparison of signature length assignment methods

The performance of the three signature file generation methods for the HL search strategies is shown in Figures 3.8 and 3.9. We can see from the figures that the EXP method is consistently the worst in all the test runs, whereas the uniform and optimal methods give much better performance. Further, the optimal method is the best everywhere. In fact, the LH approach in the optimal method is as good as the HL method in the uniform method. This confirms our previous discussion that the optimal method utilizes the space resource more effectively.

Effect of false drops on retrieval effectiveness

We can see from the graphs that although the average precision in general improves as false drop probability decreases, there is no observable improvement on retrieval effectiveness beyond a certain threshold. For the TREC subset, the average precision reaches the perfect level (when there is no false drops) when the storage overhead is about 37%. According to Figure 3.13, this corresponds to roughly 0.01% false drop probability and about 150 false drops generated from a query.

Retrieval accuracy

In their research on documents ranking, Wong and Lee [56, 58] introduce the concept of retrieval accuracy to measure the performance of their heuristic methods. They measure the percentages of top documents obtained at different check points of total disk access. However measuring the percentages of top documents obtained

Figure 3.6: Performance of LH versus HL search strategies on CACM

Figure 3.7: Performance of LH versus HL search strategies on TREC

Figure 3.8: Comparison of signature length assignment methods (CACM. HL)

Figure 3.9: Comparison of signature length assignment methods (TREC subset, HL)

at different check points of total disk access does not apply to TF-partitioned signature file model due to the orthogonality of signature file structure and inverted file structure. We will apply a similar measure to evaluate the performance of our TF-partitioned retrieval model. Our check points are space overhead of the generated signature files. First several definitions:

1. TD_f : The final set of top documents obtained when searching the document collection without any false drops. The number of top documents returned is determined by the system or the user.
2. TD_i : The set of top documents obtained after searching signature files and the space overhead is $i\%$, where i is an integer between 0 and 100.
3. Ra_i (Retrieval accuracy): $(|TD_i \cap TD_f| / |TD_f|) \times 100\%$.

The values of Ra 's indicate the percentages of top documents obtained at different storage overhead. Moreover, just as pointed out by Lee and Wong in their research, a sequence of Ra 's can show how fast the top documents in TD_f are revealed. A fast increasing of Ra 's with respect to the space overhead means the top documents are revealed with relatively less space used. It is thus clear that the faster Ra increases the better. We did experiments with prespecified top document collection sizes of 10, 20 and 50 for CACM and TREC subset for optimal and uniform models. We did not experiment with the EXP method, since it has been shown that it is not a good retrieval model. The results are plotted in Figure 3.10 for CACM and Figure 3.11 for TREC subset. From these graphs, we can see that without exception, Optimal method produces better retrieval accuracy. Not surprisingly, the best possible performance is achieved at space overhead about 40%. This is in line with the results

we presented earlier on average precision. With about 40% of the space overhead, we can achieve the precision of exact matching.

Effect of collection characteristics

We can see from the tests examined so far, the TREC subset behaves much better than CACM. For instance, the methods reach the perfect precision level much earlier in TREC than in CACM. The reason for the better performance in the TREC subset is two-fold: (1) the long document length results in many document terms falling in the high-term frequency range, so the HL search strategy is particularly favorable since a match is likely to occur in the high term-frequency range, resulting in a lower chance of encountering false drops; (2) the long queries generate a large number of query terms; thus, a few false drops produce relatively smaller effect on the overall document scores.

Number of false drops and false drop probability

Figures 3.12 and 3.13 show the average number of false drops produced by a query and the weighted false drop probability. The former is obtained by counting the total number of false drops in each run and then dividing it by the number of queries in each run. The latter is computed from the analytical formula (Eqn. 3.9). We can see that the analytical and experimental data are very consistent with each other. The uniform and TF methods are almost identical in the number of false drops as well as in false drop probability. The EXP method is much worse than the other two methods.

Figure 3.10: Retrieval accuracy for top 10, 20 and 50 documents on CACM

Figure 3.11: Retrieval accuracy for top 10, 20 and 50 documents on TREC

Figure 3.12: Number of false drops (FDs) and weighted false drop probability (FDP) for CACM (HL)

Figure 3.13: Number of false drops (FDs) and weighted false drop probability (FDP) for TREC (HL)

The 20% space overhead folklore

Unlike inverted files which may have more than 100% percentage overhead, it has been long believed [7] that signature file is typically 10-20% the size of primary database. However, if we cannot implement $tf * idf$ ranking, we cannot compare the storage overhead on equal footing. This remains a belief until now. With the successful encoding of term frequency in TF-partitioned signature file, we can now compare these two approaches on well defined benchmarks like precision and recall. Our experimental data offers quantitative proof of this belief. In Figure 3.7, Figure 3.6, Figure 3.8 and Figure 3.9, TF-partitioned signature files achieve same precision as inverted files when storage overhead reaches 40%, which is about 20% of the raw database. In our experiments, storage overhead is measured by the size of the signature file divided by the total size of the documents *after stemming and common word removal*.

CHAPTER 4

Fast Heuristic Search Algorithms

Signature file based retrieval systems suffer from slow response time problem compared to inverted file based systems. Since the size of the signature file is directly proportional to the size of the database, retrieving all or some of the top documents satisfying a query requires exhaustive search of the whole signature file. Therefore, the time spent on searching the signature file is proportional to the database size, and performance becomes a concern for large databases. In this section, we study algorithms to speed up the search process by reducing either the percentage of signatures searched or the total size of the signature file.

4.1 Search Space Reduction in HL Search Method

It is clear that the HL search method does not search the signature files exhaustively. Once a match is found, the search can proceed immediately to the next query term. This undoubtedly results in better performance. It is interesting to see how much performance gain we get.

For simplicity, we assume that all signatures in $\cup_{j=i}^n S_j$ are searched if a match is found in SF_i , even though the match may occur at the beginning of SF_i . It is clear

that the number of signatures searched is estimated conservatively. We will consider HL search method only since it has been proven to be superior.

Lemma 10 *Suppose for a document D , there are $l_1, \dots, l_n$ signatures in $SF_1, \dots, SF_n$, respectively. The search reduction ratio of HL method can be approximated as:*

$$l_1 \sum_{i=2}^n (1 - \frac{1}{i}) \frac{1}{i(i+1)} p(m_i, w_i, s).$$

Proof: Let $S_i, i = 1, \dots, n$ denote the expected number of signatures searched in HL search method. It is easy to see that

$$\begin{aligned} S_i &= l_i + (1 - q_i) S_{i-1} \quad \text{for } i = 2, \dots, n \\ S_1 &= l_1. \end{aligned}$$

Solving for S_n , we have

$$\begin{aligned} S_n &= l_n + \sum_{j=1}^{n-1} \prod_{i=j+1}^n (1 - q_i) l_j \\ &\approx \sum_{i=1}^n l_i - \sum_{k=2}^n (\sum_{j=1}^{k-1} l_j) q_k \end{aligned}$$

From Zipf's Law and formulae (3.6) and (3.7), we have for $k = 1, \dots, n$,

$$\sum_{j=1}^{k-1} l_j = \sum_{j=1}^{k-1} \frac{2}{j(j+1)} l_1 = 2(1 - \frac{1}{k}) l_1 \quad \text{and} \quad \sum_{j=1}^k l_j = 2l_1.$$

This implies that the percentage of signatures searched is

$$\begin{aligned} \frac{S_n}{\sum_{i=1}^n l_i} &= 1 - \sum_{i=2}^n (1 - \frac{1}{i}) q_i \\ &\approx 1 - \sum_{i=2}^n (1 - \frac{1}{i}) l_i p(m_i, w_i, s) \\ &\approx 1 - l_1 \sum_{i=2}^n (1 - \frac{1}{i}) \frac{1}{i(i+1)} p(m_i, w_i, s) \end{aligned}$$

Thus the reduction ratio is

$$l_1 \sum_{i=2}^n (1 - \frac{1}{i}) \frac{1}{i(i+1)} p(m_i, w_i, s).$$

□

For example, if each document has on average of 30 signatures in SF_1 , $n = 30$, and $p(m_i, w_i, s) = 0.003$, then the reduction ratio is about 0.03, or 3%. Although the gain is not significant, it is obtained absolutely free for TF-partitioned signature file.

We plot the search reduction ratios for CACM and TREC subset in Figure 4.1 and Figure 4.2. The reduction ratio decreases sharply as storage overhead runs from 10% to 35% and stabilizes around 5% for CACM and 7% for TREC subset. These ratios are a little better than the analytical results because

1. Our estimation is conservative.
2. We assumes that all signatures are randomly generated bit vectors. And this is not the case in the sample queries we used.

Another interesting thing is that the optimal method is actually the worst in terms of search space reductions. This is another proof that optimal signature configuration utilizes space more efficient than other approaches. It leads to the least number of false drops.

We can see that although the TF-partitioned signature file method does not search all signatures exhaustively, usually more than 94% of the signatures still have to be searched for reasonable precision and recall. This is shown by analysis and confirmed by experiments. This kind of reduction ratios will not make any noticeable improvement on the search speed. So as a new document retrieval and ranking model,

Figure 4.1: Search reduction ratio (CACM, HL)

Figure 4.2: Search reduction ratio (TREC, HL)

TF-partitioned signature files method suffers slow response time problem as general signature file based indexing and retrieval model. To be a viable document retrieval model, fast search algorithms have to be studied and significantly better search reduction ratio has to be achieved.

Fast Heuristic Search Algorithms

There has been extensive research done on how to improve the search speed of signature files. Most notable ones are key-based signature file partitioning methods [Lee and Leng 1989; Zezula et al. 1991]. These algorithms have achieved remarkable search space reduction ratios with little overhead.

Among the three partitioning methods, Lee and Leng [30, 31] have shown that floating-key partitioning yields the best search space reduction. All three partitioning methods introduce very little overhead. The reduction ratio ranges anywhere between 15% to 85% depending on the query signature weight. Many partitions are filtered out by the key comparison during search.

Zezula and Tiberio et al. [41] proposed a new dynamic partition model, Quicker Filter, to reduce search reduction ratio. This method allocates signatures on disk based on linear hashing functions. However these algorithms do not perform well for query signatures with low weights. This deficiency limits the usefulness of these algorithms. For instance, most Internet search queries consist of single query term, which results in query signature with low weight.

The uniqueness of TF-partitioned signature file structure calls for new algorithms to improve search reduction ratio. In this section, we introduce an upperbound search

algorithm which achieves good reduction ratio consistently. The concept of upper-bound search algorithm [51, 36, 40, 55, 56] has been employed in inverted file based systems with various degrees of success. However we will see that upperbound search method yields significant better results with TF-partitioned signature files.

4.2 Upperbound Search Algorithm in Inverted File

First we present a brief review of the upperbound search algorithm in an inverted file. Detailed information can be found in [51, 36, 40, 55, 56]. The upperbound search algorithm is used to search for certain prespecified, say top T , top documents in a document collection. Given a query $Q = \{t_1, t_2, \dots, t_k\}$ of k terms, Q is first sorted by increasing postings list length. In this way, terms with high *idf* values will be processed first, while long postings lists are processed at the end. The upperbound search algorithms works as follows. After a postings list is processed, partial document scores are sorted in descending order. The current top T documents are obtained. The upperbound of the $(T + 1)^{th}$ document is then computed with the assumption that it contains all of the remaining unsearched query terms. The search stops if the upperbound of the $(T + 1)^{th}$ document is smaller than the current score of the T^{th} document. If the search can be terminated early, we have better response time since short postings lists are processes first and the unprocessed lists are long.

It has been found [5, 56] that the upperbound search algorithm is too conservative to yield any significant performance gain for inverted files. Most of the time, almost all the query terms, thus most documents, must be processed in order to obtain all top documents.

Some variations of the upperbound search algorithm have been proposed to improve retrieval efficiency by relaxing the stopping criterion [5], or applying probability on the upper bound computation [Perry, Willett 1983; Weiss 1981]. Again it has been found [Wong 1991] that those modified upperbound search algorithms fail to yield any real performance gain and the extra computation overhead induced to calculate probabilities is very expensive. The stopping condition, hence the search reduction ratio, of these upperbound search algorithms solely depends on the relation between just two document scores, namely $S(T)$ and $S(T + 1)$. It does not utilize the overall distribution of all document scores.

4.3 Upperbound Search Algorithms for Weight-partitioned Signature File

The main idea of our upperbound search algorithm is to search signature files SF_n to SF_2 first and establish a coarse ranking using the generated document scores. Then for each document D , the SF_1 signatures of D are searched further only if the final score of D has any chance to be among the top T scores. If a large number of SF_1 signatures can be excluded from search process, we can expect better response time and fast retrieval.

According to Zipf's law (3.7), the number of terms with frequency greater than one is

$$I_{\geq 2}(D) \approx \frac{2}{2} I_1(D) = I_1(D).$$

In other words, about half of the indexed terms are in tf_1 . From Table 3.2, we can see that the percentages are even higher for the test collections we used. In CACM and TREC subset, 78.08% and 72.78% of the terms, respectively, have term frequency

equal to one. This group of terms accounts for large proportion of the signatures; yet each individual term contributes minimally in the $tf * idf$ weighting schema since $tf = 1$. The pseudo code of our upperbound search algorithm is given in Algorithm 3.

Algorithm 3: Fast Signature File Search

1. for all query term $t_i, i < k$ in query Q do
2. obtain $idf(t_i)$ of term t_i ;
3. end {for}
4. for every document D do
5. generate a k -bit vector $B(D) = \langle b_1 \cdots b_k \rangle$
6. set all k bits to zero
7. end {for}
8. search SF_n to SF_2 according to Algorithm 2.
9. if there is a match between t_i 's signature and any signature of
10. document D . set the i^{th} bit of $B(D)$ to 1 while searching SF_n to SF_2 .
11. sort current document scores by descending order.
12. set $S(T)$ = the score of the T^{th} document.
13. /* now search signature file SF_1 */
14. for every document D
15. set $U(D) = S(D) + (\sum_{1 \leq i \leq k, B(D)[i]=0} 1 * idf(t_i)) / \sqrt{d_i}$
16. if $U(D) < S(T)$
17. skip this document
18. else
19. /*search this doc*/

```

20.         for each query term  $t_i$  do
21.             if the  $i^{th}$  bit of  $B(D)$  is 0
22.                 search the document signatures for  $t_i$ .
23.             end {if}
24.         end {for}
25.     end {if}
26. end {for}
27. sort final scores by descending order
28. output the document ids of the top  $T$  documents.

```

The bit vector for each document is used to avoid matching a query term multiple times in one document. It records all matches while searching SF_n to SF_2 . If a term is found in $SF_i, i > 1$, we certainly should not search it again in SF_1 . These bit vectors take very little space. For instance, the TREC subset has 10,000 documents. It takes about 10K bytes for a query with 8 terms. This amounts to almost nothing to modern computers.

In Algorithm 3, lines 15, 16 and 17 are the key steps to achieve good reduction ratio. In line 15, we calculate the upperbound score of document D by assuming that all unmatched terms exist in document D . Each unmatched term t generates $tf(t)*idf(t) = 1*idf(t)$ points since SF_1 signatures are generated from tf_1 terms. The upperbound score $U(D)$ is obtained by adding current score $S(D)$ and $\sum_{i < k, B(D)[i]=0} 1*idf(t_i)$, extra score generated by those unmatched terms. Lines 16 and 17 exclude a document from further search if it has no chance to be among the top T documents.

Unlike existing upperbound search algorithms, Algorithm 3 does not wait passively for the magic stop point to end the whole search process. Instead, it actively attempts

to eliminate as many documents (signatures) as possible from further search. We now show the correctness of Algorithm 3 with the following theorem.

Theorem 4 *Algorithm 3 returns the same set of top T documents as Algorithm 2. Furthermore the scores of the top documents are exactly the same as in Algorithm 2. and the order of top documents generated by Algorithm 3 is identical to Algorithm 2.*

Proof: We prove this by two steps. First we show that for each document, the score generated by Algorithm 3 is less than or equal to the score generated by Algorithm 2. In fact, we can see that any match in Algorithm 3 is also a match in Algorithm 2 within the same SF_i . This is clear for any match in SF_2 to SF_n because they employ the same search strategy, the HL approach, over the same set of signatures. The bit vector for each document eliminates multiple matches for any query term. For any match of Algorithm 3 in SF_1 , we know the corresponding bit cannot be 1. In other words, Algorithm 2 will continue the search until the same match occurs. Thus we can conclude that Algorithm 3 does not introduce any more match than Algorithm 2. This implies that the score generated by Algorithm 3 is less than or equal to the score generated by Algorithm 2.

Next we show that for the two top T document collections generated by Algorithm 2 and Algorithm 3 are identical and the score generated by Algorithm 3 for any of the top documents is exactly the same as the score generated by Algorithm 2. It is enough to show that for any document in the top T documents of Algorithm 2, Algorithm 3 does not miss any match as Algorithm 2 for that document. For simplicity, let us assume D is one of the top T documents output by Algorithm 2 and the scores it receives from Algorithm 2 and Algorithm 3 are $S_2(D)$ and $S_3(D)$, respectively. From the above discussion, we know that any match in Algorithm 3 is also a match

in Algorithm 2 within the same signature file and $S_3(D) \leq S_2(D)$. If Algorithm 3 misses any match, this can only happen if this document fails the test on line 16 of Algorithm 3. We know the score of D in Algorithm 2 is

$$S_2(D) = \text{score obtained from } SF_2 \text{ to } SF_n + \text{score obtained from } SF_1$$

The worst case is that all remaining unmatched query term signatures produce matches in SF_1 , This means

$$\begin{aligned} S_2(D) &= \text{score obtained from } SF_2 \text{ to } SF_n + \text{score obtained from } SF_1 \\ &\leq \text{score obtained from } SF_2 \text{ to } SF_n + \sum_{i < k, B(D)[i]=0} 1 * idf(q_i) \\ &\leq \text{The score of the } T^{th} \text{ document in Algorithm 3} \end{aligned}$$

Thus for all the top T documents output by Algorithm 3, say $D_1, \dots, D_T$, we have $S_2(D) < S_3(D_i) \leq S_2(D_i), i = 1, \dots, T$, which contradicts the assumption that D is one of the top T documents in Algorithm 2. $\square$

The bit vector associated with each document in Algorithm 3 is used to eliminate multiple matches for a single query term. Although using little memory, it increases implementation complexity. The probability that a query signature has matches in both $\bigcup_{i=2}^n SF_i$ and SF_1 simultaneously can be calculated as

$$q_1(1 - \prod_{i=2}^n (1 - q_i)) \approx q_1(q_2 + \dots + q_n),$$

where q_i is the probability that the signature has a match in SF_i , $1 \leq i \leq n$. Note this is an order two polynomial of $q_1, q_2, \dots, q_n$, which is extremely small considering that $q_1, q_2, \dots, q_n$ are already very small. Thus dropping the bit vectors in Algorithm 3 has little, if any, impact on the final ranking and precision. We will not have the

exact scores as regular HL search without it though. We can have a simplified version of the upperbound search algorithm by dropping the bit vectors.

The exact reduction ratio of Algorithm 3 is hard to predict due to the randomness of text collections. Clearly the best reduction ratio occurs when all but the top T documents' signatures in SF_1 are skipped. The reduction ratio then is basically whatever the ratio of SF_1 signatures among all signatures. The worst case occurs when none of the documents is skipped, which degrades the reduction ratio of Algorithm 3 to that of Algorithm 2. In reality, those two scenarios are extremely unlikely to happen. We will see that the reduction ratio of Algorithm 3 is significant. From the numerous experiments we did, the number of signatures searched was consistently reduced by more than half without sacrificing any precision and recall for top documents.

The key to achieve better search reduction ratio is to have large $S(T)$ when it is time to search SF_1 signatures in Algorithm 3. From lines 15 and 16 of Algorithm 3, we can see that the larger $S(T)$ is, the more signatures are skipped, and the better reduction ratio we get. Instead of using constant $S(T)$ when searching SF_1 signatures, we can update $S(T)$ by sorting the latest document scores after some SF_1 signatures have been processed. The new T^{th} score is used as the threshold. Since the score of each document cannot decrease, we will certainly get larger and better $S(T)$ and more SF_1 signatures will be skipped.

Incorporating these ideas, we have Algorithm 4, which is presented here for simplicity and clarity. Some omitted details can be found in Algorithm 3.

Algorithm 4: Fast Signature File Search With Update

1. create a bit vector for each document.
2. search SF_n to SF_2 , record matched terms on bit vectors.


```

3. for ( $id = 0; id < N; id++$ )
4.   if ( $0 == id \% uf$ ) do
5.     obtain latest  $S(T)$ .
6.     set  $U(D_{id}) = S(D_{id}) + (\sum_{1 \leq i \leq k, B(D_{id})[i]=0} 1 * idf(t_i)) / \sqrt{d_i}$ 
7.     if  $U(D_{id}) \geq S(T)$  do
8.       search  $SF_1$ .
9. end {for}

```

In Algorithm 4, constant uf is a design parameter which decides how often $S(T)$ is updated. Its value can be anywhere from 1 to N , the number of documents in the collection. Algorithm 3 corresponds to the case $uf = N$, in which $S(T)$ is never updated. It is clear that updating $S(T)$ after each document ($uf = 1$) produces the best search reduction ratio. But the overhead of calculating $S(T)$ so often may diminish the performance gain. The proper update frequency, uf , can be obtained by empirical tests. In lines 4 and 5 of Algorithm 4, the threshold $S(T)$ is updated after searching every uf documents. Line 6 calculates the upperbound for the documents as in line 15 of Algorithm 3.

4.4 Search Reduction Ratios of Heuristic Algorithms

We did experiments with Algorithm 3 and Algorithm 4 for searching top $T = 20$ documents. The reduction ratios were calculated as the average reduction ratios among the standard queries, 64 for CACM and 25 for TREC subset.

The update frequency uf was set to 300 for CACM and 1000 for TREC subset. These values are roughly 10% of the number of documents in both collections. After searching every 300 (1000) documents, we sort all current scores and use the new T^{th}

score, $S(T)$, as new threshold (line 7 of Algorithm 4). The test results are plotted in Figure 4.3 for CACM and Figure 4.4 for TREC subset.

From Figure 4.3 and Figure 4.4, we can see the reduction ratios are substantial. For example, the reduction ratios for optimal without updating range from 43% to 57% for CACM, and 60% to 65% for TREC subset. The uniform method's reduction ratios are even better, 50% to 65% for CACM and 64% to 70% for TREC subset. The reduction ratios do not fluctuate as the query signature weight changes. These algorithms perform well even with low query weights. The search reduction ratios will certainly translate into fast search speed. Several points are worth making:

1. The reduction ratio decreases as storage overhead goes up. More space implies fewer false drops. This, in return, gives us small and tight $S(T)$ and more signatures in SF_1 have to be searched.
2. Updating $S(T)$ while searching SF_1 improves search reduction ratios. The improvements are about 2% for CACM and less than 1% for TREC subset. The effectiveness of updating $S(T)$ while searching SF_1 is questionable. Updating $S(T)$ performs better for CACM because CACM documents are on average short and have more terms in tf_1 group percentage wise. Frequency one terms, thus SF_1 signatures, carry more weight in short documents in terms of document ranking
3. The reduction ratios for TREC subset are consistently better, about 7%, than CACM's. In general, CACM documents are shorter than TREC subset documents. There are larger percentage of tf_1 terms in CACM, which diminishes the search reduction ratio for Algorithm 3. On the other hand, signatures in

Figure 4.3: Search reduction ratio for top 20 documents (CACM, HL)

Figure 4.4: Search reduction ratio for top 20 documents (TREC, HL)

$SF_i, i > 1$ of TREC subset more or less determine the set of top documents and their rankings. Algorithm 3 and 4 are more effective for long document collections.

4. The reduction ratios for the uniform method are better than the reduction ratios of the optimal method. The differences are about 6% CACM and 4% for TREC subset respectively. This is no surprise considering that the uniform method allocates more storage to SF_1 than the other two methods.

Finally, there is no reason to limit the document elimination process to SF_1 . The same operation (Lines 15, 16 and 17 in Algorithm 3) can be applied after searching SF_n , then after searching SF_{n-1} and so on, or after a batch of SF_j 's. If a document is excluded after searching SF_j , all signatures of this document in $\bigcup_{i < j} SF_i$ are excluded from further searching.

4.5 Weight-Partitioned Signature File Under Low Storage

In the previous section, we propose several algorithms to speed up the search process by reducing the percentage of signatures that need to be searched. But the fundamental cause for slow response time of the signature file is the overall size of the signature files. Small signature files mean fast response time, even though there is little need to squeeze for storage saving today. It is safe to say size is speed. From the performance figures 3.8 and 3.9, we can see that the average precisions at low storage overhead (e.g., 10-20%) are unacceptably low. The average precision is less than 60% of the false drop free precision when storage overhead is between 10-20%. It is important to improve the performance of the signature file under low storage overhead. In this section, we look into the performance problem from a different angle.

study ways to reduce the total signature file size while achieving excellent precision and recall. We explore two different approaches in this section.

4.5.1 Exact Search for Top Documents

The first approach is to use the signature files to produce a coarse ranking and apply exact matching (without false drops) only to the top ranked documents, thus obtaining low storage overhead and high precision. The percentage of the top documents searched should be small to have good response time and extensive enough to improve precision and recall of the retrieval. The performance of this approach is fundamentally tied to the ranking generated by searching the signature files. For instance, in a unlikely worst case, if no relevant documents are among the top 50 documents generated by searching the signature files due to large number of false drops, exact searching of the top 50 documents will not produce any significantly better result. The algorithm can be conceptually described as follows.

Algorithm 5: Exact Search for Top Documents

1. search SF_n to SF_1 with HL approach.
2. find the top T_{es} documents.
3. search the text of top T_{es} documents.
4. return top T documents.

where T_{es} stands for the number of top documents we want to search. This approach is relative easy to implement on top of the TF-partitioned signature file system. In our experiments, we did exact search for the top 50 and 100 documents, i.e., $T_{es} = 50$ and $T_{es} = 100$. The experiment results are plotted in Figure 4.5, 4.6 for CACM and Figure 4.7, 4.8 for TREC. From those figures, we can see that the average precision

improvements are significant when space overhead is less than 20%. The average precision of the optimal method reaches the level achieved by false drop free searching with about 25% storage overhead for both CACM and TREC subset. The drop of storage overhead from 40% to 25% represents a 37.5% reduction, which translates into 37.5% improvement on response time.

4.5.2 High Frequency Term Indexing

As we mentioned earlier, tf_1 terms, in general, contribute less to the final ranking, yet they take more than 50% of all storage overhead. In Algorithm 3 and 4, many signatures are excluded in the search process to reduce search space. In certain degree, it is a waste to index and store signatures from tf_1 terms. We now try to just index and generate signatures for terms with frequency more than one. If this approach produces good precision and recall with low storage overhead, we have fast search algorithm for all practical purposes. The precision and recall undoubtedly suffer from leaving tf_1 terms out all together, especially for short document collections like CACM. To compensate for this, we perform exact text search to top documents generated after searching $SF_j, j > 1$. The algorithm can be conceptually described as follows.

Algorithm 6: High term frequency signature file searching.

0. /* only terms with $tf > 1$ are indexed.*/
1. search SF_n to SF_2 with HL approach.
2. find the top T_{es} documents.
3. search the text of top T_{es} documents.
4. return top T documents.

Figure 4.5: Exact match for top 50 and 100 documents (CACM, uniform)

Figure 4.6: Exact match for top 50 and 100 documents (CACM, optimal)

Figure 4.7: Exact match for top 50 and 100 documents (TREC, uniform)

Figure 4.8: Exact match for top 50 and 100 documents (TREC, optimal)

The real difference between Algorithm 6 and Algorithm 5 is the underlining signature files. All tf_1 terms are left out in this method. We did some tests with CACM and TREC subset by doing exact matching on top 50, 100, 200 documents. The experimental data were collected and plotted in Figure 4.10 and Figure 4.12.

From these two figures, we can see searching high term frequency groups like $SF_{j,j} > 1$ alone will not achieve the same precision as regular TF-partitioned signature files, no matter how much storage we use and how rare the false drops are. Though the degradation varies with collections. We lose about 16.47% and 4.4% of average precision for CACM and TREC subset respectively. The tf_1 terms, as a group, carries real weight in document ranking.

Doing exact matching for top documents does not improve enough for CACM collection. Documents in CACM collection, which contains 3,204 bibliographic records from the *Communication of ACM*, tend to be very short. Terms in $tf_{j,j} > 1$ cannot provide acceptable representations of short documents, since not many terms appear in a single document multiple times. On the other hand, documents in TREC subset are much longer. Terms in $tf_{j,j} > 1$ provide a relatively faithful representation of the document content. For any query term, it would be unusual for a long document to be highly relevant and contain the term only once. With 10% storage overhead, the precision achieved by Algorithm 6 is almost as good as searching TREC subset with no false drops. The difference (degradation) for average precision is less than 1%, which is a small trade-off. Thus searching signature files from SF_n to SF_2 plus searching top documents yields good precision with low storage overhead.

In section 3.4, we know that it takes 40% of space overhead to achieve the precision and recall level of false drop free searching. The storage overhead difference between

10% and 40% represents a 75% search space reduction. In other words, response time is shortened by 75%, let alone the time saved during indexing. The small precision degradation for a long document collection can easily be compensated for by incorporating techniques such as user feedback etc.

Unfortunately ignoring tf_1 terms in short documents is a problem. One way to overcome this problem is to do full text indexing for short documents and selective indexing for long documents. Then we weed out the unqualified short documents using Algorithm 3 and 4 during search. We will explore those ideas in future research.

Figure 4.9: Performance of high TF term indexing (CACM, uniform)

Figure 4.10: Performance of high TF term indexing (CACM, optimal)

Figure 4.11: Performance of high TF term indexing (TREC, uniform)

Figure 4.12: Performance of high TF term indexing (TREC, optimal)

CHAPTER 5

Key Partitioned Signature Files

Many researchers have been studying new methods to speed up retrieval process by reducing the search space. Lee and Leng [30, 31] proposed several ways to partition the signature files according to certain keys of the signatures. With very little overhead, their methods reduce the number of signatures searched by 15% to 80%, depending on the query signature weight.

In this thesis, we propose some improvements on Lee and Leng's partitioning methods to further reduce the search space. Fast and accurate analytical results are obtained for many performance measures. We also answer an interesting question concerning any key based partitioning methods.

In the fixed-prefix partitioning, the key length is fixed and a long linked list may be created at the leaf nodes when all key bits are exhausted. The search reduction ratio suffers when this occurs. To remedy this problem, we introduce a variation of the fixed-prefix partitioning schema, namely variable-prefix partitioning in this chapter. We will study the performance of the new variable-prefix partitioning. Our main goal is to improve the search reduction ratio.

Throughout this section, we use m to denote the signature length, n the number of signatures in the database, k the key length and p the page size. We also use w

to denote the possibility that a signature bit is set to 1. The probability w is also called bit weight of the signature. For generality, we consider the general case, that is w can be anywhere between 0 and 1 ($0 < w < 1$) even though $w = 0.5$ is the optimal value. The reason will be made clear in later sections. Notation $\mathcal{S}(n, m, w)$ will be used to denote a collection of n signatures with length m and bit weight w . Other notations will be introduced when they are used. For convenience, we collect important notations used in this chapter in table 5. We will use page and partition interchangeably since we use fixed page size. Since page size p is usually fixed, we will drop p from functions for simplicity from time to time.

5.1 Variable-prefix Partitioning

Theoretical results

As implied by the name, the fixed-prefix key partitioning method requires a fixed key length. If the key length k is long, it will result in many empty or sparse partitions. On the other hand, small k will result in large partitions and increase search time. Proper key length needs to be selected first based on the number of signatures in database. This is not always possible if the database grows and shrinks over time.

With the partition file structures introduced in [31], it is possible to eliminate the limitation of partition key length. In the trie structure described in [31], key length k is increased or linked pages are used when the depth of the trie reaches k . One natural way to generalize this is to split and create new partitions whenever it is necessary. The prefix is still used as the partition key but without any length limitation.

We will call this method variable-prefix partitioning method since it shares many characteristics with the fixed-prefix partitioning method. In fact, the fixed-prefix

k	Key length in bits.
m	Signature length in bits.
w	Probability of a signature bit is set to 1.
w_Q	Probability of a query signature bit is set to 1.
n	Number of total signatures in a database.
p	Page size (number of signatures it can store).
m_k	Number of key segments for floating key partitioning.
$S(n, m, w)$	A set of n signatures with length m and bit weight w .
$S(j)$	Signatures with key position j .
$P(j, i)$	Probability a signature is in $S(j)$ and with i bits set in key section.
$P(j)$	Probability a signature is in $S(j)$.
$w(j)$	bit weight of key segment j .
S_i^k	i^{th} k bit segment of signature S .
$SSR(n, m, k)$	Storage saving ratio.
$P_1(n, k, m, w[, p])$	Number of pages generated for fixed-prefix.
$P_2(n, m, w[, p])$	Number of pages generated for variable-prefix.
$P_3(n, k, m[, p])$	Number of pages generated for floating-key.
$L_1(n, k, m, w[, p])$	Longest path generated for fixed-prefix.
$L_2(n, m, w[, p])$	Longest path generated for variable-prefix.
$S_1(w_Q, n, k, m, w[, p])$	Signature search reduction ratio for fixed-prefix.
$S_2(w_Q, n, m, w[, p])$	Signature search reduction ratio for variable-prefix.
$S_3(w_Q, n, k, m[, p])$	Signature search reduction ratio for floating-key.
$R_1(w_Q, n, k, m, w[, p])$	Partition (page) search reduction ratio for fixed-prefix.
$R_2(w_Q, n, m, w[, p])$	Partition (page) search reduction ratio for variable-prefix.
$R_3(w_Q, n, k, m[, p])$	Partition (page) search reduction ratio for floating-key.

Table 5.1: Symbols for Variable-prefix Partitioned Signature Files

partitioning becomes variable prefix partitioning when the signature length is used as key length. The variable-prefix partitioning is just a special case of the fixed-prefix partitioning. We will provide an analytical study of the expected number of partitions generated and search reduction ratio of the variable-prefix partitioning method. Without loss of generality, we will assume that the number of signatures is far greater than the page size p . With less than p signatures, at most one page is generated.

Since the variable-prefix partitioning method is a degenerate case of the fixed-prefix method, we formulate our results for the fixed-prefix method first. The results can be easily generalized to the variable-prefix partitioning method. The first result is about the number of pages generated.

Lemma 11 *Suppose we have a signature collection $\mathcal{S}(n, m, w)$. If we partition the signatures into a trie structure with prefix key of length k , the expected number of pages generated, $P_1(n, k, m, w)$, can be calculated by*

$$P_1(n, k, m, w) = \begin{cases} \lceil n/p \rceil & \text{if } k = 0. \\ \lceil n/p \rceil & \text{if } n \leq p. \\ \sum_{l=0}^n \mathcal{P}(n, l, w) (P_1(l, k-1, m-1, w) + P_1(n-l, k-1, m-1, w)) & \text{else.} \end{cases} \quad (5.1)$$

where $\mathcal{P}(n, l, w) = (1-w)^l w^{n-l} \binom{n}{l}$ is the probability of having exactly l signatures among these n signatures with 1 as its first bit.

Proof. We will prove this by double induction on k and n . The base case is clear. If key length is 0, that means no partitioning is needed. Thus we have at exactly $\lceil n/p \rceil$ pages.

The induction case: Now assume the function is correct for any collection of signatures with key length less than k . Let's study the case when the key length is k . For this case, we do induction on n . If $n \leq p$, then by the insertion operation described in [31], the trie will consist of at most one page at the root of the trie. Thus, we have $\lceil n/p \rceil = 1$ page generated. Otherwise, the trie will branch into left and right sub-branches and signatures are hashed into the left branch or right branch depending on whether the first bit is 0 or not. The left and right branches behave exactly as the original trie except they partition signatures according to their second bit. Their key lengths are $k-1$ and signature lengths are $m-1$ (minus the first bit). If l signatures out of those n signatures are hashed into the left sub-trie, the expected number of pages generated in this branch will be $P_1(l, k-1, m-1, w)$. The remaining $n-l$ signatures are hashed into right branch, this generates on average $P_1(n-l, k-1, m-1, w)$ pages by our induction hypothesis. A simple calculation tells us that the probability that l out of n signatures have first bit set to 0 is $(1-w)^l w^{n-l} \binom{n}{l}$. The weighted sum gives us the expected number of pages generated

$$\sum_{l=0}^n (1-w)^l w^{n-l} \binom{n}{l} (P_1(l, k-1, m-1, w) + P_1(n-l, k-1, m-1, w)). \quad \square$$

For variable-prefix partitioning, the key length is always the same as the signature length, that is $k = m$. But we cannot conclude that $P_1(n, m, m, w)$ is the number of pages generated. The base case, $m = 0$, makes no sense here since there is no zero length signature. For the variable-prefix method, the induction is on signature length m and it starts at 1.

Corollary 2 *Under the same assumption in Lemma 11, the expected number of pages generated for the variable-prefix partitioning, can be calculated as*

$$P_2(n, m, w) = \begin{cases} \lceil n/p \rceil & \text{if } m = 1. \\ \lceil n/p \rceil & \text{if } n \leq p. \\ \sum_{l=0}^n (P_2(l, m-1, w) + P_2(n-l, m-1, w)) \mathcal{P}(n, l, w) & \text{else.} \end{cases} \quad (5.2)$$

Proof. The proof is similar to that of Lemma 11. When the key length is one, there are at most two signatures ($n \leq 2$). The conclusion is clearly true, so is the case for $n \leq p$.

When we have more than p signatures and $m > 1$, splitting is going to occur. The signatures are hashed into the left or right branches. Using the same reasoning as in Lemma 11, we have

$$P_2(n, m, w) = \sum_{l=0}^n (P_2(l, m-1, w) + P_2(n-l, m-1, w)) \mathcal{P}(n, l, w) \quad (5.3)$$

This completes the else case and the proof. $\square$

Another important performance measure is the search space reduction. The response time is mainly determined by the percentage of signatures searched since that percentage determines the number of disk accesses. Given a query signature S_Q , the search process as described in [31], starts with the root of the trie as follows.

1. If the first bit is zero, both the left and right branches have to be searched: otherwise, only the right child will be searched.
2. Repeat the previous step for each bit of S_Q until a leaf page is reached.
3. The leaf pages reached from the previous step contain qualified signatures and should be retrieved for further comparisons.

We assume that the query signature S_Q is generated with bit weight w_Q . In other words, the probability that a bit in S_Q is set is w_Q , which is also referred to as query signature weight, or query weight for short. The subscript Q is added to distinguish it from the bit weight of signatures in the database. Analyzing the above search process, we formulate a recursive function for estimating the number of pages searched for the fixed-prefix method.

Lemma 12 *Suppose we have a signature collection $\mathcal{S}(n, m, w)$ and signatures are partitioned into a trie structure with fixed-prefix key of k . Given a query signature S_Q of query weight w_Q , the expected number of pages to be searched, $R_1(w_Q, k, n, m, w)$, can be calculated as:*

$$R_1(w_Q, k, n, m, w) = \begin{cases} \lceil n/p \rceil & \text{if } k = 0. \\ \lceil n/p \rceil & \text{if } n \leq p. \\ \sum_{l=0}^n \mathcal{P}(n, l, w) (R_1(w_Q, k-1, n-l, m-1, w) & \text{else.} \\ \quad + (1-w_Q)R_1(w_Q, k-1, l, m-1, w)) & \end{cases}$$

Proof. We prove it by double induction on k and n . For the base case that k is 0, it is clear. There is no partitioning, all signatures are stored in a linked list of pages. We have to search all pages to find qualified signatures. Now let's assume that our conclusion is true for all key signature lengths less than k . We will prove the case where the key length is exactly k . The proof will be an induction on n . If we have no more than p signatures, we have $\lceil n/p \rceil = 0$ or 1, and we clearly have to search it. If we have more than p signatures, the search will be different since we have two branches at every stage. From the search process described before, we have to search only the right branch if the splitting bit of the query signature is 1, and both branches

if it is 0. Let's assume that l signatures are hashed into the left and $n - l$ to the right. By the induction hypothesis, we have to search $R_1(w_Q, k - 1, l, m - 1, w)$ pages if we search the left branch, and $R_1(w_Q, k - 1, n - l, m - 1, w)$ pages in the right branch to get all qualified signatures.

Since the query signature has weight w_Q , each bit of the query signature has probability of w_Q to be set to 1. Thus the probability to search both branches is $(1 - w_Q)$ and only the right branch is w_Q . The expected number of pages searched is

$$w_Q R_1(w_Q, k - 1, l, m - 1, w) + (1 - w_Q)(R_1(w_Q, k - 1, l, m - 1, w) + R_1(w_Q, k - 1, n - l, m - 1, w))$$

which is equivalent to

$$R_1(w_Q, k - 1, l, m - 1, w) + (1 - w_Q)R_1(w_Q, k - 1, n - l, m - 1, w).$$

Summing over all the possibilities weighted by their occurring probabilities, we get the number of pages that have to be searched. $\square$

Generalizing this result to variable prefix partitioning, we have

Corollary 3 *Suppose we have a query signature S_Q with query weight w_Q , then for the variable-prefix partitioning, the expected number of pages searched can be calculated as:*

$$R_2(w_Q, n, m, w) = \begin{cases} \lceil n/p \rceil & \text{if } m = 1. \\ \lceil n/p \rceil & \text{if } n \leq p, \\ \sum_{l=0}^n P(n, l, w)(R_2(w_Q, n - l, m - 1, w) + (1 - w_Q)R_2(w_Q, l, m - 1, w)) & \text{else.} \end{cases}$$

Proof. In variable-prefix partitioning, we split and create new pages whenever it is necessary. The key length is the same as the signature length. The base cases, $m = 1$

and $n \leq p$, are obvious and similar the base case of Lemma 2. The reasoning for case $n > p$ is similar to the case of $n > p$ for Lemma 12. $\square$

The technique used in the proof of Lemma 12 can be used to derive many useful analytical results such as page fill factor, the number of signatures searched and the length of the longest path length etc. We list several here. Their proofs are omitted to avoid repetition.

For the fixed-prefix partitioning:

1. The number of signatures to be searched, $S_1(w_Q, k, n, m, w)$ is

$$S_1 = \begin{cases} n & \text{if } k = 0. \\ n & \text{if } n \leq p. \\ \sum_{l=0}^n \mathcal{P}(n, l, w)(S_1(w_Q, k-1, n-l, m-1, w) & \text{else.} \\ \quad + (1-w_Q)S_1(w_Q, k-1, l, m-1, w) & \end{cases} \quad (5.4)$$

2. The longest path generated during signature insertion, $L_1(k, n, m, w)$, is

$$L_1(k, n, m, w) = \begin{cases} \lceil n/p \rceil & \text{if } k = 0. \\ \lceil n/p \rceil & \text{if } n \leq p. \\ \sum_{l=0}^n \mathcal{P}(n, l, w) \text{Max}_{t=l, n-l} \{L_1(k-1, t, m-1, w)\} & \text{else.} \end{cases} \quad (5.5)$$

For variable prefixed partitioning:

1. The expected number of signatures to be searched, $S_2(w_Q, k, n, m, w)$, is

$$S_2(w_Q, k, n, m, w) = \begin{cases} n & \text{if } m = 1. \\ n & \text{if } n \leq p. \\ \sum_{l=0}^n \mathcal{P}(n, l, w)(S_2(w_Q, k-1, n-l, m-1, w) & \text{else.} \\ \quad + (1-w_Q)S_2(w_Q, k-1, l, m-1, w) & \end{cases} \quad (5.6)$$

2. The longest path generated during signature insertion, $L_2(k, n, m, w)$, is

$$L_1(k, n, m, w) = \begin{cases} \lceil n/p \rceil & \text{if } m = 1, \\ \lceil n/p \rceil & \text{if } n \leq p, \\ \sum_{l=0}^n \mathcal{P}(n, l, w) \cdot \text{Max}_{t=l, n-l} \{L_2(k-1, t, m-1, w)\} & \text{else. (5.7)} \end{cases}$$

Computational Complexity

The number of signatures n in a database can easily reach tens of millions. It is important for our formulas to be inexpensive to compute. In this section, we will try to analyze their complexities. Approximating formulas, which are easy to compute and still give excellent performance, will be derived for functions with high complexities.

It is clear that there are several contributing factors to the complexities of these functions. However, in this dissertation, computational complexities will be presented in terms of n , the number of signatures in the database. Other parameters, such as signature length m and key length k etc., are either fixed or bounded by fixed numbers in any database. The signature count, n , dominates the complexity when it is large.

It is clear these functions are similar in structure. We will focus on the computation of $P_1(n, k, m, w)$ in Lemma 11. The result obtained here applies to all other functions. For simplicity, we put our priority on improving time complexity. Memory constraint is secondary in our analysis.

Claim 1. The computational time complexity of $P_1(n, k, m, w)$ is $O(n^2)$ with space overhead $O(n)$.

Proof. From Lemma 11, function $P_1(n, k, m, w)$ is clearly a recursive function. The value of $P_1(n, k, m, w)$ depends on values of $P_1(i, k-1, m-1, w)$, $i = 0, \dots, n$ and so on. Since w, p are fixed throughout the computation, let's abbreviate $P(n, k, m)$

as $P_1(n, k, m, w)$. The computation of $P(n, k, m)$ can be visualized as a matrix:

$$\begin{pmatrix} 0 & P(0, 0, m - k) & P(1, 0, m - k) & \cdots & P(n, 0, m - k) \\ 1 & P(0, 1, m - k + 1) & P(1, 1, m - k + 1) & \cdots & P(n, 1, m - k + 1) \\ \dots & \dots & \dots & \dots & \dots \\ k - 2 & P(0, k - 2, m - 2) & P(1, k - 2, m - 2) & \cdots & P(n, k - 2, m - 2) \\ k - 1 & P(0, k - 1, m - 1) & P(1, k - 1, m - 1) & \cdots & P(n, k - 1, m - 1) \\ k & & & & P(n, k, m) \end{pmatrix} \quad (5.8)$$

The first column is the key length. The computation proceeds from row 0 to row $k - 1$ and then to $p(n, k, m)$. Each entry in row zero, which corresponds to $k = 0$, takes constant steps to compute from the base case of $P(n, k, m)$. After we have calculated every entries in row $i - 1$, let's see how to compute entries in row i .

For each entry $P(j, i, t)$, $0 < i < k$, in row 1 to row $k - 1$, the computation of $P(j, i, t)$ takes j steps since

$$P(j, i, l) = \sum_{l=0}^j \mathcal{P}(j, l, w) (P(l, i - 1, m - k + i - 1) + P(j - l, i - 1, m - k + i - 1)).$$

Thus the computation of all entries in row i takes $O(\sum_{l=0}^n l) = O(n^2)$ steps. and the computation of $P(n, k, m)$ takes $O(kn^2) = O(n^2)$ steps. We have to save all results on row $i - 1$ in order to compute entries on row i . The space complexity $O(n)$ follows.

□

The $O(n^2)$ time complexity is expensive when n is large, which is typical for signature files. We now try to simplify the algorithm and find a linear solution. From Lemma 11, we know for large $n > p$, $P_1(n, k, m, w)$ is

$$\sum_{l=0}^n \mathcal{P}(n, l, w) (P_1(l, k - 1, m - 1, w) + P_1(n - l, k - 1, m - 1, w))$$

where $\mathcal{P}(n, l, w) = (1 - w)^l w^{n-l} \binom{n}{l}$ is the probability that there are l out of n signatures having corresponding bit set to 0. For large n , small l and large l represent the situations when we have extremely unbalanced trie structures, which would rarely

happen, when signatures are generated uniformly. Intuitively, omitting terms corresponding to large or small l from the sum will not change the result in any noticeable way. This is the main idea for our approximation. But the difficult part is to decide which terms can be left out. We need some statistical results here.

Given n signatures, let X_n denote the number of signatures whose first bit are 0. Then X_n has binomial distribution $B(n, w)$ with mean nw and variance $nw(1 - w)$. According to *The Central Limit Theorem* [2], $\frac{X_n - nw}{\sqrt{nw(1 - w)}}$ converges to the standard normal distribution $N(0, 1)$ when n is large. Mathematically it is

$$\lim_{n \rightarrow \infty} \mathcal{P}\left\{\left|\frac{X_n - nw}{\sqrt{nw(1 - w)}} - z\right| \geq \varepsilon\right\} = 0$$

where $z \sim N(0, 1)$ is the standard normal distribution. Thus for large n ,

$$\mathcal{P}\left\{\left|\frac{X_n - nw}{\sqrt{nw(1 - w)}}\right| \geq z_{1-\frac{\alpha}{2}}\right\} = \alpha$$

where $z_{1-\frac{\alpha}{2}}$ is tabulated in many statistics books. Followings are a few entries from the standard normal distribution table in [26].

α	0.05	0.025	0.01	0.005	0.002	0.001	0.0004
$z_{1-\frac{\alpha}{2}}$	1.96	2.24	2.57	2.81	3.09	3.27	3.49

A rule of thumb [2] is that for most purposes the approximation can be used when nw and $n(1 - w)$ are both larger than 5. From the table, we can see that

$$\mathcal{P}\left\{\left|\frac{X_n - nw}{\sqrt{nw(1 - w)}}\right| \geq 3.09\right\} = 0.002,$$

which means

$$nw - 3.09\sqrt{nw(1 - w)} \leq X_n \leq nw + 3.09\sqrt{nw(1 - w)}$$

with probability 0.998. Dropping the l 's which are far away from mean nw , we have the approximation formula $\bar{P}_1(n, k, m, w)$.

Lemma 13 *The number of expected pages generated, $\bar{P}_1(n, k, m, w)$, can be approximated as*

$$P_1(n, k, m, w) \quad \text{for } n \leq 5\left(\frac{1}{w} + \frac{1}{1-w}\right) + p.$$

and for $n > 5\left(\frac{1}{w} + \frac{1}{1-w}\right) + p$

$$\sum_{l=nw-\beta\sqrt{nw(1-w)}}^{nw-\beta\sqrt{nw(1-w)}} \mathcal{P}(n, l, w) (\bar{P}_1(l, k-1, m-1, w) + \bar{P}_1(n-l, k-1, m-1, w)) \quad (5.9)$$

where β is a constant depending on the accuracy requirement. It can be found in the normal distribution table.

Proof. The conclusion follows from the *Central Limit Theorem* and facts we presented before. The number $5\left(\frac{1}{w} + \frac{1}{1-w}\right) + p$ is used as the splitting point since normal distribution table works well when $n > 5\left(\frac{1}{w} + \frac{1}{1-w}\right)$. We add p to the lower bound of n since we want at least p signatures in the database. $\square$

The time complexity of this function is reduced drastically. For the two main summations,

$$\sum_{l=nw-\beta\sqrt{nw(1-w)}}^{nw-\beta\sqrt{nw(1-w)}} \mathcal{P}(n, l, w) (\bar{P}_1(l, k-1, m, w) + \bar{P}_1(n-l, k-1, m, w))$$

sums over about 3.090 terms and

$$\sum_{l=0}^n \mathcal{P}(n, l, w) (P_1(l, k-1, m-1, w) + P_1(n-l, k-1, m-1, w))$$

sums over 1000,000 terms when $n = 1000,000$, $w = 0.5$ and $\beta = 3.09$. In fact, we have the following result:

Claim 2. The computational time complexity of $\bar{P}_1(n, k, m, w)$ is $O(n)$.

Proof. Again we use $P(n, k, m)$ as a shorthand for $\bar{P}_1(n, k, m, w)$. We will refer terms on row j , $j = 0, \dots, k$, as level j terms in matrix 5.8. From Lemma 13, we know that

$$P(n, j+1, m) = \sum_{l=nw-3\sqrt{nw(1-w)}}^{nw-3\sqrt{nw(1-w)}} \mathcal{P}(n, l, w)(P(l, j, m-1) + P(n-l, j, m-1))$$

For each term $P(n, j+1, m)$ at level $j+1$, we need to compute several terms at level j . The number of terms needed at level j is

$$(nw + 3\sqrt{nw(1-w)}) - (nw - 3\sqrt{nw(1-w)}) = 2\beta\sqrt{nw(1-w)} = O(\sqrt{n}). \quad (5.10)$$

We now estimate how many terms need to be computed at each level. Let's use l_j and h_j to denote indices of the first and last terms that need to be computed in level j . Value $h_j - l_j + 1$ gives us the number of terms that need to be calculated at level j . It is clear $h_k = l_k = n$ and only one term, $P(n, k, m)$, that need to be computed at level- k . And it is also clear that $2\beta\sqrt{nw(1-w)} = O(\sqrt{n})$ many level $k-1$ terms need to be computed for $P(n, k, m)$.

Assume we have l_{j+1} and h_{j+1} , let's derive l_j and h_j . From function 5.9 in Lemma 13, the left-most and right-most terms we need to compute on level j are indexed by $wl_{j+1} - 3\sqrt{l_{j+1}w(1-w)}$ and $wh_{j+1} + \beta\sqrt{h_{j+1}w(1-w)}$ respectively. This implies that

$$l_j = wl_{j+1} - \beta\sqrt{l_{j+1}w(1-w)} \quad (5.11)$$

$$h_j = wh_{j+1} + \beta\sqrt{h_{j+1}w(1-w)} \quad (5.12)$$

So we need to compute

$$h_j - l_j = w(h_{j+1} - l_{j+1}) + 2\beta\sqrt{w(1-w)}(\sqrt{h_{j+1}} + \sqrt{l_{j+1}})$$

level- j terms. By simple induction, we can show that for $j = k-1, k-2, \dots, 0$,

1. $h_j - l_j = O(\sqrt{n})$ and
2. $h_j = l_j = O(n)$.

This means we have $O(\sqrt{n})$ terms at each level and for each term, it takes no more than $O(\sqrt{n})$ steps to compute. Combining these facts together, we conclude the complexity is $O(n)$. In fact, the above discussion also leads to the conclusion that space complexity is $O(\sqrt{n})$. $\square$

From this Claim, we can see the speed up is in the magnitude of n . In our simulation, the computation of expected number of pages generated is greatly improved. We will see these approximation formulas are also very accurate.

Similarly we can get efficient approximating formulas for other performance measures. We list several of them here. Their proofs are similar to the proof of Lemma 13. We omit them to avoid repetition.

Corollary 4 *Under the same assumption, the number of expected pages generated for the variable-prefix partitioning, $P_2(n, m, w)$, can be approximated by $\bar{P}_2(n, m, w)$, which is*

$$P_2(n, m, w) \quad \text{for } n \leq \tilde{5}\left(\frac{1}{w} + \frac{1}{1-w}\right) + p,$$

and

$$\sum_{l=nw-\beta\sqrt{nw(1-w)}}^{nw-\beta\sqrt{nw(1-w)}} \mathcal{P}(n, l, w)(\bar{P}_2(l, m-1, w) + \bar{P}_2(n-l, m-1, w))$$

for $n > \tilde{5}\left(\frac{1}{w} + \frac{1}{1-w}\right) + p$.

Corollary 5 Suppose we have a query signature S_Q with weight w_Q , the expected number of pages searched for the variable-prefix partitioning, $R_2(w_Q, n, m, w)$, can be approximated by $\bar{R}_2(w_Q, n, m, w)$, which is

$$R_2(w_Q, n, m, w) \quad \text{for } n \leq 5\left(\frac{1}{w} + \frac{1}{1-w}\right) + p,$$

and

$$\sum_{l=nw-3\sqrt{nw(1-w)}}^{nw-3\sqrt{nw(1-w)}} \mathcal{P}(n, l, w) B(l, m, w, w_Q).$$

where

$$B(l, m, w, w_Q) = (1 - w_Q) \bar{R}_2(w_Q, l, m - 1, w) + \bar{R}_2(w_Q, n - l, m - 1, w)$$

$$\text{for } n > 5\left(\frac{1}{w} + \frac{1}{1-w}\right) + p.$$

To see how accurate our approximation formulas are, we did simulations with the variable-prefix partitioning method with signature of length of 512. Data was collected for the number of pages generated and number of pages searched for query signatures with different query weights. The number of signatures ranges from 20,000 to 250,000. Simulation results are compared with results obtained by theoretical estimations. The results are plotted in Figure 5.1 for number of pages generated and Figure 5.2 for percentage of pages searched when query signature weight $w_Q = 0.1172$. We only plot the page search reduction ratio here since signature search reduction ratio is almost identical to page search reduction ratio [30, 31].

From Figure 5.1 and Figure 5.2, we can see results from the approximation formulas match simulation results very accurately. When the number of signatures reaches 250,000, the error for number of pages generated is under 0.3%. Error for the percentage of pages searched is about 4% for number of signatures less than 250,000 and

Figure 5.1: Pages generated (Variable-prefix)

Figure 5.2: Pages searched (Variable-prefix)

less than 1% when signature number reaches 250,000. The approximation formulas are derived from statistical results. We can expect the error will be narrowed furthermore with more signatures in the database. When the number of signatures is small, the simulation results can be perturbed by sampling anomaly. This is evident in Figure 5.2. When the number of signatures is relatively small, the number of pages generated, therefor the number of pages searched, can be changed drastically with just a few new signatures. In the worst case scenario, each new signature can cause a new page to be generated. This effect becomes less visible when the number of signatures is big.

Search Reduction Ratio of the Variable-prefix Partitioning Method

We propose some modifications to the fixed-prefix method. In this section, we want to see how the new partition method stacks up to the performance of the fixed-prefix partitioning. In our simulation, we use signature length $m = 512$ and page size $p = 16$. Signature reduction ratio for the fixed-prefix method when $k = 15$ is obtained from [30]. Figure 5.3 shows the signature search reduction ratios of both the fixed-prefix partitioning and the variable-prefix partitioning. It is clear that the variable-prefix method performs better than the fixed-prefix method for large databases. The margin gets bigger as the number of signatures in the database increases. The reduction ratio is improved by additional 10% when the number of signature reaches 16 millions.

This is no surprise given the way that the variable-prefix partitioning refines the fixed-prefix partitioning. Fixed-prefix key partitioning can generate at most 2^k partitions and stores signatures with identical key in one partition. When a partition has too many signatures, the search reduction ratio for the fixed-prefix method suffers

Figure 5.3: Signature search reduction ratio

since we have to search the whole partition if the partition key qualifies. Whereas for the variable-prefix method, no partition with more than p signatures is generated. Key matching algorithm eliminates more signatures than that of the fixed-prefix method.

5.2 Floating Key Partitioning

Lee and Leng [30, 31] showed that the floating-key method has the best search reduction ratio. In this section, we show how the formulas obtained for the variable-prefix method can be used to estimate the number of pages generated and the search space reduction ratio for the floating-key partitioning method.

Unlike in the variable-prefix partitioning method, we will assume that all our signatures are generated optimally. That is, for each bit position, the probability that the bit set to 1 is 0.5 for all signatures during signature generation. The parameter w is dropped from our analysis.

To better present our result, we now introduce some notations. We assume that the signature length is m , key length is k . As described in Lee and Leng's paper, signatures are divided into m_k segments, where $m_k = \lceil m/k \rceil$. The key segments will start at $0, k, 2k, \dots, (m_k - 1) * k$. The first $m_k - 1$ segments all have length k and the last one consists of $m - m * m_k = m \% k$ bits. The last segment, with length less than k or equal to 0, will not participate in the key selection process. These key segments will be named sequentially as $S_0^k, S_1^k, \dots, S_{m_k-1}^k$ according to their starting bit positions. For any signature S , the floating key of S has to be one of S_j^k , $0 \leq j < m_k$. We will refer position $0, k, 2k, \dots, (m_k - 1)k$ or even $0, 1, 2, m_k - 1$ as key positions if no confusion arises. For any key position $j * k$, let us use $\mathcal{S}(j)$ to denote the collection of all signatures with key position starting at $j * k$.

In the floating-key partitioning, all signatures with the same key position $j * k$ are grouped together and then partitioned into a trie exactly as in the fixed-prefix partitioning. The difference is that the floating key segment starting at $j * k$ is used as partitioning key instead of the prefix of the signature. For each $\mathcal{S}(j)$, the probability that a bit position in the key segment is set to 1 is no longer 0.5. This is the reason we assume the bit weight is not necessarily 0.5 when working on variable prefix partitioning. We can use results obtained for the variable-prefix partitioning method to estimate the number of pages generated, signature search reduction ratio etc. for an individual trie. To do so, we have to obtain the expected number of signatures partitioned into $\mathcal{S}(j)$, and the probability with which bits in the key are set to 1.

The first segment with the least number of 1's is selected as the key. There is no question that the selection process has an effect on the structure of each trie, though

the effect differs for different key positions. Apparently, not all tries in the floating key partitioning will have the same size. For each bit within the key segment starting at $j * k$, the probability that this bit is set to 1 is not 0.5 for signatures in $\mathcal{S}(j)$. We can expect that the probability to be smaller than 0.5 due to the way this segment is selected. In the variable-prefix partitioning analysis, we know this probability affects the generated trie balance and many performance measures.

In [30], Lee and Leng obtained an analytical formula, $\mathcal{P}(j, i)$, for the probability that a signature is hashed into $\mathcal{S}(j)$ and whose key segment has exactly i bits set to 1.

Lemma 14 *Given a random signature S , the probability that S belongs to $\mathcal{S}(j)$ is*

$$\mathcal{P}(j) = \sum_{i=0}^k \mathcal{P}(j, i) \quad (5.13)$$

Thus the expected size of $\mathcal{S}(j)$ is $\mathcal{P}(j) * n$ if there are n signatures in the database. Furthermore, the probability of a bit in the key segment set to 1 is given by

$$w(j) = \frac{1}{k\mathcal{P}(j)} \sum_{i=0}^k \mathcal{P}(j, i)i \quad (5.14)$$

Proof. We know for any signature S and integer $0 \leq i \leq k$, the probability that S_j^k is the key and S_j^k has exactly i bits set to 1 is $\mathcal{P}(j, i)$. So the sum $\sum_{i=0}^k \mathcal{P}(j, i)$ gives us the probability that segment S_j^k is the key. This is the same thing as S belongs to $\mathcal{S}(j)$. The expected size of $\mathcal{S}(j)$ follows.

For a random signature S with key segment S_j^k , the probability that S_j^k has i bits set is $\frac{\mathcal{P}(j, i)}{\mathcal{P}(j)}$. The expected number of bits set to 1 in the key segment is therefor the weighted sum

$$\sum_{i=0}^k \frac{\mathcal{P}(j, i)}{\mathcal{P}(j)} i = \frac{1}{\mathcal{P}(j)} \sum_{i=0}^k \mathcal{P}(j, i)i$$

Dividing this sum by segment length k , we have $w(j)$ which determines the balance of trie at key position $j * k$. $\square$

We list two interesting facts here without proofs since they are not essential to our main topic.

1. $\mathcal{P}(j)$ is strictly decreasing.
2. $w(j)$ is strictly decreasing and always less than 0.5.

This means that tries at small key positions receive more signatures, and hence are larger than tries at late key positions and tries at early key positions have more balanced tree structures.

We now have everything needed to study various performance measures about floating-key partitioning. We will first study the number of pages generated and then estimate the search space reduction ratio. One thing worth pointing out is that the variable-prefix partitioning method is just a special case of the floating-key partitioning method. They are equivalent when key segment length $k > m/2$.

Lemma 15 *Suppose we have n signatures of length m and the key length is k . If we partition those signatures according to float-key partitioning, $P_3(n, k, m)$, the expected number of pages generated can be calculated as*

$$\sum_{j=0}^{m_k-1} P_1(\mathcal{P}(j) * n, k, m, w(j)) \quad (5.15)$$

where P_1 is the formula we used to estimate the number of pages generated for the fixed-prefix method.

Proof. For any key position j , we know the expected number of signatures fall in trie j is $\mathcal{P}(j) * n$. Inside each trie, the variable prefix of the signatures is used to partition

incoming signatures. Thus we can use the results obtained for the variable-prefix partitioning. From Lemma 11, the expected number of pages generated in trie j is $P_1(\mathcal{P}(j) * n, k, m, w(j))$ since the bit weight for key segment j is $w(j)$. Summing over all possible tries, we have the desired result. $\square$

Similarly we have formulas for the number of pages searched and the number of signatures searched.

1. With the same assumption as in Lemma 15, if we search the database with a signature with query weight w_Q , the expected number of pages searched, $R_3(w_Q, n, k, m)$, is

$$\sum_{j=0}^{m_k-1} R_1(w_Q, k, \mathcal{P}(j) * n, m, w(j)) \quad (5.16)$$

where $R_1(w_Q, k, n_j, m, w)$ is the formula we obtained for the fixed-prefix partitioning.

2. With the same assumption as in Lemma 15, the expected number of signatures searched for a signature of query weight w_Q , $S_3(w_Q, n, k, m)$, is

$$\sum_{j=0}^{m_k-1} S_1(w_Q, k, \mathcal{P}(j) * n, m, w(j)) \quad (5.17)$$

where $S_1(w_Q, k, n_j, m, w)$ is the formula we obtained for the fixed-prefix partitioning.

Their proofs are similar to the proof for expected number of pages generated.

We performed simulations with float key partitioning and compared the results with those obtained from the theoretical approximation formulas. The simulations range from 20,000 to 500,000 signatures. So we can see how the theoretical results measure up to the simulation results on a wide spectrum. We use $m = 512$, $k = 32$

Figure 5.4: Number of pages generated for $m = 512$ and $k = 32$

Figure 5.5: Percentage of signatures searched for $m = 512$, $k = 32$ and $w_Q = 0.1172$

in our simulation. The results are plotted in Figure 5.4 for the number of pages generated and Figure 5.5 for the number of signatures searched.

For the number of pages generated, they match almost exactly. For search reduction ratio, we did simulation with query signature weight $w_Q = 0.1172$. For each simulation, we randomly generated 1000 signatures with weight 0.1172 ($= 60/512$) and searched them through all generated tries. The average number of signatures searched is used as the number of signatures searched. Again they match pretty nicely. Actually there is no noticeable difference for page size 64 and the difference for page 32 is always smaller than 2% and less than 0.2% when number of the number of signatures reaches 500.0000. Since we use the variable-prefix partitioning method in each individual trie, the improvement of the new floating-key partitioning method over the old one is the same as the improvement achieved by the variable-prefix partitioning method over the fixed-prefix partitioning method.

These formulas are not only accurate but also very efficient. In fact, they are linear in terms of n .

Lemma 16 *The time complexities of $P_3(n, k, m)$, $S_3(n, k, m)$ and $R_3(w_Q, n, k, m)$ are $O(n)$.*

Proof. The computation of $P_3(\mathcal{P}(j) * n, k, m, w(j))$ takes $O(\mathcal{P}(j) * n)$ steps according to the results we obtained for the variable-prefix partitioning method. Thus, the total complexity of $P_3(n, k, m)$ is

$$\sum_{j=0}^{m_k-1} O(\mathcal{P}(j) * n),$$

which is $O((\sum_{j=0}^{m_k-1} \mathcal{P}(j)) * n) = O(n)$. The proofs for other functions are similar. $\square$

5.3 Storage Saving

In the three partition schemas proposed in [30, 31], all signatures with the same key are grouped into one partition. The key portions of the signatures need not be stored for every signature. There will be some storage saving by storing only the non-key part of a signature inside a partition and storing just one copy of the partition key. Any saving here comes absolutely free since it does not slow down the retrieval process. This kind of storage saving does not occur only in the three partitioning methods developed by Lee and Leng. It happens to other key based partitioning methods, like *Quick Filter* [41] and Parametric Weighted Filter [4], as well.

It is interesting to know how much space is saved by storing only one copy of the key. Finding the answer, surprisingly, turns out to be non-trivial. In this section, we will estimate the saving for the fixed-prefix partitioning. The result should give us a good indication about storage savings about other methods, including *Quick Filter* [41] and Parametric Weighted Filter [4]. For simplicity, we only consider the case where all partitions are generated. Thus the storage overhead is the storage of the partition keys, $2^k k$, plus the storage overhead of the non-key part of signatures, $(m - k)n$. These two terms together give us the new storage overhead. In other words, the storage overhead can be written as

$$f(m, k, n) = (m - k)n + 2^k k.$$

With f as the space overhead used by the fixed-prefix partitioning, the storage saving ratio (SSR) is

$$SSR(m, k, n) = 1 - \frac{f(m, k, n)}{mn} = \frac{mn - ((m - k)n + 2^k k)}{mn} = \frac{nk - 2^k k}{mn}$$

We now study this function and find value of k such that $SSR(m, k, n)$ is maximal or near maximal, which, in return, tells us how much space can be saved.

Lemma 17 *For the fixed-prefix key partition method with parameters m, n , the key length*

$$k = \frac{\ln n - \ln(1 + \ln n)}{\ln 2}$$

gives the maximal storage saving. And the saving ratio can be approximated by

$$\frac{\ln n}{m(1 + \ln n) \ln 2} \ln \frac{n}{1 + \ln n}$$

Proof. As a function of k with parameters m and n , $S(m, k, n)$ is a smooth function in a closed interval $[0, m]$. The existence of the maximal point is clear. Furthermore it is easy to see neither of the boundary points, $k = 0, m$, can be the maximal point. We can find the maximal point by finding the critical point(s) of $S(m, k, n)$. Taking the derivative of $S(m, k, n)$, we have

$$\frac{\partial}{\partial k} SSR(n, m, k) = \frac{n - 2^k - k2^k \ln 2}{mn}$$

Setting the derivative to zero, we get

$$2^k(1 + k \ln 2) = n \tag{5.18}$$

The left hand side of equation (5.18) is clearly a strictly increasing function of key length k . This means we have exactly one critical point, which must be the optimal key length.

Solving for k in equation (5.18), we can see that k must satisfy

$$k = \frac{\ln n - \ln(1 + k \ln 2)}{\ln 2} \tag{5.19}$$

Iterating equation (5.19) repeatedly, we have

$$\begin{aligned}
k &= \frac{\ln n - \ln(1 + k \ln 2)}{\ln 2} \\
&= \frac{\ln n - \ln(1 + \ln n - \ln(1 + k \ln 2))}{\ln 2} \\
&= \frac{\ln n - \ln(1 + \ln n - \ln(1 + \ln n - \ln(1 + k \ln 2)))}{\ln 2}
\end{aligned} \tag{5.20}$$

Line two and three of equation (5.20) imply that

$$k \geq \frac{\ln n - \ln(1 + \ln n)}{\ln 2} \tag{5.21}$$

$$k \leq \frac{\ln n - \ln(1 + \ln n - \ln(1 + \ln n))}{\ln 2} \tag{5.22}$$

respectively. This means that the optimal key length k must be in the interval

$$\left[\frac{\ln n - \ln(1 + \ln n)}{\ln 2}, \frac{\ln n - \ln(1 + \ln n - \ln(1 + \ln n))}{\ln 2} \right] \tag{5.23}$$

The length of this interval is very small. For example, the interval length is about 0.152812 for one billion signatures ($n = 1000,000,000$). It is safe to use $k = \lfloor \frac{\ln n - \ln(1 + \ln n)}{\ln 2} + 0.5 \rfloor$ or even $k = \lfloor \frac{\ln n - \ln(1 + \ln n)}{\ln 2} \rfloor$, as the optimal key length. This value is accurate enough for most applications. Using $k = \frac{\ln n - \ln(1 + \ln n)}{\ln 2}$, we obtain

$$SSR(n, m, k) = \frac{\ln n}{m(1 + \ln n) \ln 2} \ln \frac{n}{1 + \ln n} \tag{5.24}$$

with some algebraic operation. $\square$

The optimal key length $k = \frac{\ln n - \ln(1 + \ln n)}{\ln 2}$ is actually $\log_2 n - \log_2(1 + \ln n)$, which is very close $\log_2 n$. The offset is $\log_2(1 + \ln n)$, a small number relative to $\log_2 n$. This implies that the optimal key length is very close to logarithm function of the signature count. It is easy to see that this saving ratio is an increasing function of the number of signatures. For a fixed signature length, more signatures means more saving. The maximal saving will be reached when the database contains all possible signatures.

Figure 5.6: Maximal space saving by not storing Keys

Figure 5.7: Space saved by not storing keys

Figure 5.8: Optimal key length

The savings gained by not storing unnecessary keys are plotted in Figure 5.3. From these figures, we can see that maximal space saving is quite substantial. The maximal possible saving reaches 98% when signature length reaches 1000. However, the actual saving in the second diagram, Figure 5.6, shows a totally different outlook. For signature lengths 256, 512 and 1024, the savings are less than 10% even when there are over one billion signatures in the databases.

The reason behind these seemingly contradicting results is we have much fewer signatures than the maximal possible signature count. The high percentage saving can only be reached when the signature file is full or nearly full. For example, there are 2^{256} possible signatures for $m = 256$, which is far too large for any real database. The maximal possible saving cannot be materialized in real world application. But no matter how small the saving is, it comes absolutely free.

For completeness, we also plot a diagram to show the optimal key length for different signature count. Just as predicted, the optimal key length displays pattern similar to a logarithm function.

CHAPTER 6

Conclusion and Future Work

6.1 Summary and Conclusion

This thesis presents a general technique, the weight-partitioned signature files, for implementing the vector space statistical model with signature files. In weight-partitioned signature files, terms in a document are grouped according to their term frequencies and hashed as a group into the signature file with the corresponding term frequency. Thus, when a match is found, the signature file in which the match occurs will indicate the term frequency of the matching term. This avoids storing the term frequency of each term explicitly.

We also present an analytical study of the performance of two search strategies, namely, the LH and HL methods, and obtained an optimal way of assigning signature lengths, and thus storage, to the signature partitions to minimize the effect of false drops on precision and recall. Experiments were performed to support our analytical study.

We also study how to improve search response time by utilizing the unique structure of TF-partitioned signature files. The newly improved upperbound search algorithm achieves 40% to 70% search reduction ratio without sacrificing any precision and recall for top documents. We also experiment TF-partitioned signature files

method under low storage. For long document collection, TF-partitioned signature files can achieve comparable precision and recall by doing exact matching on the top documents with 10% of space overhead.

In the second part, we extend results obtained by Lee and Leng [30, 31]. A new way of organizing a signature file is presented to avoid the exhaustive search of all signatures. Instead of using fixed prefix key, our method uses variable prefix key. No link list is used at leaf nodes. New partitions (pages) are generated whenever an overflow occurs.

Simulation results show that the new approach, the variable-prefix partitioning, achieves better performance. The search space reduction ratio is improved by an additional 10% for large databases. Several very accurate and fast analytical heuristic functions are developed for estimating the number of pages generated and the search reduction ratio for the fixed-prefix, variable-prefix and floating-key partitioning methods. Their time complexities are linear in terms of the signature number. With these functions, database designers can have a better understanding of the issues without costly simulations.

Due to the nature of key partitioning methods, some storage saving can be achieved by only storing one copy of each partition key. We obtain formulas to calculate the maximal saving and the optimal key length.

We feel that the work presented in this thesis is a significant step toward a viable way of supporting document ranking with signature files. The signature file technique will become much more competitive in speed if it doesn't have to perform false drop elimination, and only a small portion of the signatures need to be searched.

6.2 Future Research

Fusion Strategy

The development of weight-partitioned signature files marks a significant improvement for signature file model. However, there are still many interesting and promising research areas that could be investigated to extend the current work.

The key-based partition methods developed by Lee and Leng [30] and several search algorithms presented in this thesis all lead to significant improvements in response time. Interestingly, Lee and Leng's methods [30, 31] and our new fast search algorithms are not mutually exclusive. We are investigating heuristics to integrate them together to improve search performance and reduce the effect of false drops. A possible scenario is described as follows.

We can do full text indexing on short documents and selective indexing (terms with $tf > 1$) on long documents. Then we partition each $SF_i, i < n$ using the floating key partitioning. The signature file size is reduced even before applying any search reduction algorithms.

When a query arrives, we process key-partitioned signature files $SF_i, i > 2$ first and establish a coarse ranking. Certain percentage of the signatures in $SF_i, i > 2$ will be excluded from searching by the floating key method in this phase.

Next, we select T^{th} document's score $S(T)$ and stop the search if the upperbound of $S(T+1)$ is less than $S(T)$. Otherwise, we apply the upperbound search algorithm on SF_2 , $S(T)$ can be updated if desired. In this step, we can expect a certain percentage of documents, not just individual signatures, to be excluded from further searching.

Finally, we repeat the above step for SF_1 and return the final top T documents to the user.

We expect these methods together will give a significant performance boost to signature file based retrieval systems. It certainly will narrow the gap between signature file and inverted file. Fast retrieval will make signature file approach more appealing since it uses significant less storage resource.

We are also investigating ways to apply the idea of our upperbound search algorithm back to inverted file. Instead of waiting for the magic stop point to end the whole search process, we should try to eliminate as many documents, postings lists as possible from further processing.

Parallel Text Processing

In this thesis, we did not explore weight-partitioned signature files in a parallel environment. With the falling price and increasing number of multi-processor machines, parallel text search will be more important and popular. The weight-partitioned signature files can readily utilize the power of parallel processing because of its unique structure. The most straight-forward way to implement HL search algorithm in parallel environment can be described as follows. For each document, we distribute its signatures from SF_n among all processors evenly. Once a processor finds a match, it broadcasts the result to other processors and the search is stopped. Otherwise, the processors can proceed to next tf group signatures. After all signatures from this document is processed, the search proceeds to next document's signatures. This algorithm can be easily modified to implement the upperbound search algorithms (Algorithm 3. 4). After searching $SF_i, i > 1$ for all documents, the T^{th} score is calculated. For each document, its SF_1 signatures are distributed among all processors and searched only if it has any chance to be one of the top documents. One problem

for this algorithm is that the number of signatures in some SF_i may be less than the number of processors for some high tf groups. It may be necessary to search several documents' signatures at the same time for some high tf groups to utilize all processors evenly. It will be interesting to find out how to distribute resources when the term frequency, thus the number of signatures, changes and how to do inter-processor communication to synchronize the search process. The granularity of the above algorithm is signature level.

We can also implement the weight-partitioning signature files in parallel environment at the document granularity level. Each processor fetches all signatures from one documents, searches them from high to low and then proceeds to the next documents. This approach solves the load balancing problem. It requires little inter-processor communication. We expect this implementation scales well as the number of processors increases.

Using AI in Information Retrieval

For most existing IR systems, precision and recall still hover below 50% for text retrieval. To a certain degree, the vector space model is like a brute force method in that it does not in any way try to simulate human deduction. The performance of the vector space model is fast approaching its limit. There is no doubt that new research will still achieve small-incremental improvements in search efficiency. However, we believe that significantly new ideas need to emerge for IR systems to improve beyond the current stage. Some kind of inference and deduction, either based on AI techniques like Bayesian network or logical model is needed to have a break through in retrieval

effectiveness. With the advance of computer power, it is possible to incorporate some AI methods into IR systems.

BIBLIOGRAPHY

- [1] AHO, A. V. and ULLMAN, J. D. Optimal partial-match retrieval when fields are independently specified. *ACM Trans. Database Syst.*, 4(2):168-179. June, 1979.
- [2] BICKLE, P. and DOKSUM, K. A. *Mathematical Statistics: Basic Ideas and Selected Topics*. Holden-Day, Inc., 1977.
- [3] D. C. Blair. The data-document distinction in information retrieval. *Commun. ACM*, 27(4). April, 1984.
- [4] Makris, C. Bozanis, P. and Tsakalidis, A. Parametric weighted filter: an efficient dynamic manipulation of signature files. *Computer Journal*. 38(6), 1995.
- [5] Buckley, C. and Lewit, A. F. Optimization of inverted vector searches. In *Proceedings of the ACM SIGIR Conference*, pages 97-110. 1985.
- [6] Chen, Ye-Sho and Leimkuhler, F. F. A relationship between lotka's law and bradford's law. and zipf's law. *Journal of American Society for Information Science*. 37(5):307-314. 1986.
- [7] CHRISTODOULAKIS, S. and FALOUTSOS, C. Design considerations for a message file server. *IEEE Trans. Soft. Eng.*, SE-10(2):201-210, March, 1984.
- [8] K. W. Church. One term or two? In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. July, 1995.
- [9] CROFT, W. B. and SAVINO, P. Implementing ranking strategies using text signatures. *ACM Trans. Off. Inf. Syst.*, 6(1):42-62, Jan., 1988.
- [10] Dumais, Susan Deerwester, Scott and Harshman, Richard. Indexing by latent semantic analysis. *Journal of the Society for Information Science*, 41(6), 199x.
- [11] Dillon, M. and Gray, A. Fully automatic syntax-based indexing. *Journal of ASIS*, 34(2), March, 1983.

- [12] C. Faloutsos. *Information Retrieval*, chapter Signatures files, pages 44–65. Prentice Hall, 1992.
- [13] C. FALOUTSOS. Signature files. In W. B. Frakes and R. Baeza-Yates, editors. *Information Retrieval: Data structures and algorithms*, chapter 3. Prentice Hall, 1992.
- [14] FALOUTSOS, C. and CHRISTODOULAKIS, S. Description and performance analysis of signature file methods for office filing. *ACM TOIS*, 5:237–257, July, 1987.
- [15] E. A. Fox. Lexical relations: Enhancing effectiveness of information retrieval systems. *ACM SIGIR Forum*. 15(3), 1980.
- [16] Frakes, William B. and Baeza-Yates, Richard, editors. *Information Retrieval*. Prentice Hall, 1992.
- [17] Fukunaga, K. and Narendra, P. M. A branch and bound algorithms for computing k-nearest neighbors. *IEEE Trans. on Computers*, C-24(7):750–753, July, 1975.
- [18] D. Harman. How effective is suffixing? *Journal of the American Society for Information Science*, 42(1), 1991.
- [19] D. HARMAN. Overview of the first text retrieval conference. In *Proceedings of the 16th Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval (Pittsburgh, Pa, June)*, pages 36–47. ACM, 1993a.
- [20] D. HARMAN. Overview of the second text retrieval conference (trec-2). In *Proceedings of the 2nd Text Retrieval Conference*, pages 1–20, Gaithersburg, Md., Aug., 1993b.
- [21] Baeza-Yates, R. A. Harman, D. Fox, E. and Lee, W. *Information Retrieval*, chapter Inverted files, pages 28–43. Prentice Hall, 1992.
- [22] Iadh Ounis and Jean-Pierre Chevallet. Using conceptual graphs in a multifaceted logical model for information retrieval. In *Database and Expert Systems Applications. 7th International Conference, DEXA '96 Proceedings*. Springer-Verlag, Sept., 1996.
- [23] Jae Woo Chang Jae Soo Yoo, Yoon-Joon Lee and Myoung Ho Kim. The hs file: a new dynamic signature file method for efficient information retrieval. In *Database and Expert Systems Applications. 5th International Conference, DEXA '94 Proceedings*. Springer-Verlag, Sept., 1994.

- [24] Jeong-Ki Kim and Jae Woo Chang. A new dynamic signature file method in parallel processing environment. In *Database and Expert Systems Applications. 6th International Conference, DEXA '95 Proceedings*. Springer-Verlag, 1995.
- [25] Choon-Hee Lee Jeong-Ki Kim and Jae-Woo Chang. Two-dimensional dynamic signature file method using extendible hashing and frame-slicing techniques. *Information Sciences*. 98(1-4). May 1997.
- [26] JOHNSON, R. A. and WICHERN, Dean W. *Applied Multivariate Statistical Analysis*. Prentice-Hall, Inc., 1988.
- [27] LEE, D. L. KIM, Y. M. and PATEL, G. Efficient signature file methods for text retrieval. *IEEE Trans. Data Knowl. Eng. To be published*, 1994.
- [28] KNAUS, D. and SCHAUBE, P. Effective and efficient retrieval from large and dynamic document collections. In *Proceedings of the 2nd Text Retrieval Conference*, pages 163–170, Gaithersburg, Md., Aug., 1993.
- [29] LEE, D. L. and CHUANG, H. Performance of document ranking and relevance feedback. *Submitted for publication*, 1994.
- [30] LEE, D. L. and LENG, C.-W. Partitioned signature files: Design issues and performance evaluation. *ACM TOIS*. 7(2):158–180, April. 1989.
- [31] LEE, D. L. and LENG, C.-W. A partitioned signature for multi-attribute and text retrieval. In *Proceedings of 6th international conference on data engineering*, pages 389–397. Los Angeles, CA. Feb., 1990.
- [32] LEE, D. L. and Ren, L. Document ranking on weight-partitioned signature files. *ACM Transactions on Information Systems*. 14(2):109–137. April. 1996.
- [33] LIN, Z. and Faloutsos, C. Frame-sliced signature files. *IEEE Transaction on knowledge and data engineering*. 4(3):281–289, 1992.
- [34] LIPMAN, D. J. and PEARSON, W. R. Rapid and sensitive protein similarity searches. *Science*. 227:1435–1441, March, 1985.
- [35] H.P. Luhn. Keyword-in-context for technical literature (kwic) index). *ASDD Report No. RC-127*. August, 1959.
- [36] Mohan, K. C. and Willett, P. Nearest neighbor searching in serial files using text signatures. *Journal of Info. Science*. 11:31–39, 1985.
- [37] J. Nie. An outline of a general model for information retrieval systems. In *In Proceedings of SIGIR 88, Grenoble*. ACM, 1988.

- [38] CHRISTODOULAKIS, S. THEODORIDOU, M. HO, F. PARA, M and PATHRIA, A. Multimedia document presentation, information extraction and document formation in minos: A model and a system. *ACM Trans. Off. Inf. Syst.*, 4(4):345-383, Oct. 1986.
- [39] P. K. PEARSON. Fast hashing of variable length text strings. *Communication of The ACM*. 33(6). June, 1990.
- [40] Perry, S. A and Willett, P. A review of the use of inverted files for best match searching in information retrieval system. *Journal of Info. Science*. 6:59-66. 1983.
- [41] ZEZULA, P. RABITTI, F. and TIBERIO, P. Dynamic partitioning of signature files. *ACM Trans. Inf. Syst.*, 9(4):336-339, Oct., 1991.
- [42] M. BOUGHANEM R. LAYAIDA and A. CARON. Construction an information retrieval system with neural networks. In *Database and Expert Systems Applications. 5th International Conference, DEXA '94 Proceedings*. Springer-Verlag, Sept., 1994.
- [43] C. S ROBERTS. Partial-match retrieval via the method of superimposed codes. In *Proceedings of the IEEE*, volume 67. pages 1624-1642. Dec., 1979.
- [44] G. SALTON. *The SMART Retrieval System-Experiments in Automatics Document Processing*. Englewood Cliffs, N.J.:Prentice Hall, 1971.
- [45] G SALTON. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York. 1983.
- [46] G SALTON. *Automatic Text Processing: The transformation, analysis, and retrieval of information by computer*. Addison-Wesley Publishing Company. 1989.
- [47] SALTON, G and BUCKLEY, C. Term-weighting approaches in automatics text retrieval. *Information Processing And Management*. 24(5), 1988.
- [48] Salton, G. and Lest, M. Computer evaluation of indexing and text processing. *Journal of the American Indexing and Text Processing*, 15(1), 1968.
- [49] Fox, E. A. Salton, G. and Wu, H. Extended boolean information retrieval. *Commun. ACM*, 26(12), December 1983.
- [50] A. F. Smeaton. Incorporating syntactic information into a document retrieval strategy: An investigation. In *Proc. 1986 ACM-SIGIR Conference on Research and Development in Information Retrieval, Pisa, Italy*, 1986.

- [51] Smeaton, A. F. and C. J. van Rijsbergen. The nearest neighbor problem in information retrieval. an algorithm using upperbounds. *ACM SIGIR Forum*. 16:83–87, 1981.
- [52] STANFILE, C. and KAHLE, B. Parallel free-text search on the connection machine system. *Commun. ACM*, 29(12):1229–1239, Dec., 1986.
- [53] C. J. van Rijsbergen. A non-classical logic for information retrieval. *The Computer Journal*. 29(6), 1986.
- [54] C. J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 1990.
- [55] S. F. Weiss. *A probabilistic algorithm for nearest neighbour searching*, pages 325–333. R.N. Oddy, S. E. Robertson, C. J. van Rijsbergen and P. W. Williams. Butterworths, London, 1981.
- [56] W. Y. P. WONG. *Design And Performance Evaluation Of Access Method And Heuristic Techniques For Implementing Document Ranking Strategies*. Ph.D. dissertation, The Ohio State University, 1991.
- [57] WONG, W. Y. P. and LEE, D. L. Signature file methods for implementing a ranking strategy. *Inf. Process. Manage.*, 26(5):641–653, Sept., 1990.
- [58] WONG, W. Y. P. and LEE, D. L. implementations of partial document ranking using inverted files. *Inf. Process. Manage.*, 29(5):647–699, Sept., 1993.
- [59] Wu, Sun and Manber, Udi. Fast text searching allowing errors. *Communications of ACM*, 35(10), October, 1992.
- [60] Lee, Y.J. Yoo, J.S., Kim, M.H. and Im, B.M. Performance evaluation of dynamic signature file methods. In *Proceedings. Nineteenth Annual International Computer Software and Applications Conference (COMPSAC'95) (Cat. No.95CB35838)*, 1995.
- [61] Buckley, C. Yu. C. T. and Salton, G. An evaluation of term dependence models in information retrieval. In Salton, G. and Schneider, H. J., editors. *Lecture Notes in Computer Science*, volume 146. Springer-Verlag, Berlin, 1983.
- [62] Lam, K. Yu. C. T., Buckley, C. and Falton, G. A generalized term dependence model in information retrieval. *Information Technology: Research and Development*, 2(4). Oct., 1983.
- [63] P. Zezula. Linear hashing for signature files. In *Proceedings of the IFIP TC6 and TC8 International Symposium on Network Information Processing Systems*. May, 1989.

.[]

IMAGE EVALUATION TEST TARGET (QA-3)

APPLIED IMAGE, Inc.
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved