University of A Coruña

Index Compression for Information Retrieval Systems

Ph.D. THESIS

Roi Blanco González

2008

PhD Thesis

Index Compression for Information Retrieval Systems

Roi Blanco González

PhD supervisor: Álvaro Barreiro García

Thesis committee:

Prof. Dra. Amparo Alonso Betanzos

Prof. Dr. Ricardo Baeza Yates

Prof. Dr. Fabio Crestani

Dr. Fabrizio Silvestri

Dr. David E. Losada Carril

Abstract

Given the increasing amount of information that is available today, there is a clear need for Information Retrieval (IR) systems that can process this information in an efficient and effective way. Efficient processing means minimising the amount of time and space required to process data, whereas effective processing means identifying accurately which information is relevant to the user and which is not. Traditionally, efficiency and effectiveness are at opposite ends (what is beneficial to efficiency is usually harmful to effectiveness, and vice versa), so the challenge of IR systems is to find a compromise between efficient and effective data processing.

This thesis investigates the efficiency of IR systems. It suggests several novel strategies that can render IR systems more efficient by reducing the index size of IR systems, referred to as *index compression*. The index is the data structure that stores the information handled in the retrieval process. Two different approaches are proposed for index compression, namely *document reordering* and *static index pruning*. Both of these approaches exploit document collection characteristics in order to reduce the size of indexes, either by *reassigning the document identifiers in the collection* in the index, or by selectively discarding information that is less relevant to the retrieval process by *pruning the index*.

The index compression strategies proposed in this thesis can be grouped into two categories: (i) Strategies which extend state of the art in the field of efficiency methods in novel ways. (ii) Strategies which are derived from properties pertaining to the effectiveness of IR systems; these are novel strategies, because they are derived from effectiveness as opposed to efficiency principles, and also because they show that efficiency and effectiveness can be successfully combined for retrieval.

The main contributions of this work are in indicating principled extensions of state of the art in index compression, and also in suggesting novel theoretically-driven index compression techniques which are derived from principles of IR effectiveness. All these techniques are evaluated extensively, in thorough experiments involving established datasets and baselines, which allow for a straight-forward comparison with state of the art. Moreover, the optimality of the proposed approaches is addressed from a theoretical perspective.

A Papá, Mamá, Iago e Antía

Our chief weapon is surprise...surprise and fear...fear and surprise....
Our two weapons are fear and surprise ... and ruthless efficiency ...
Our three weapons are fear, surprise, and ruthless efficiency ... and an almost fanatical devotion to the Pope...
Our four ... no ...
Amongst our weapons ...
Amongst our weaponry ... are such elements as fear, surprise ...
I'll come in again.

Cardinal Ximinez - Spanish Inquisition[1]

[1] Monty Python - *Nobody expects the Spanish Inquisition*

Acknowledgments

This thesis would not have been possible without the help of many. I wish to thank my supervisor Alvaro Barreiro for his invaluable mentorship, advice, guidance, and for giving me the opportunity to be a part of his group and writing this thesis under his supervision; I would also like to thank the whole Information Retrieval Lab at University of A Coruña, and particularly Chema, David, Jaime, Javier and Edu for their support all through these years.

My especial gratitude goes to David Losada for his advice and support during all the stages of this dissertation and to Fidel Cacheda for his encouragement, openness, and in general for being one of the nicest guys in the building.

I was lucky enough to be able to do two research stays that moved me steadily to learn and mature as a researcher. In 2005 I spent a few months in Glasgow University Information Retrieval group, where I was part of the *Terrier* development team. I am grateful to Prof. Van Rijsbergen's group, and especially Iadh Ounis for hosting me back in those days and opening the door to one of the leading groups in IR, and to my former office-mates who were kind enough to stand me back then. For that I thank Ben, Craig and most especially Vassilis and Iraklis for being there when I needed it most. The best thing I took away from Glasgow was having met Christina, who I am deeply indebted to. Most of these pages would have a somewhat different shape without her.

In 2007 I spent some months at ISTI-CNR, Pisa, kindly invited by Fabrizio Silvestri, who provided me with invaluable insights, support, and most of all friendship. Nothing else to say but *tu e la tua famiglia sono stati i migliori mai ospitanti*.

I wish to thank the other members of my PhD defence committee, Fabio Crestani, Ricardo Baeza and Amparo Alonso, for taking the time to read this thesis and for making very insightful suggestions, and also to the several anonymous reviewers of the papers that make up this dissertation.

Finally, I want to thank all the friends that have stood by my side all along the road, and foremost, my family, for their constant support and unconditional love.

Index

List of Tables

List of Figures

Chapter 1

Thesis Outline

1.1 Introduction

Information Retrieval (IR) systems have to address the problem of returning relevant information in response to a user's information need. In text-based retrieval, the data returned is grained in the form of *documents* which are conveniently accessed from the so-called index. IR systems must, at least, handle two different problems after being posed a user information need, or query. Firstly, they must distinguish and identify *relevant* information related to the query, and secondly, the response must be *fast*. This thesis focuses on the latter, but some of the techniques discussed are rooted on the same fundamental problem as the former.

This thesis investigates some new strategies for reducing the index size of IR systems. An index, also called inverted file (IF), serves as the data structure in charge of storing the information handled in the retrieval process. Traditionally, indexes are stored using static compression techniques, which encode internal representations of the data at hand. This work addresses two different approaches for index compression, namely document reordering and static index pruning. These two approaches are essentially different from static compression schemes, although they are complementary to them. The approaches proposed have one thing in common: they make use of some properties inherently related to a document collection.

The perspective taken in this thesis is to further exploit collection characteristics in order to reduce the size of indexes. In particular, techniques utilised in the IR field are adapted to:

- Enhance the compressibility of indexes by *reordering the document collection*.
- Reduce the size of indexes by selectively discarding information that is less relevant to the retrieval process by *pruning the index*.

The remainder of this chapter is organised as follows. Section 1.2 presents the motivation of the work to follow. Section 1.3 states the contributions of this work. Section 1.4 gives an overview of the thesis.

1.2 Motivation

The task of presenting in a efficient way the retrieval results is not easy, and has been addressed so far by different research branches, both algorithmic (faster methods) and structural (faster/smaller data structures). Typical examples of mechanisms involved in delivering results to the user quickly are index compression and query processing strategies.

Compression strategies aim to reduce the size of inverted files. This is useful for many reasons. Probably the most important reasons are a) increasing the caching capacities of the IR system, and b) reducing the bottleneck transfers between hard disk and Random Access Memory (RAM). Compression is increasingly important, given large-scale evaluation of IR systems and the exponential information explosion, mostly due to the prolific propagation of the Word Wide Web (WWW).

Traditionally, textual index compression encodes the data scanned from a collection of documents. The data is internally represented by natural numbers, and thus ordinary compression mechanisms deal with the problem of encoding integers. Therefore, how to represent those integers in an efficient manner is a central problem for efficient IR. There are many options to tackle the encoding issue, some of them more established in the field than the others.

Although index data encoding is the most reliable and effective form of compression, there exist other aids for improving the performance of IR systems. For instance, slight modifications in index structures, or suitable query resolving algorithms can boost the index throughput. More recently, it has been stated that some collection properties can affect compression. Specifically, traditional encoding schemes will work better with a *clustered* collection. Also, it is a fact that effective query evaluation can be performed without consulting all of the index data. Collection clustering and selective query processing are two different techniques employed to enhance the efficiency of IR systems.

This thesis follows a different path. Based on other concepts and mechanisms founded in retrieval modelling, this work investigates how both efficiency and effectiveness can be intertwined, in order to develop new approaches aimed at optimum index compressibility.

1.3 Thesis Statement

The claim of this thesis is that it is possible to manipulate the index of a textual document collection so that the index is smaller. This issue is addressed using two different techniques: document identifier reassignment, and static index pruning. Moreover, the optimality of both approaches is determined from a theoretical perspective. In particular, the work on index pruning bridges the gap between efficiency and effectiveness for compression with data loss for information retrieval: it brings knowledge commonly employed in retrieval modeling to tackle efficiency problems.

It is important to stress that the work presented in this thesis differs from classical compression of inverted files, although both approaches are complementary to each other. Whereas classical index compression tries to find a compact bit-representation of the information, the techniques developed in this thesis reorganise or discard that information in order for that representation to be as small as possible.

The intuition behind the mechanisms presented in this thesis is that the information contained in a document collection can be rearranged or discarded in order to reduce its space occupancy. Furthermore, the modified structural form representing the document collection content can be queried in *faster* and *better* ways. These two points refer to the problems of *efficiency* and *effectiveness* respectively.

The main contributions of this work are the following. Two variations on techniques for compressing inverted files are introduced. First, some empirical properties of textual collections are studied. Then, we characterise formally how a document collection should be re-arranged in order to obtain the maximum compression ratio for most utilised encoding schemes. Based on this fact, it is shown how compression of indexes of a textual document collection can be enhanced by algorithms that reorder the collection to enhance its empirical properties. Secondly, three different approaches are proposed for discarding information from an index. All these approaches are able to improve both the effectiveness and efficiency of IR systems. The first technique is a variation on a well-known pruning algorithm. The modification empirically demonstrates the fact that discarding information can be beneficial for IR effectiveness. The second approach is a set of methods for detecting non content-bearing words in documents, by using collection statistics. The last approach, defines a framework where the amount and nature of information to be discarded from an index is characterised on the basis of the estimation of probabilities. This framework differs from previous approaches for pruning an index. Under some assumptions, the novel pruning method proposed provides *optimal* retrieval effectiveness. As a side result from the development of this framework, we investigate the issue of finding indicators of the retrieval quality of documents, *without any query evidence*. These indicators are known as *document priors* which can greatly enhance retrieval quality on their own. All these techniques are validated on empirical evidence, through several series of thorough experiments,

which prove to be robust across different collections.

1.4 Thesis Outline

The novel contributions in this thesis are presented in chapters 4, 5 and 6. The rest of the chapters present basic concepts of retrieval and efficiency with up-to-date references. An IR specialist could skip the first sections and jump directly to chapter 4, but any interested reader may consult them as a brief introduction to the state of the art in IR. The organisation of the following chapters is as follows:

- Chapter 2 is a brief overview of the main concepts of IR, and points out how information retrieval systems deal with efficiency and scalability. Both efficiency and effectiveness are the main and older issues that IR systems have been dealing with, although we will see that growing advances in computation and networking capabilities generate new needs for IR, which eventually create new problems for the field. Section 2.4 presents the first of the issues a retrieval system must solve efficiently: how to manage the amount of information available nowadays, in different digital platforms, and the structures needed to support efficiently the retrieval process. Section 2.5 describes three different facets that involve in some way the search process, or how a user's information need is satisfied by an IR system. First, we describe the models used by IR systems, listing those employed for evaluating the approaches proposed in this thesis. Second, we describe the evaluation methodology for comparing retrieval approaches is described. Finally, we describe the ways in which retrieval systems achieve fast response times to user queries, by query evaluation procedures and dynamic pruning techniques.

- Chapter 3 introduces general schemes for compressing data, with a special focus on text compression. Then, we explain the mechanisms and techniques employed for compressing index structures, their motivations and the main issues faced by modern index compression methods.

- Chapter 4 begins with an empirical analysis of indexes of text collections with a special emphasis on how the distribution of terms in documents affects compressibility, and how this issue could be tackled. Then, we state the problem of finding an optimum disposition for an index for a given compression scheme and determine its NP-completeness. Next, we compare older solutions to the problem, and propose a novel approach for tackling the problem efficiently, by reducing the dimensionality of the data using Singular Value Decomposition (SVD). Furthermore we combine and compare several cluster-based techniques with the SVD-based technique and analyse empirically their efficiency versus time tradeoffs.

- Chapter 5 examines and develops several new strategies for reducing the size of indexes by discarding data. First, we demonstrate how index pruning can be beneficial for both effectiveness and efficiency. Next, we present several techniques to prune terms from an index, based on their relative informational content. Finally, we present and discuss a framework in which the amount of pruning is no longer dependent on any retrieval function at hand, but on particular probability estimations. All the techniques presented are able to outperform a well-known pruning method that is considered state of the art in index pruning. In particular, the development of the aforementioned framework led to the need of suitable *document priors* which are disccussed separately.

- Chapter 6 examines a set of novel document priors that arise in the formulation of the framework presented in the previous chapter. Those priors are estimations of the likely relevance of a document without any query information. This chapter presents two new families for computing those priors just based on document length.

- Finally, chapter 7 presents the conclusions of the thesis, a summary of the research contributions, and future research directions.

Chapter 2

Introduction to Information Retrieval

This chapter briefly introduces the basic concepts of Information Retrieval (IR) and presents some of the problems addressed in the field (section 2.2). The focus is on the two main concerns of IR research, which are to retrieve relevant bits of information to a user inquiry effectively (section 2.2) and efficiently (section 2.3).

2.1 What is Information Retrieval?

The field IR deals with the representation, storage of and access to information items (Baeza-Yates and Ribeiro-Neto [1999]). Modern IR is the most usual way of information access, mostly due to the increasing widespread of the World Wide Web (WWW). The concept of information in this context can be very misleading, as it is strongly bound to a user's request. IR systems are posed a *user information need*, normally in the form of a query, and they must return the whereabouts of pieces of information related to this enquiry, normally in the form of documents (van Rijsbergen [1979]). Notably, IR systems present the information solely as a ranking of documents, whereas other systems, like Question-Answering (QA), might further elaborate this presentation with an ulterior process. The inherent subjective facet inside a user's interest, implies that the problem of satisfying a user information need is always going to be open.

In this context, IR deals with the problem of finding and presenting documents of unstructured data that satisfy an information need from within collections of documents (Manning et al. [2008]). There are some points in the definition that need clarification. First of all, *unstructured* refers to data which have a semantically arbitrary structure, that can be conveyed unambiguously to a computer, as opposed to a relational database for instance. The term *document* refers to the granularity of the information presented to the user. It can represent abstracts, articles, Web pages, book chapters, emails, sentences, snippets, and so on. Moreover, the term *document* is not restricted to textual information, as users may be interested in retrieving and accessing multimedia data, like video, audio or images. *Collection* refers to a repository of documents from which information is retrieved. A user information need, also referred to as *query*, must be 'translated' in order for an IR system to process it and retrieve information *relevant* to its topic. This 'translation' is usually made by extracting a set of keywords that summarise the description of an information need. Lastly, the presentation of the information must be in such a way that facilitates the user to find the documents that he is interested in. A good IR system must support document browsing and filtering tasks for facilitating a user's *retrieval experience*.

Central to a typical IR scenario is the notion of *relevance*. The most difficult part of the retrieval process is deciding which documents are related to, or satisfy a certain query. Documents should be preferably ranked in decreasing order of relevance. An IR system achieves its maximum retrieval effectiveness when the relevant documents with respect to the query are ranked higher, whereas the non-relevant are ranked lower.

IR can include a wider range of tasks, for instance classification (associating a document to one or

more classes out of a predetermined set, like detecting Spam in a collection of emails), clustering (grouping together documents of the same topic), or filtering (discarding documents not related to a certain topic). All these tasks share in common the need of a *document* representation that a computer-based system is able to understand and process.

IR systems may be general or specialised with regard to the amount and type of data they have to scale up to, each facing different challenges. Examples of general IR systems are the widely available Web Search Engines (WSE) that process billions of heterogeneous Web documents. Examples of specialised IR systems are email retrieval systems or Desktop search systems integrated in modern Operating Systems (OS).

This issue of IR scalability is the main topic of interest of this thesis: IR systems must provide responses to user queries efficiently, and they must do it *fast*. IR systems need to be fast because of the exponentially increasing amount of information that is available for retrieval today. In fact, today's technological advancements have allowed for vast amounts of information to be widely generated, disseminated, and stored. This has rendered the retrieval of relevant information a necessary and cumbersome task, which requires effective and efficient systems.

2.2 The retrieval process

IR systems must cope with at least three different processes (Croft [1993], Hiemstra [2001]), illustrated in figure 2.1:

- Representing the content of documents.
- Representing a user's information need.
- Comparing both representations.

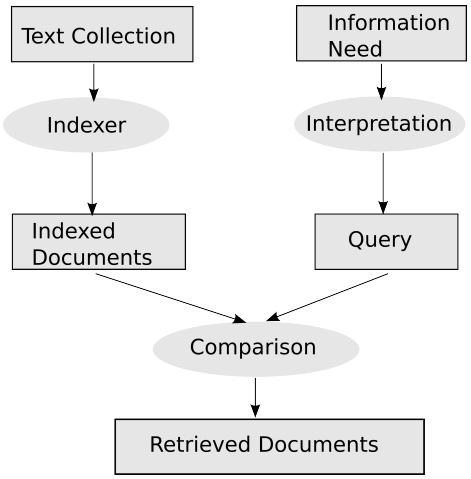


Figure 2.1: A General Framework for IR

The process of representing the content of documents is also called *indexing*. This task involves not only deriving which parts and keywords are extracted from the documents in the collection, but also a complex computational process described in more detail in section 2.4. IR systems usually carry out the former task in a trivial manner, for instance, down-casing the words occurring in the document. Other approaches are *stemming* and *stopwords removal*.

Stemming refers to the conflation of words to their lemmatised *base* form, where the morphological variants of words are stripped to one single suffix entry. This process employs shallow linguistic information (e.g. removing derivational endings, such as "-ion"), as well as language pattern matching rules (Porter [1980]). Stopwords removal refers to eliminating very frequent terms, like `the, of, a`,

from the indexing process, as they are likely a good indicator of poor content. Ruling out these words from the index can result in significant space savings, while not hurting retrieval performance. Traditionally, stopwords are chosen out of a fixed list, which is constructed from lexical information. This has led to the appearance of different stopword lists which vary in size, like Smart's (Buckley et al. [1995]) comprised of 571 terms, or Okapi's (Robertson and Walker [2000]), made up of 220 terms. Section 5.3 explains different ways for automatically detecting a candidate set of collection dependent stopwords, and includes an empirical study of the tradeoff between index occupancy and retrieval performance.

The representation of a user's information need by the IR system is also referred to as the *query formulation* problem. In a typical IR scenario, like a Web search engine for instance, the user is presented with a simple interface where he inputs some words stating his request for information. The system is expected to return an ordered list of documents, where the most relevant to the query should be on top. This is called *ad-hoc* retrieval. In order to facilitate the matching process, queries are represented in the same way as documents, by 'translating' the words occurring in the query into query terms, in the same way as documents during the indexing stage. In an operational situation, the matching process must proceed after a query is entered to the system (online), while the document processing-indexing procedure is carried out beforehand, without any user interaction (offline). Both offline indexing and online query processing are processes that must be fast and scalable, in order for a retrieval system to be truly operational.

The initial formulation of the user's query can be further modified. The process of formulating successive queries, either in an automatic way (pseudo relevance feedback) or by the user, is known as *relevance feedback* (Rocchio [1971]). The aim of relevance feedback is to enrich the query with other words that can be descriptive of the query, so that the new representation can retrieve more relevant documents. The procedure for pseudo-relevance feedback, also known as query expansion, takes an original query and a set of previously top-ranked retrieved documents. Then, it expands the query with terms likely to be correlated with relevance with respect to the query (Salton and Buckley [1980]). Other query expansion techniques reformulate the query using other sources of information like word relationships (Xu and Croft [1996]) or thesauri (Mandala et al. [1999]).

The matching process must interpret the representation of both query and document content, in order to rank documents according to their relevance with respect to the query. This implies that the IR system must employ some ranking algorithm to decide which documents to retrieve in response of a query. The design of ranking algorithms is based upon mathematical models that exploit some properties found in the document collection, like statistics or lexical information, or even data from external sources. A retrieval model predicts and explains what a user will find relevant given the user query (Hiemstra [2001]). Section 2.5.1 presents a short overview on retrieval modelling, and describes more deeply the foundations and equations governing some of the models used in this work.

2.3 Efficiency and Scalability

The algorithms and techniques that deal with retrieving data focus on understanding user information needs, document contents, and the notion of relevance. The success of the retrieval process depends not only on the outcome of the ranking process itself, but also on the speed in which the results are presented. Retrieval models care about *what* information is found, but not *how fast*. Modern IR systems have to deal with data of magnitudes never seen before which renders the retrieval process a tedious and difficult task. Without adequate and efficient techniques, retrieving information in modern-size datasets would be unfeasible. This is an important topic of research for both academic and corporate environments. In academia, it is imperative for new ideas and techniques to be evaluated on as near-realistic environments as possible; this is reflected in the past Terabyte track and recent Million Query track organised by the Text REtrieval Evaluation Conferences (Voorhees and Harman [2005]). In corporate environments, it is important that systems response time is kept low, and the amount of data processed high.

These efficiency concerns need to be addressed in a principled way, so that they can be adapted to

new platforms and environments. There are increasingly more such new platforms and environments, which need to have efficient retrieval capabilities. Some examples of new devices or environments are mobile devices, desktop search, or distributed peer to peer. In addition, new search scenarios appear, like expert search where the goal is to find a person knowledgeable about a topic, or federated search, where the goal is to combine the outputs of different search engines. This proliferation of new search applications is mostly due to hardware technological advances and to the wider spread and dissemination of information available to end users. For example, advances in circuit reductions allow mobile devices and PDAs a great amount of computational power. In addition, improvements in hard-drive capacities make nowadays half a Terabyte of data normal for desktop computers. The Web also plays a major role in this proliferation. The Web is formed of a set of interconnected nodes (Web pages) written in Hypertext (HTML) format, which is (normally) accessed through a *browser*. The Web allows for the development of new distributed architectures, for example peer to peer (P2P) networks. Peer to peer networks are formed of independent nodes that interact thanks to a communication protocol. Also, the Web provides access to data only accessible through search interfaces, or a search engine may decide to integrate several different sources of contents or the results coming from other search engines. IR comes at play when a user is to inspect, index or distribute the content of those networks/architectures. Then, the Web explosion, as the ultimate example of system in a scale never seen before, poses a great challenge for efficient retrieval. Hundreds of millions of people engage in information retrieval everyday, accessing a repository of a vast amount of Web pages, blogs, and multimedia content.

All the examples presented before pose new problems for IR. Although the goal for efficient IR should be in all cases the same (present the results as quickly as possible), the different scale of the systems, or their different architectural nature, forces to follow new paths.

Efficiency addresses problems found in two different stages of the retrieval process:

- Building the collection index
- Answering queries

Efficiency research over the past years has focused on efficient indexing, storage (compression) and retrieval of data (query processing strategies).

Efficient index building is necessary because of the huge amount of documents available for retrieval in modern systems. If an indexing algorithm is not designed and chosen carefully, a Web search engine may not scale up to billions of Web pages, or a desktop search system may require a personal computer idle for a longer period than what a user would accept as reasonable.

Answering queries fast is, arguably, a more important issue. Web search engines have taught the users to expect their results in less than a second. Also, there is a constant outgrowth of disposable content, much of which is updated on a daily basis, blogs or message boards being just some examples. Suitable and fast information mining is crucial in order for the accessible information to be of real use. In the last years, we are witnessing an ebullition of multimedia content: millions of images, videos, audio clips and so on, are now available to everyday users; again, managing that content poses a series of problems of its own: how to store multimedia content for faster access? how to index that content? should an IR system only store characteristics related to the file or also include information (or tags) on the context where it is found?

In summary, modern retrieval is not just about retrieving snippets of text efficiently, but also about retrieving other forms of digitalised content which may not be so “easy” to handle.

2.4 Indexing

IR systems need an indexing mechanism for performing efficiently the retrieval process. Although there are other options, the most popular data structure employed by IR systems is the *inverted file* (IF). An IF is a traversed representation of the original document collection, organised in *posting lists*. Each entry in the inverted file contains information about a single term in the document collection. Since this structure requires a large amount of space to be stored, posting lists usually are compressed.

Chapter 3 gives a more detailed explanation of traditional inverted file compression techniques. The index is built off-line, without any user interaction, in a rather complex process that involves many factors, most of them hardware-related.

2.4.1 Overview of the Indexing Process

Building an index from a document collection involves several steps, from gathering and identifying the actual documents to generating the final data structures (Manning et al. [2008]). These steps are illustrated in figure 2.2. We call the machine or machines in charge of indexing the collection *indexers* and the process of generating the data structures *indexing*. There are several steps to this process:

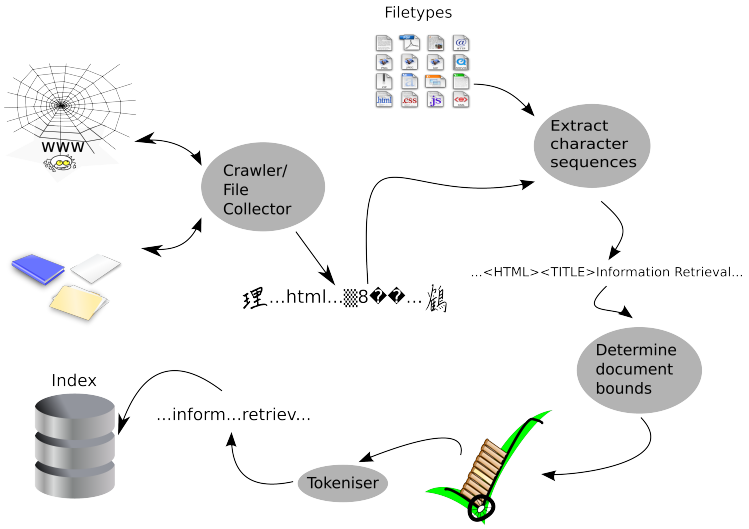


Figure 2.2: Global view of the indexing process

Step 1: It is necessary to provide indexable items coming from sources of information. The goal of this first subsystem is to feed raw text data to the indexer, and can consist of a simple script or a more complicated set of programs, depending on the problem at hand. The sources of information may be of different nature: files in a hard drive, email messages in local or remote folders, or Web pages coming from the Internet are some examples. The latter case is of special importance. Web pages constitute a hyper-linked environment, where it is possible to access one of more pages from another. These links and pages define the so-called *web graph*. The process of retrieving a set of Web pages for its subsequent indexing needs to *walk* through the web graph, and it is also known as *crawling* or *spidering*. This chapter does not address crawling, but the work by Castillo [2004] is a good reference for further reading. Another environment where retrieving data for indexing is not a trivial task is enterprise applications. Most of these applications contain documents encapsulated in different applications, like databases or content management systems, that have to be accessed in an appropriate manner.

Step 2: It is necessary to determine the character sequence inside each document. Documents are fed to the indexer as a stream of bytes, which have to be turned into characters. This is mandatory for the contents in every document to be processed in a uniform manner. There are some issues in this transformation worth pointing out. First, documents may come in different character encoding formats, the easiest case to handle being plain ASCII encoding. However, documents often come encoded by other methods (Unicode UTF-8, for example). The encoding mechanism has to be determined in order for a correct treatment of the text. Also, documents of different kind come in different formats. Some of these formats include markup languages which have to be correctly determined in order for their unnecessary tags to be discarded. This is the case of PDF, XML or Postscript (PS) files. The extraction of the textual part of some kind of files may not be trivial. In brief, the indexer not only needs to know about how to identify and decode different types of character

encoding schemes, but also needs APIs (Application Program Interface) that allow the manipulation of different kinds of files. Hereinafter, it is assumed that the indexer is able to solve all these issues and that all the documents are formed of sequences of characters. A third issue is to identify the bounds of the indexing units. In collections of files or Web pages this can be solved easily, but for other applications such as email indexing, we may want to define a file-per-email strategy. This needs a deeper understanding of the document structure.

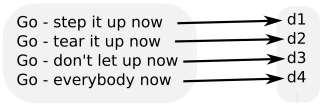
Step 3: At this point it is necessary to decide the granularity of the index. Most of the chapters to come assume that the index records only occurrences of terms in documents, but this is not the only case. In particular, we may want to index the sentences where a term occurs in, chapters, paragraphs, fields in HTML documents, and so on. All this information requires knowledge about the structure of the document. A special case is that of an index where every position of a word in a document is recorded. These types of indexes are useful for answering *phrase queries*, which require the exact match of a list of keywords in a precise order.

Step 4: This step transforms words from a document to indexable units, called tokens. This process is done just before the actual indexing phase. Some of the simple choices in this stage can be reducing punctuation signs, removing hyphens or down-casing words. Other options require linguistic information. Stemming and stopword removal are the most common linguistic techniques at this stage. More details of these two techniques were pointed in section 2.2. Some indexers abandon the traditional one-token-per-word strategy and index sequences of k words. This is known as a k-gram indexer.

Step 5: The last step of the index building process is the actual inversion of the collection, for creating two data structures called *dictionary* and *postings file*. This point is detailed in section 2.4.2 as it is where efficient design of algorithms comes at play. When focusing on the scalability of systems, this point is by far the most crucial part of the indexing process. Compression mechanisms, which are the main interest of this thesis, come to stage in this step.

2.4.2 Data Structures

Storing and fast accessing digital data is as old as computers themselves. In general, which data structure is more suitable for indexing depends on the particular application. For instance, databases (DB) must support range queries over keywords. A range query is of the form “*abc*" where every term containing the substring “abc" is a valid result. The case of information retrieval is not different. Some applications require boolean matching (see section 2.5.1.1), that is, only documents matching all the keywords are returned, whereas other environments need more sophisticated ranking algorithms, that incorporate term frequency information, document length and so on (see section 2.5.1). Also, the amount of data to be indexed is an important factor.



	go	step	it	up	now	tear	dont	let	everybody
d1	1	1	1	1	1	0	0	0	0
d2	1	0	1	1	1	1	0	0	0
d3	1	0	0	1	1	0	1	1	0
d4	1	0	0	0	1	0	0	0	1

Figure 2.3: Binary File

Suffix Arrays (SAs) (Manber and Myers [1990]) or Compressed Suffix Arrays (CSAs) (Grossi et al. [1993]) are a useful choice for applications that need pattern matching, which provide suffix

or substring search. Also, suffix arrays may be handy to support autocompletion queries, where every typed prefix is searched in the index and the search results change as the user completes the query keywords. However, these structures are not suitable for document search. Suffix arrays provide pointers to the text where the indexed sub-string occurs. The results a SA provides are sorted alphabetically and not in the order they appear in the document collection. Hence, their usability for boolean search is very limited.

Probably the simplest structure that provides keyword search is the *bitmap*. A bitmap is useful for environments that only require boolean search and are low-demanding in number of indexed documents. This structure, figure 2.3, can provide very fast boolean matching through documents and queries, at the expense of higher storage requirements.

Signature files (SFs), figure 2.4, are another option to be considered for indexing (Faloutsos and Christodoulakis [1984]). SFs store a descriptor for every document, which is a bit-based fingerprint representation of its contents. There are many ways for choosing the descriptor. Some of the options (Faloutsos and Christodoulakis [1987]) include word signatures, superimposed coding and compression-based methods: bit-block compression, run length encoding (RL), and variable bit-block compression (VBC). In the simplest form, using word signatures, every term t in the collection is hashed into a bitvector of size f, $H(t) \in \{0,1\}^f$. The descriptor for a document is just the concatenation of the descriptor of its terms. Searching is performed by retrieving the documents that contain the descriptors of the query terms. The main drawback of SFs is that any search may return false positives, which force to furthermore scan the document in order to perform an exact match.

Inverted files are considered the only practical choice for IR systems. Witten et al. [1999] provide a comparison between the performance of signature files, bitmaps, and inverted files, and an empirical basis for this claim. The work by Carterette and Can [2005] however, suggests that signature files are appropriate for searching large lexicons, especially if false drop resolution is fast relative to the time required to process lists.

IFs need to store both a dictionary and a posting list file (also called *concordance* file). The former contains the tokens indexed and global statistical information about them. The latter contains information about which terms appear in which documents, and probably where inside those documents, for instance the exact position, fields, or passages. The posting list file is in orders of magnitude larger than the dictionary file. The dictionary is usually stored in main memory for fast retrieval, whereas the postings normally reside on disk. There are some other approaches that consider loading the whole index in memory (Strohman and Croft [2007]), although it is probably a better idea to let memory caching systems (Baeza-Yates et al. [2007]) decide which information is kept on disk and which on RAM.

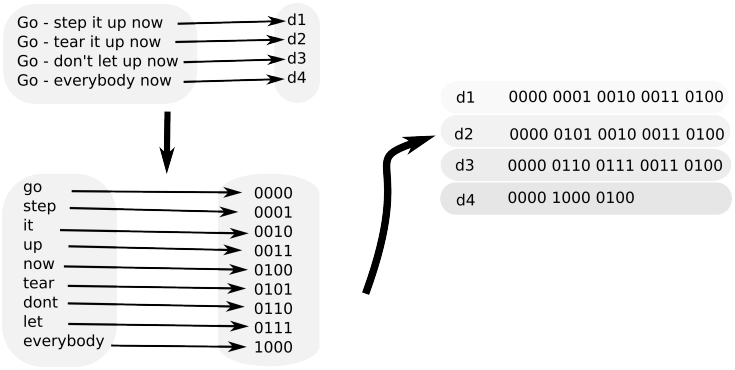


Figure 2.4: Signature File

2.4.2.1 Dictionary file

The dictionary file, also known as *lexicon* contains one entry for every token indexed from the collection. As mentioned, these can be k-grams, or single words, which might be preprocessed using

linguistic techniques. The lexicon is responsible for mapping every token into the position of its corresponding posting on disk. For effective ranked retrieval, every entry must, at least, store the information about:

- The actual token, represented as a stream of characters (string)
- The number of documents the term appears in (term frequency)
- A pointer to the on-disk posting file, where the concordance information for the term is stored.

When a query formed of a number of query terms arrives to the system, the first step is to look-up the dictionary file to see if the collection contains those terms. If that is the case, then the dictionary pointers provide a link to the rest of the data needed for the retrieval process. This look-up must be fast. There are many options for this structure: some examples are hashing tables, search trees, and the common *dictionary-as-a-string*, which is commonly used as the final dictionary structure in IR systems. Hashing tables are commonly employed during the indexing process, search trees are useful for suffix search, and dictionary-as-a-string is a common form of a sorted array.

Hashing tables transform the character representation of the term into an integer in a certain range. If the vocabulary set is known in advance, it is possible to design the mapping function in such a way that every term is assigned uniquely an integer. If that is not the case, collisions may appear, and they have to be resolved using external structures, usually hard to maintain.

The look-up process using hashing tables is very fast: it involves hashing the query term and performing an access to a vector in $O(1)$ time. Hashing only provides results for exact term matching. Section 3.2 provides some extra details on hashing functions.

On the other hand, search trees allow for suffix search. They are a suitable choice if the dictionary does not fit completely in main memory. String search inside trees begins at the root and descends upon the tree and performing a test at each branch to decide the path to follow. Search trees need to be balanced in order to achieve optimum efficiency. This means that the depth of the different sub-trees for a given node must be the same or differ in 1. One of the most widely used search trees is the B-tree. This kind of search tree is convenient because it leverages the effort of balancing. The number of terms inside each node is determined by the disk block size. This is due to optimising the fetch operations of terms from disk if needed.

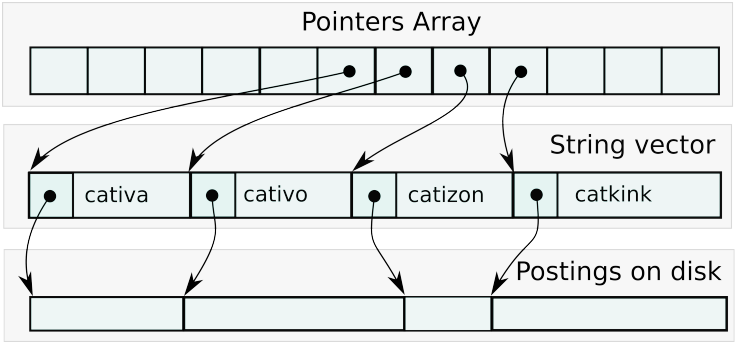


Figure 2.5: Inverted file with 2-level dictionary-as-a-string

Dictionary as a string (Witten et al. [1999]) stores the whole term set as a single string. Look-ups into the lexicon are realised through a simple binary search. In order to achieve better effectiveness, it is necessary for the terms to be stored within the same amount of space. Fragmentation, due to terms not having all of them the same length, is handled by a two-level scheme, where the first level is composed of pointers to the beginning of each term, and the second level is composed of the terms themselves. This is exemplified in figure 2.5. Other forms of dictionary files exploit string compression techniques in order to reduce their space requirements up to a point that they fit completely in main memory. The also make the indexing process more efficient. These dictionary compression techniques are also useful in dynamic environments, unlike hashing tables. This is discussed in section 3.2.

Finally, if regular expression search is needed over the term set, permutem indexes (Ferragina and Venturini [2007]) may be an efficient choice.

2.4.2.2 Postings file

The second data structure of an inverted file is the postings file or *concordance file* which contains the information of every term occurrence in a document. As stated before, this information is not only restricted to the presence or absence of terms. It can be enriched with other data such as frequency of terms in documents or the exact position of a term inside a document. Positional information can also refer to structural parts of the documents (fields in HTML text for instance), or arbitrary subdivisions of the documents, in sentences, paragraphs, or blocks of words of a certain size.

The format of the posting lists reflects the granularity of the inverted file, addressing in which documents and positions the term appears. Its simplest form only records binary occurrences of terms in documents:

$$< t; df_t; d_1, d_2, \ldots, d_{f_t} > , \tag{2.1}$$

where df_t stands for the frequency of the term t (number of documents in which t appears) and d_i is the document identifier. A document identifier is an integer bearing the internal representation of a document in the system. How to organise these identifiers in an appropriate manner is the main topic of chapter 4. As the notation implies, the document identifiers are ordered. This ordered disposition is not the only format option for postings lists, but it is useful for providing high compression ratios. Other dispositions of identifiers in the postings may aid the querying evaluation process by pruning the posting dynamically (see section 2.5.2). Some of them are frequency-ordered posting lists or impact-sorted postings (Anh and Moffat [2002]). For effective ranked retrieval it is necessary to record frequency of terms. In much of the work in this thesis, we assume a document level granularity with term frequency information. Therefore the posting list for term t is:

$$< t; df_t; (d_1, f_{t1}), (d_2, f_{t2}), \ldots, (d_{f_t}, f_{df_t}) >, d_i < d_j \forall i < j , \tag{2.2}$$

where f_{ti} is the term frequency in document i. A more fine-grained word-level index would have to include the ordinal position of every occurrence of a term in a document. These positions are sorted to coordinate query processing in document-at-a-time (DAT) query processing algorithms (see section 2.5.2):

$$< t; df_t; (d_1, f_{t1}; (pos_1, pos_2 \ldots, pos_{f_{t1}})), \cdots >, d_i < d_j \forall i < j , \tag{2.3}$$

where pos_i is the ordinal position in the corresponding document. Postings lists are stored compressed and contiguously on disk. This is advantageous for both space requirements and for making memory caches more effective (they are stored in contiguous blocks of memory). However this can be a problem for indexes that change over time. The issue of indexing a dynamic collection is presented in section 2.4.5. Posting structures can be further extended with pointers that allow to *skip* reading some portions of memory. This makes traversing the lists during query processing time faster.

2.4.3 Inverting the collection

The core part of the indexing process poses a scalability challenge itself. It may involve managing millions of words and documents, and billions of text occurrences. Just a few indexing algorithms scale up to the great amount of data available today, or are suitable to run in memory limited environments, such as mobile devices. The book by Witten et al. [1999] is the most renowned reference for indexing algorithms in the field. A newer revision of the state of the art can be found in Zobel and Moffat [2006]. The main algorithm described here, section 2.4.4.2, namely efficient single-pass indexing, is taken from Heinz and Zobel [2003]. This is the best performing approach for centralised indexing algorithms. Compression plays a key role in the indexing tasks, and it is necessary for the real scalability of indexing algorithms.

2.4.4 The problem

Hardware development defines the major constraints when an indexing algorithm is designed. These are some interesting facts to take into account (Manning et al. [2008]):

- Hard drives maximise their input/output throughput when the data is stored contiguously on disk. This is due to the fact that disks employ heads that take time to move to the right part of the disk.
- Disk data access is much slower than memory access. This is mostly due to the advantages in solid-state technologies and because RAM memory does not need any physical movement of a header to read randomly allocated data.
- Operating systems read or write blocks of a fixed size of bytes to disk. This issue is further discussed next.
- Data transfer is carried out by the system but and not by the processor, so it can perform other operations during I/O. This makes even more necessary the use of compression: it takes less to read and decompress compressed data than to read uncompressed data.

Hard drives encode data on rapidly rotating platters with magnetic surfaces. In order to read or write information, the platters spin at very high speeds, whilst read-and-write heads operate close over the magnetic surface. Platters are divided into tracks: concentric circles around the central spindle on both sides of the platter. These tracks are further subdivided into different sectors. Latency in a hard drive is the time for the access arm to reach the desired track and the delay for the rotation of the disk to bring the required sector under the read-write mechanism. Chief among the factors that determine the performance of a hard drive is *rotational speed*, measured in revolutions per minute (rpm). It directly affects the latency and disk transfer rate. Hard drives only spin at a constant speed; modern drives rotate at 7,200 or 10,000 rpms and have a media transfer rate of over 1 Gbit/s or higher. Rotational speed has not increased substantially in the last years (ULtraSCSI spinned at 10,033 rms in 1997).

The major advance in hard drive technology is the areal density increase. A higher number of platters over the magnetic region implies increasing the storage capacity. Since its very beginning (more than 50 years ago) the industry has increased dramatically the areal densities, roughly 27% per year (peaking in the 90s as much as 60% per year). It is expected that perpendicular recording technology can scale to about 1 Tbit/inch^2 at its maximum. Disk transfer rate improves at a 1:2 ratio with respect to areal density because data is stored in both sides of the platter but read from one dimension only. This is one of the reasons why, even though the storage capacity has increased by a factor of 50 in 10 years, sequential reading or writing has only increased in about a factor of 10.

All these facts motivate the use of suited indexing algorithms that minimise disk accesses, and retrieval strategies that can reduce the number of bits to be transferred from disk to main memory. Compression plays again a crucial role regarding this matter.

Lastly, it is possible to develop an analytical model of indexing, based on hardware performance parameters. The computational model allows for the estimation of total indexing time given a fixed set of conditions. That model has been used and described elsewhere, but that analysis is skipped in this chapter because it is not required for understanding the following sections. The reader is pointed to Witten et al. [1999], and Heinz and Zobel [2003] for a more in-depth detailed analysis of indexing algorithms.

2.4.4.1 An overview of the solutions

This section describes briefly four indexing approaches used in the past, and concludes with the considered state-of-the art in centralised indexing.

The simplest algorithm comes from viewing the occurrences of terms in documents as a big (sparse) matrix, much in the way a bitmap does. Indexing is just transposing the matrix, which can be constructed in two phases, each one performing a different pass over the text collection. The first

one determines the number of terms and documents to be indexed. The second one operates after allocating the matrix and filling it while re-reading the documents. This approach is unfeasible, as the space requirement for the matrix is prohibitive (89Tb for a 2.6G collection, as reported in Heinz and Zobel [2003]).

A slightly different option is to consider the matrix as a dynamic linked list (Moffat and Bell [1995]). The list is built on the fly and new entries are allocated on demand. Again, size requirements for the indexing phase make this approach unfeasible for collections in the order of Terabytes (for 6G a 2.6G collection).

Disk-based inversion, described by Candela and Harman [1990], accumulates postings on disk whereas only the lexicon resides in main memory. Postings are stored as linked lists, one for each term. A final inversion is finally achieved by traversing all the linked lists and serialising them into the final postings. The advantage of this approach is that it requires just one pass over the collection, at the expense of a large amount of temporary disk space. However, a major issue emerges when the algorithm traverses the linked list: the high number of random accesses leads to unpractical running times.

The first scalable approach to indexing is described by Moffat and Bell [1995], and is an improved version of an algorithm by Fox and Lee [1991]. Basically, the big extra space requirements are limited with the use of compression. The algorithm needs two different passes over the collection: a first pass that collects statistics on frequency of terms and precomputes the vocabulary which is conveniently stored in-memory, in compressed format. A second pass partitions the text in several loads, previously determined in the first pass, and allocates a big on-disk vector that will contain all the compressed postings. Every text chunk reserves another vector in memory, containing the postings in that portion of the collection. Both vectors are divided so that terms in the collection and respective chunks are allocated enough size. Every chunk is inspected and the in-memory vector subsequently filled. After a chunk is processed, the in-memory vector is flushed to an on-disk vector. The statistics gathered during the first phase of the inversion allow the indexer to calculate higher bounds for the subdivisions of the vectors, based on the compression encoding used. This approach operates with a limited amount of memory and does not require a large volume of temporary extra disk space. However, for optimum compression performance, the on-disk vector has to be re-compressed again: the original allocated buckets for each term are reserved in advance according to their upper estimated bounds, and thus better performance can be achieved when the exact size of each bucket is known.

Another family of scalable indexing algorithms are merge-based. The basic idea is to generate triples $(termid, docid, frequency)$ for their latter merge. The $termid$ and $docid$ numbers are integers assigned to the terms and documents in the collection based on their order of arrival. Firstly, the procedure scans the collection generating those triples until a certain memory threshold is reached, which means that we exceeded the available memory space available. Then, the triples are sorted by $termid$ first and $docid$ second, and flushed to disk. The lexicon, mapping between string terms and $termid$ is kept in memory during all the different runs. A second step, after all the documents have been processed, merges the K-runs written on disk. This is done by opening a buffer for every run, which is re-filled only when the contents of the buffer are consumed, i.e., the postings in that portion of the run are flushed into the final index. Buffering minimises random accesses to disk, and maximises the number of sequential accesses. Which of the K buffers is to be flushed to disk is decided by a priory queue in $\lfloor log_2 K \rfloor$ time, which is employed in comparisons and swaps.

Sort-based approaches can furthermore reduce the temporary extra space requirements, by writing the final inverted file in the blocks of disk occupied by the temporary runs. The process is described by Moffat and Zobel [1996]. A drawback of this approach is that the resulting index sorts terms by $termid$ and not in alphabetical order, making it difficult to perform range queries.

There are a number of similarities between two-pass and sort-based approaches. They make use of compression techniques for lowering the space overhead. Both keep the lexicon in memory during index construction (which is also compressed for greater efficiency) and allow for a trade-off between memory consumption and running time. To perform efficient memory computations it is necessary to determine the number of loads the text has to be subdivided in. Figures reported in Heinz and Zobel [2003] show that the sort-based approach outperforms the two-pass approach in constructing word-level or document-level indexes.

Finally, the main drawbacks of the sort-based approach are solved by the single-pass in memory algorithm, which is described next.

2.4.4.2 Efficient Single-pass Indexing

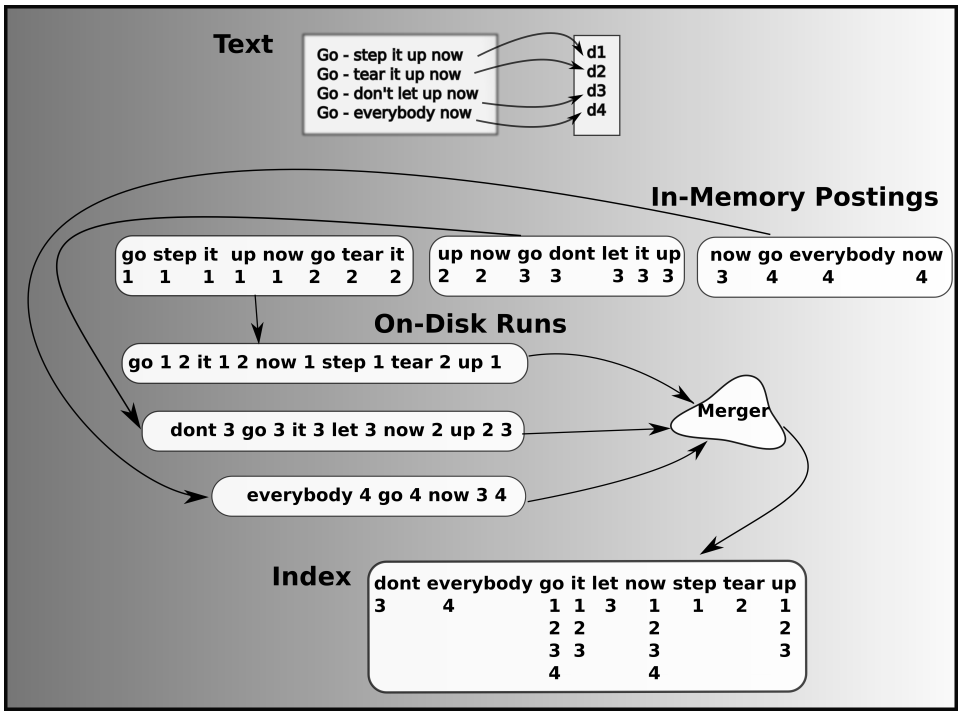


Figure 2.6: Single pass indexing

The single-pass approach is proposed in Heinz and Zobel [2003] and illustrated in figure 2.6. This algorithm outperforms the other techniques presented before, in indexing time. Also, single-pass indexing operates with a limited amount of memory and uses a restricted volume of extra disk space. The algorithm shares some common points with the two-pass and sort-based approaches. Firstly, like in sort-based indexing, it scans all the documents and generates the $(termid, docid, fre)$ triples. The dictionary is stored using a hash table, proved to be faster than tries and other structures (Zobel et al. [2001]). There are no string-to-integer maps in this phase of the algorithm. If a new term arrives, it is allocated a bit-vector, which will store its compressed postings. When the memory threshold is met, the lexicon is sorted. If we were using some tree-based structures, like tries or splash tries, this step could be omitted, as they provide sorted linear search. However, look-up and insertion times are much slower than those devised by a hash table. After sorting the dictionary, both the lexicon and the postings are serialised on disk. The lexicon is freed, and the process starts again with the initial amount of memory available again. As no statistics are gathered in the whole process, only non-parameterised compression schemes (section 3.3) can be employed for compressing the postings in the bit-vectors. When all the documents are processed, the merging phase follows up. This is done, again, like in the sort-based approach: using an in-situ K-way merge where the comparisons now are between strings and not integer term identifiers.

One of the main advantages of this approach is that there is no need to keep the lexicon in memory between runs. This is especially beneficial for rare terms never accessed, or accessed very few times. Furthermore, the final dictionary produced is alphabetically sorted, and thus efficient support for range queries can be provided. Another advantage of this indexing scheme over the sort-based scheme is that it is asymptotically more efficient. Therefore, it is more likely to scale up better.

2.4.5 Incremental Indexing

The indexing cases considered so far assume that the documents comprising the collection never change. However, some document collections are dynamic, which means that they change their contents over time. Two examples of applications employing this kind of collections are desktop search and enterprise search. Although it is possible to require the IR system to handle document updates and deletions, most literature on dynamic updates focus on the problem of adding new documents. Dynamic indexing is also known as on-line indexing, because the documents arrive to the system when it is already available for querying.

Recall that the postings are conveniently stored contiguously on disk, and changes over postings imply extending these structures in a rather time-consuming process. This would degrade greatly the effectiveness of a retrieval system. The most common option is to just rebuild the index from scratch at a given point. This is the pattern most Web search engines follow. Index updates add complexity and tensions to be taken into account into the process of building the index, like query throughput and document insertion rate. In addition, the support for delete/update operations force the usage of more complicated maintenance procedures.

If documents must be available as they arrive, there are two possible options, namely 'in-place updates' and 'merge updates'. Both options maintain an in-memory disk which contains the most recently updated documents, and differ in their behaviour when memory consumption reaches a certain limit:

- In-place update preallocates size in the end of the postings lists, so newer documents can fit in the postings.
- Merge update merges the in-memory and on-disk indexes.

As mentioned earlier, the IF is stored in two parts, an in-memory part of recently included documents and an on-disk component. List preallocation is problematic in many ways. First of all, it is the issue of determining the amount of size at the end of the posting. Too much preallocation wastes space and resources, whereas allocating too little space will force the use of pointers to link the newer parts of the posting, leading again to the problem of disk random accessing. Merge update joins both the in-memory and on-disk components on demand. When the memory threshold is reached, both indexes are merged into a new on-disk index, in a sequential manner and avoiding unnecessary disk seeks. This approach was first presented by Cutting and Pedersen [1990], where it is also stated that the ratio between the number of terms and postings to be updated $\frac{|postings|}{|terms|}$ grows in a ratio of $\Theta(log(postings))$. There are several variants of these two basic approaches: Tomasic et al. [1994], Shieh and Chung [2005], Lester et al. [2004], Lester et al. [2005b]. In general, merge update is considered more efficient than in-place update. Finally, a hybrid maintenance scheme is presented in Büttcher et al. [2006].

2.4.6 Distributed Indexing

Distributing architectures bring additional problems to the field. The main problem is how to build and maintain an inverted file using more than a single machine. This is important when dealing with a very large document collection, like the case of the Web. The index is built by a cluster of machines in order to achieve reasonable indexing times. The several indexers must be coordinated in some way so that the final inversion can be done. Normally, the load is balanced between indexers. The final index produced is a partition of the one that a centralised indexer would build. The partition can be done in two different ways: splitting the documents across different servers or assigning different index terms to different servers. Term-based index partitioning is also known as distributed *global indexing*, whereas document-based partitioning is also known as distributed *local indexing* (Ribeiro-Neto and Barbosa [1998]).

Choosing between the two assignment options depends on the final representation of the index that is going to be used for querying. Normally, queries arrive at a central point, known as broker, which distributes the query and returns back the results. The broker is in charge of merging the

posting lists and producing the final document ranking. In a term-based environment (figure 2.7), the query is sent to the servers containing the query keywords, and merged in a *broker*. Load balancing depends on the distribution of query terms and its co-occurrences. This can be decided on the basis of clustering properties, query log analysis, and so on. The main drawback is that the rule which decides optimal balancing is not fully well understood, as it depends on the distribution of terms over queries. Assigning documents to each partition is by far a more common approach. Every server

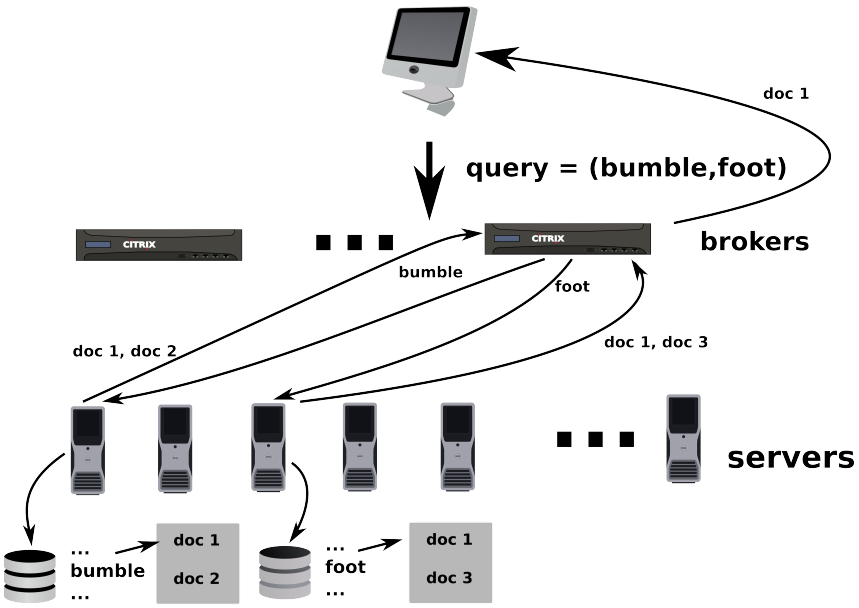


Figure 2.7: Term partitioned index in a distributed architecture

receives all the query terms, and performs a local search. Documents are sent to the broker, which sorts them. The main complication on this scheme is to maintain global collection statistics inside each server, which are needed for ranking the documents. The statistics may be updated on the fly by a structure shared by all the servers, or gathered after indexing. The latter approach scales up better for collections of millions of documents.

Most of the techniques for distributed indexing rely on the fact that it is possible to employ a big set of inexpensive machines, configured as a cluster for instance, instead of a powerful single machine. This is useful in the case of failure of an indexer, because its workload can be easily reassigned. Some approaches are based on simply splitting the workload into different index servers and performing a single-machine local indexing algorithm, but with certain additional considerations (Ribeiro-Neto et al. [1999]).

More sophisticated techniques include pipeline scheduling for optimising indexing time (Melnik et al. [2001]). Pipelining splits the indexing process into three different phases: loading the documents, processing the documents (building a local index) and flushing the documents. It is possible to schedule these three phases on a pipeline, so that it can be executed in parallel by three different threads. It is possible to conduct some theoretical analysis that drives the selection of optimal size of the loads, or buffer sizes.

One example of a specific distributed indexing algorithm is the map-reduce algorithm, claimed to be in use in some commercial search engines, (like Google[1] in 2004). It is a method for splitting the construction of the index into small tasks, assignable to different index servers. The construction is managed by a central node, in charge of distributing the indexing load. The basic idea is that the master node assigns partitions of the collection to index on the fly, depending on the availability of each index server. More details on map-reduce can be found in Dean and Ghemawat [2004].

[1] `http://www.google.com`

2.5 Querying

2.5.1 Retrieval Models

Models for IR try to provide a mathematical explanation of what a user will find relevant for a given query. Formalisation of the retrieval process is needed for the better understanding of the process itself, and for providing a description to guide its implementation.

Classical retrieval modeling considers documents as *bags of words*. This stands for the view of the model as an entity without structure, where only the number of occurrences of terms are important for determining relevance. Other approaches consider the notions of term proximity and co-occurrence of data (van Rijsbergen [1977], Xu and Croft [2000], Metzler and Croft [2007]), and field structure for HTML documents (Robertson et al. [2004], Plachouras and Ounis [2007]). Whenever a query is inputted to a retrieval system, every document is scored with respect to the query. The scores are sorted and the final ranked list is presented to the user. A retrieval model is in charge of producing these scores.

In general, models for retrieval do not care about efficiency: they solely focus on understanding a user's information need and the ranking process. For this reason, it is necessary to develop suitable query resolving mechanisms, section 2.5.2, or additional structures that extend the capabilities of the index in order to provide fast ranking, section 2.5.3. The issues of effective retrieval are presented later in this section.

2.5.1.1 The boolean model

The Boolean model is the first model of IR which treats the user input query as an expression devised by boolean logic. It is an exact match model (Fox et al. [1992]), which implies that a document is retrieved if and only if it matches the description of the query term set. The query description is devised by boolean logic, and uses three different `AND`, `OR` and `NOT` operators. The retrieval process is carried out using operations with sets. Every term in the collection defines a set formed by the documents it indexes. Boole's operators define precisely which subset of documents must be retrieved, by the intersection, union and symmetrical difference operations.

Even though the Boolean model is very simple, search engines employed it intensively until the mid 1990s. The Boolean operators allow for redefining the query successively, in order to refine the result set. If the document set is too big, it can be effectively narrowed down by inputting more query terms, or by using other operators, and vice versa. This is very effective for the experienced user, but of no real applicability for the average user from a modern IR point of view.

The absence of any ranking for the returned results is one amongst the most criticised points for the Boolean model. Another drawback is the restrictions in the query formulation process, which may return misleading results if the query is not well-stated. Also, Boolean logic operators are hardly able to model a user's information need posed in natural language. All this disadvantages turn the Boolean model down as an option for modern retrieval systems, and more precisely for web search.

Models of ranked retrieval use statistics of the terms in the collection to overcome the issues of Boolean models. These statistics can de derived from different sources, like the distribution of terms over documents or document lengths. The way sources of information are combined to retrieve documents is decisive for the scoring function of the retrieval model. Those scores are no longer binary, and thus allow for more effective retrieval. One of the ingredients of ranked retrieval is to capture the notion of *how much* a certain term contributes to the description of the document, in what is called *term weighting*. Models presented next move away from the Boolean model by incorporating term weights into their ranking process.

2.5.1.2 The vector space model

A first attempt to model query-document matching in a non Boolean way is Salton's vector space model (Salton et al. [1975a]). Within this model, documents and queries are elements of a highly dimensional vector space, where the number of dimensions is determined by the number of terms

occurring in the collection. The main disadvantage of the vector space model is that it does not subscribe what the values of the vector components should be.

A first example of a possible term weighting scheme could be to consider non-binary vectors: for every document d_j, the value in its $i-th$ component represents the frequency of term t_i in d_j. Recall that most of the components are 0, and that the resulting vectors are very sparse. This representation allows for several document-query distance measures. The objective is to measure the similarity (sim) between documents and queries. A first approach is to compute the cosine angle between the query $\overrightarrow{Q} = (q_1, q_2, ..., q_n)$ and document vectors $\overrightarrow{D} = (d_1, d_2, ..., d_n)$:

$$sim(\overrightarrow{D}, \overrightarrow{Q}) = \frac{\sum_{i=1}^{n} d_i \cdot q_i}{\sqrt{\sum_{i=1}^{n} d_i^2} \cdot \sqrt{\sum_{i=1}^{n} q_i^2}} \tag{2.4}$$

There are several variants of this similarity measure in the IR literature. This section only presents a variant employed later on, and some key features addressed by this and other IR models. The following nomenclature is used in the next sections:

- N is the number of documents in a text collection
- $|Q|$ is the number of query terms in a query Q
- $dl(D)$ is the length of document D, measured in number of tokens
- $avgdl$ is the average document length in the collection
- $df(t_i)$ is the number of documents a term t_i occurs in, or *document frequency*
- $tf(t_i, D)$ is the number of times a term t_i occurs in document D, or *term frequency*
- $avgtf$ is the average term frequency in a document
- TF_i is the number of tokens of a term t_i in the collection, or $TF_i = \sum_d tf(t_i, D)$
- T is the number of tokens occurring in a collection, or $T = \sum_d TF$

One of the crucial aspects for term weighting functions is to normalise the contribution of the document length to the final score or similarity measure between $\overrightarrow{Q}$ and $\overrightarrow{D}$. In equation 2.4 this was controlled by the normalised length of the document vector, $\sqrt{\sum_{i=1}^{n} d_i^2}$. The weighting model presented next is also based on the vector space retrieval model, but with a different document length normalisation factor.

$$sim(\overrightarrow{Q}, \overrightarrow{D}) = \sum_{i=1}^{|Q|} \frac{\frac{\log(1+tf(q_i,D))}{\log(1+avgtf)} \cdot \log \frac{N}{df(q_i)}}{\sqrt{(1-slope) \cdot avgdl + slope \cdot dl(D)}} \tag{2.5}$$

The document length normalisation $\sqrt{(1-slope) \cdot avgdl + slope \cdot dl(D)}$ is referred to as pivoted normalisation (Singhal et al. [1996]), and the default value for $slope$ is 0.2 (Buckley et al. [1995]). As well, the $\log \frac{N}{df(q_i)}$ component is called Inverse Document Frequency (idf) Sparck Jones [1972]. Idf has become a crucial part of weighting models, whether introduced in a heuristic fashion, like in the case of equation 2.5, or derived from a probabilistic point of view, like in equation 2.7. It is a measure of the *specificity* of the term, but can also be seen as a measure of the relative content, or informativeness. The main idea behind idf is that the fewer the documents the term appears in, the more discriminative the term is, and thus the more important for retrieval. Some variants of the idf formula are scattered over the IR literature. Some of them are employed in section 5.3 to show how to select content-poor words for decreasing the index size and speeding up query evaluation.

There is extended work on the vector space model. In particular, section 5.3.3 presents how it is possible to measure the discriminative value of terms on the basis of their disposition in the vector space formed by the collection, and equation 2.5 is used in several subsequent retrieval experiments.

2.5.1.3 Probabilistic model

This model of retrieval is based on the estimation of the probability of relevance for a document given a query (Robertson and Sparck-Jones [1976b]). It assumes a probability distribution of the term distribution over the set of relevant documents. Furthermore, it is possible to refine the knowledge about this distribution by interaction with the user.

The exact formulation of this principle as well as a new application for it, is presented in section 5.4.1.

The probabilistic model is one of the few retrieval models that does not need an additional term weighting algorithm to be implemented. Ranking algorithms are completely derived from theory in the following way:

$$sim(Q, D) = \sum_{t \in Q} \log \frac{(r + 0.5)/(R - r + 0.5)}{(df(t) - r + 0.5)/(N - df(t) - R + 0.5)} , \tag{2.6}$$

where R is the number of relevant documents, and r is the number of relevant documents that contain a term t. However, in most real applications the distribution of terms over relevant and non-relevant documents will not be available. In absence of any relevance information, the score becomes (Harter [1975]):

$$sim(q, d) = \sum_{t \in Q} \log \frac{N - df(D) + 0.5}{df(D) + 0.5} \tag{2.7}$$

Bookstein and Swanson [1974] presented a model to identify index terms from a document. The model states that the number of occurrences tf of terms in documents follows a mixture of two Poisson distributions (2P) as follows. Let X be a random variable for the number of occurrences, $\pi \in [0, 1]$ a parameter controlling the mixture distribution, and μ_1 and μ_2 the means of the Poisson distributions:

$$p(X = tf) = \pi \frac{\mu_1^{tf} e^{-\mu_1}}{tf!} + (1 - \pi) \frac{\mu_2^{tf} e^{-\mu_2}}{tf!} , \mu_1 \geq \mu_2 \tag{2.8}$$

For each term the documents are cast into two different sets. Documents in subset one (*elite*) treat a subject referred to by a term to a greater extent than documents in subset two (*non-elite*). The documents in the elite set for a term are considered as the answer to query formed of the single term. Harter [1975] presented a first way to rank documents under the 2-Poisson distribution assumption (extended by Margulis [1992] to a N-Poisson mixture). It assumes that the 2P can be used as a measure of the effectiveness of a term, by using:

$$z = \frac{\mu_1 - \mu_2}{\sqrt{\mu_1 + \mu_2}} \tag{2.9}$$

The probabilistic model was further extended by Robertson et al. [1981] to include term occurrences into the ranking. This led to estimating probabilities of relevance and non-relevance within the elite and non-elite sets, which had to be eventually estimated by a number of parameters. Unfortunately, estimating those parameters in a straightforward way is not possible.

The Best Match (BM) family of scoring functions by Robertson et al. [1995] are based on some observed characteristics of the 2-Poisson models, and are not directly derived from the estimation of probabilities. A particularly effective retrieval model, equation 2.10, is known as Okapi's Best Match25 (BM25). BM25 has been proved to be robust and stable in many IR studies and considered as a well-established baseline for comparisons.

$$sim(Q, D) = \sum_{i=1}^{|Q|} \log \frac{N - df(q_i) + 0.5}{df(q_i) + 0.5} \frac{(k_1 + 1) \cdot tf(q_i, D)}{K + tf(q_i, D)} \frac{(k_3 + 1) qtf(q_i)}{k_3 + qtf(q_i)} , \tag{2.10}$$

where $K = k_1 \cdot ((1 - b) + b \cdot \frac{dl}{avgdl})$, and $qtf(q_i)$ is the frequency of q_i in the query. The behaviour of the BM25 scores is governed by three parameters, namely k_1, k_3, and b. Some studies (Chowdhury et al. [2002], He and Ounis [2003]), have shown that both k_1 and k_3 have little impact on retrieval

performance compared to the b parameter, and in many experiments they are set to the constant values recommended by Robertson et al. [1995] ($k_1 = 1.2$, $k_3 = 1000$). The $b \in [0, 1]$ parameter controls the document length normalisation factor. Basically, it decides upon how much of the score is assigned to the document length, and how much to the average document length.

The BM25 scoring function is based upon the following four hypotheses, taken directly from the 2-Poisson model:

- the weight should be zero when the frequency is zero
- the weight should vary monotonically according to the term frequency
- the weight should converge to an asymptotic maximum and,
- that maximum should be equation 2.7.

The last probabilistic model family presented is the Divergence From Randomness (DFR) framework (Amati and van Rijsbergen [2002]). The key assumption behind DFR is that an informative term's distribution is not random. Then, it is possible to quantify the informativeness of the term by measuring how much the actual distribution diverges from a distribution that predicts random occurrences of terms. More precisely, DFR combines two probabilities for determining the weight of a term:

$$weight(t) = (1 - p(tf|Elite)) \cdot (-\log p(tf|\mathcal{C})) \ , \tag{2.11}$$

where $Elite$ represents the elite set of documents. This set is considered to be the set of documents where the term appears in. $\mathcal{C}$ refers to the collection of documents. Different estimations of these probabilities lead to different models.

We present next the formulation for DLLH (Amati [2006]), used in section 5.4.3.5. This model is useful in some of the following experiments because it is *parameter-free*, so tuning is not necessary for obtaining high retrieval performance. Parameter-free models reduce the number of variables and factors to take into account, which will be convenient for proving that the performance of certain techniques is independent of the choice of a particular set of parameters.

$$sim(Q, D) = \sum_{i=1}^{|Q|} \log(1 + \frac{1}{tf(q_i, D)}) \cdot tf(q_i, D) \cdot \log(tf(q_i, D) + \frac{avgdl}{dl(D)} \cdot \frac{N}{TF_i}) + \\ \frac{1}{2} \cdot \log(2\pi \cdot tf(q_i, D) \cdot (1 - \frac{tf(q_i, D)}{dl(D)})) \tag{2.12}$$

2.5.1.4 Language modeling

Language models (LM) for information retrieval (Ponte and Croft [1998]) are retrieval models (taken from the speech recognition field) that do not impose a explicit parametric form for the probability of relevance. Lafferty and Zhai [2003] presented a formal connection between probabilistic and language models.

The basic idea of the language modeling approach to information retrieval is to assume that a query Q is *generated* by a probabilistic model of document D. In this context, the generative language models approach estimateï£¿is the probability of the query being generated by a document. Hence, LM estimate the conditional probability $p(Q|D)$. Ranking can be performed by applying Bayes rule:

$$p(D|Q) \stackrel{rank}{=} p(Q|D)p(D) \tag{2.13}$$

where the document independent component $P(Q)$ has been dropped (it is document-independent and therefore it does not affect ranking).

It is usual to decompose the query into its query terms $Q = \{q_1, q_2, \ldots, q_n\}$ and assume that:

- term occurrences are independent of each other and generated by a multinomial distribution.

- $p(D)$ is uniform over the whole set of documents. However, $p(D)$ can represent query independent information for ranking. Chapter 6 shows how to estimate $p(D)$ in a way that retrieval performance is enhanced with respect to a LM that assumes $p(D)$ to be uniform.

Then, the *query likelihood* $p(Q|D)$ is estimated as follows (multinomial model):

$$p(Q|D) = \prod_{i=1}^{n} p(q_i|D) \tag{2.14}$$

In order to rule out zero probabilities for non-seen terms in a document, this estimate has to be *smoothed*, which eventually leads to different language model-based scoring functions. Most smoothing methods employ two distributions, one for words occurring in the document (p_s) and one for *unseen* words (p_u). Taking logs, it can be shown that equation 2.19 suffices to provide a document rank using sums of logarithms, equivalent to the one that equation 2.14 would yield:

$$\log p(Q|D) = \sum_{i} \log(q_i|D) \tag{2.15}$$

$$= \sum_{i \backslash tf(q_i|D)>0} \log p_s(q_i|D) + \sum_{i \backslash tf(q_i|D)=0} \log p_u(q_i|D) \tag{2.16}$$

$$= \sum_{i \backslash tf(q_i|D)>0} \log \frac{p_s(q_i|D)}{p_u(q_i|D)} + \sum_{i=1}^{N} \log p_u(q_i|D) \ , \tag{2.17}$$

where $tf(q_i, D)$ stands for the frequency of term q_i in document D, α_d is a parameter and $p(q_i|\mathcal{C})$ is the collection language model.

Finally, the formula becomes

$$\log p(Q|D) = \sum_{i \backslash tf(q_i|D)>0} \log \frac{p_s(q_i|D)}{\alpha_d p_u(q_i|D)} + |Q| \log \alpha_d + \sum_{i=1}^{N} \log p(q_i|\mathcal{C}) \tag{2.18}$$

$$\stackrel{rank}{=} \sum_{i \backslash tf(q_i|D)>0} \log \frac{p_s(q_i|D)}{\alpha_d p_u(q_i|D)} + |Q| \log \alpha_d \ , \tag{2.19}$$

where α_d is a document dependent constant. Different smoothing techniques lead to different p_s estimations and eventually to different matching functions. For instance, Jelinek-Mercer (JM) (also known as linear interpolation):

$$p_s(q_i|D) = (1-\lambda) p_{mle}(q_i|D) + \lambda \, p(q_i|\mathcal{C}), \ \lambda \in [0,1] \, , \alpha_d = \lambda \tag{2.20}$$

where $|D| = \sum_{w_i \in D} tf(w_i, D)$ (the document length), p_{mle} is the maximum likelihood estimator for a term q_i given a document d, $p_{mle}(q_i|D) = \frac{tf(q_i,d)}{|D|}$ and λ is a parameter controlling the amount of mass distribution assigned to the *document* and *collection*.

Another popular and effective smoothing technique is Dirichlet smoothing:

$$p_s(q_i|D) = \frac{tf(q_i, D) + \mu p(q_i|\mathcal{C})}{|D| + \mu}, \ \alpha_d = \frac{\mu}{|D| + \mu} \ , \tag{2.21}$$

where μ is a parameter.

A full derivation of the connection of language models and the probabilistic model of retrieval is presented in section 5.4.1; it is used to justify the development of novel high-quality query independent metrics. In section 5.4 we use the language modeling approach as an estimation inside a novel static pruning framework. Moreover, in chapter 6 we show how it is possible to improve significantly the retrieval performance of language models by adequate and novel document priors.

	SvS algorithm
1:	**Input:**
	The posting lists $P = [p_i], 1 \leq i \leq k$
	The query terms $q_1, \ldots, q_k$
2:	**Output:**
	A set D containing the intersection of every $p_i \in P$
3:	Sort the lists $[p_i]$ by size
4:	$D \leftarrow min(P)$
5:	**for** every set p_j in $P \setminus D$,
	selected in increasing order by size
6:	$l \leftarrow 1$
7:	**for** every $e \in D$
8:	$found =$BinarySearch(e,p_j)
9:	**if** !$found$ **then** $D \leftarrow D \setminus \{e\}$ **fi**
10:	Set l to the value of low at the end of BinarySearch.
11:	**end for**
12:	**end for**
13:	**return** D

Table 2.1: SvS algorithm (Demaine et al. [2001])

2.5.2 Querying with an inverted file

Next we discuss how to resolve queries using an inverted file (both boolean and ranked queries). Also, we present some mechanisms for speeding up query evaluation. Whilst during query processing, time minimising sequential accesses was a key factor for the scalability of algorithms, at query time postings are (usually) fetched on their entirety. This fact is controlled by operating system mechanisms, and it is difficult to surpass. However, if the search engine (or part of it) was embedded inside the OS's kernel, there could be other approaches for efficient querying.

2.5.2.1 Efficient boolean querying

The problem of Boolean querying is of special importance, as this model prevails in some Web search engines. We present the case of an `AND` query. The steps to be taken by the search engine when a conjunctive query $q = [q_1, q_2, \ldots, q_k]$ is entered in the system are:

1. Look-up the query terms in the dictionary
2. Inspect the postings file and retrieve the corresponding k postings for the query terms: $[p_1, p_2, \ldots, p_k]$
3. Intersect the k lists.

It is assumed that the postings are document-identifier ordered (as in equation 2.2). The main issue in the ranking process is to compute the intersection between sorted sets. A simple but efficient algorithm is to walk through the lists simultaneously. Let n be the total number of elements in the postings $n = \sum_{i=1}^{k} |p_i|$. Then, the complexity of this linear algorithm is $\Theta(n)$.

There is another method to compute the document result set effectively: the SvS algorithm by Demaine et al. [2001] (figure 2.1). The procedure repeatedly computes the intersection between sorted sets. The technique iteratively intersects the two smallest lists p_i and p_j. Suppose that $|p_i| > |p_j|$. Then, p_i is scanned consecutively by simple binary searching the elements in p_j. If r is the size of the resulting intersection over the k lists, it is possible to prove that the algorithm makes at least $\Omega(r \log(n))$ and has a worst case of $O(n \log(n/k))$. It follows that SvS is not better than linear search in every case. However, binary search is advantageous when the postings can be fitted in main

memory. In any case, document ordering is required for the algorithms discussed so far. Hence, for resolving efficiently Boolean queries, document order is a must. This is even more crucial if postings are not stored in main memory, where non-ordered postings would imply on-disk random accesses and decrease query answering times. The techniques presented are not only employed for Boolean querying, but they are also useful for phrase queries in a positional index, see equation 2.3, proximity search or for document at a time evaluation systems, which are discussed next.

2.5.2.2 Ranked queries

Ranked queries produce an ordered list of documents. As mentioned earlier, every document is assigned a *score*, usually a float point value. Inverted indexes are also suitable for supporting these kinds of queries. Every ranking method presented in the previous section computes this score its own way, but there is a common procedure to all of them. We can use that for our advantage when designing query processing strategies.

Algorithms for resolving ranked queries are cast into two families:

- Term at a Time (TAT)
- Document at a Time (DAT)

The difference between TAT and DAT systems is the way in which they scan the postings: term by term or all the document at once. The outcome of the scoring functions seen so far essentially stems from three different sources:

- A score dependent just on information about the query term, for instance inverse document frequency, or in-query term frequency: $w_q(t)$
- A term-document score (dependent on the contribution of the term on the document): $w(t,d)$
- A document normalisation factor, document dependent but term-independent: W_d

For instance, the pivoted tf-idf scoring function presented before (equation 2.5) can be seen as:

$$w_q(t) = \log \frac{N}{df(t)} \tag{2.22}$$

$$w(t,d) = \frac{\log(1 + tf(t,d))}{\log(1 + avgtf)} \tag{2.23}$$

$$W_d = \sqrt{(1-s) + s(dl)} \tag{2.24}$$

The TAT algorithm is detailed in table 2.2.

Term-at-a-time processing retrieves every posting list for the query terms and computes the partial scores in a set of accumulators A_d. These accumulators have to be stored in memory, and cause a significant overhead if the postings fetched contain many documents. In order to reduce the number of accumulators and operations, there are many *dynamic pruning* strategies (see section 2.5.3). They are especially important in memory-limited environments or in production sites with high demanding query throughput requirements, as it is the case of web search engines. Those strategies are further discussed in section 2.5.3.1.

On the other side, the document-at-a-time strategy loads document-ordered postings and arranges them in a priority queue. Postings are sorted according to the position of the term in each list. The procedure inspects the first document in the posting list at the top of the queue. Then the postings for the query terms are inspected in document order, and exact scores for the documents can be computed on the fly. The procedure needs that the postings be sorted according to their document identifiers.

There is no clear decision about which of these two query processing strategies performs better overall. For the two cases, a number of heuristics and algorithms have been developed in order to shorten response times of IR systems, which are discussed next. These optimisations trim the number of accumulators to be maintained for TAT processing and keep the memory usage low.

	TAT Query evaluation
1:	**Input:**
	The Index I
	The query terms $q_1, \ldots, q_k$
2:	**Output:**
	A set D containing the top r ranked documents
3:	**for** every document d allocate an accumulator $A_d \leftarrow 0$ **end for**
4:	**for** each q_i
5:	$P_i \leftarrow postings(I, q_i)$
7:	Calculate $w_q(q_i)$
8:	**for every document** $d \in P_i$
9:	calculate $w(q_i, d)$
10:	$A_d \leftarrow A_d + w(q_i, d) \times w_q(q_i)$
11:	**end for**
12:	**end for**
13:	Read the array of W_d values
14:	**if** $A_d > 0$ **then** $A_d \leftarrow A_d / W_d$ **fi**
15:	$D \leftarrow top(k, A_d)$
16:	**return** D

Table 2.2: General procedure for the TAT query processing strategy

Complex queries involving positional information are different. In DAT evaluation, all inverted lists are processed simultaneously, eliminating the need to store proximity information in memory. As a rule of thumb, if postings are small enough to be loaded completely into memory, document at a time performs well, and does not need the drawback of maintaining the in-memory accumulator list.

2.5.3 Dynamic Index Pruning

2.5.3.1 Reducing the number of accumulators

Strategies for reducing the number of in-memory accumulators in TAT query processing strategies are based on the fact that very frequent terms are normally of low importance for the retrieval process. This is reflected by state-of-the-art retrieval models, which include *idf* as a core component, and as well by query log analysis, which shows that those terms are not common in web queries. Then, it is a fair claim to assume that most queries will only cover a restricted range of documents, and so the number of accumulators to be maintained will be lower than the total number of documents in the collection N. This motivates the following heuristics, first described by Moffat and Zobel [1996]. In that paper, the authors describe two heuristics called Quit and Continue, that rule the accumulator creation policy.

The first direct approach is to limit the total number of accumulators to a number $L < N$. This check is done at the end of every posting list. As frequent terms are less likely to have a great impact in the final ranking, the algorithm processes the less frequent terms first. This method alters the behaviour of the retrieval process, and even if it appears to be effective in practice, it is less compelling for experimental settings. The Quit heuristic stops query evaluation after the accumulator memory limit is reached (we have processed more than L posting entries). The Continue heuristic goes on with query evaluation but with the accumulator set already allocated so far, without creating new accumulators for newer documents that appear in the ranking process at latter steps. The original study reports that for long queries in TREC-style evaluation a number of accumulators in the order of 1%-5% of N is enough to guarantee a decent retrieval performance.

The original description of Quit-continue strategies presents some problems. First, long lists may exceed the accumulator limit, by requiring the inspection of a number of new documents much higher

than L; if the stopping condition is only checked after scanning the posting list, the burstiness of some terms can result in unpredictable memory consumption figures. However, checking after every posting entry is checked can degrade the performance. Finally, the whole set of methods are reported to be somehow poor performing in the context of Web-style queries (Lester et al. [2005a]) because the effectiveness of these mechanisms is not just dependent on the number of accumulators, but also on the ordering of the documents. Most of these issues can be overcome by discarding some accumulators when they are too small.

Lester et al. [2005a] propose an adaptative pruning approach that tries to overcome these issues in Web-style queries: accumulators are created and discarded as each list is processed, which are conveniently sorted in order of importance as assessed by the term weight. In brief, accumulators are created on demand, and the system decides whether to introduce a new accumulator based on the partial contribution of a term to the final score. The accumulator creation criteria is to check if every posting entry scores higher than a score threshold S, which is conveniently re-calculated at the beginning of every posting list scanning phase. The threshold is initially small and increases during query evaluation, implying that new documents are less likely to be in the ranked result list. A more refined version of this approach is the basis of some static pruning algorithms presented in chapter 5. With this approach, the number of accumulators can be fixed at less than 1% of the number of documents for large Web-like collections without considerable degradation in retrieval performance.

2.5.3.2 Skip lists

Posting lists can be extended with additional structures to enhance the search capabilities. Some of them give support for binary search inside the document arrays, or organise the postings as B-trees or some similar structure. *Skip lists* (Moffat and Zobel [1996]) are probably one of the most extended techniques for fast query evaluation. The approach introduces document identifier descriptors embedded in the postings in compressed format, pointing to a certain position (document identifier) inside the posting list. This way it is possible to jump over long portions of the compressed array in order to speed up evaluation. Postings are subdivided into chunks, and the descriptor allows for deciding if evaluation can skip the current chunk to the next.

In the original formulation for skip list, it is assumed reasonable to place a skip every $\sqrt{df(t)}$ list entries in the posting for term t. The work in Boldi and Vigna [2005a] provides much accurate bound for the number of skips, based on statistical properties of the inverted files. Moreover, they provide a novel variant of the Golomb code for representing the skips encoded, based on the fact that the distribution of skips resembles more a Gaussian than a geometrical function.

2.5.3.3 Document-at-a-time optimisations

The approaches presented next are based on the main assumption that a user only browses through the top ranked documents returned by a search engine.

Brown [1995] presents a technique where the number of candidate documents for scoring is reduced to the set of documents likely to appear in the top k results. The candidate list can be found and scored efficiently by using the skip described before. Basically, the algorithm computes off-line a list containing the top-k results for every possible one-term query. This is very similar to the foundations of the static pruning algorithm presented in section 5.1.1. The query-dependent candidate list is created by merging the query term-specific candidate lists. The author reports excellent speedups for this approach. The main drawback is that when the query terms are of extremely low co-occurrence the candidate-list based scoring can lead to poor effectiveness.

A second optimisation is presented by Broder et al. [2003]. In that work, query evaluation is treated as a two-stage process. The goal is to score k documents. In a first stage the algorithm retrieves and scores the documents containing every query term. If the number of scored documents is greater than k the process stops, otherwise a second query is issued in the normal fashion.

The most important query optimisation technique for document-at-a-time query processing algorithms is the MAX-SCORE algorithm. It was presented by Turtle and Flood [1995] among other

series of query optimisation techniques. Unlike previous approaches, the process is dynamic and guarantees correct results: it returns the exact top k results a full index evaluation would provide.

The algorithm operates as follows. Terms are sorted by frequency of occurrence (posting list length), like in the *Continue* approach. Let the ordered terms be $t_0, t_1, \ldots, t_n$. In a first phase, the algorithm computes a standard DAT evaluation, until k documents have been scored. At this point, the smallest score becomes a lower bound for document scores returned for this query. Query evaluation continues and the algorithm keeps track of this *threshold* k-th minimum score. At some point, the threshold score will be large enough to prove that any document appearing in the top k results must contain some query term other than the most frequent query term t_0. Then, the system only needs to score documents containing at least one of every other query terms $t_1, \ldots, t_n$. For long queries, this process can be applied subsequently to the rest of the terms. When conditions are favourable, the algorithm only needs to score documents that contain a few discriminating terms.

MAX-SCORE makes use of skips in order to achieve top performance, so it can avoid reading long parts of postings to find the next potentially top-scoring document. Experimental results show that query evaluation with MAX-SCORE can save up to 50% of the total query execution cost when the search engine only needs to return the top 10 documents to the user.

Turtle and Flood [1995] also explore a TAT MAX-SCORE procedure. This procedure operates in a similar fashion as a TAT optimisation presented in one of the earliest papers on modern query optimisation by Buckley and Lewit [1985]. The basic idea is the same: at some point during query evaluation, it may be possible to show that it is not necessary to consider any more terms, as the document at rank k+1 cannot surpass the score of the document at rank k.

Another variation of the MAX-SCORE algorithm was presented by Strohman et al. [2005]. In brief, the method combines the MAX-SCORE with candidate list approaches, where these lists serve as a hint for finding top scoring documents.

2.5.3.4 Non doc-id ordered posting lists

There are other ways of organising information inside posting lists in order to provide a faster access. The basic principles for fast query evaluation is to process the potentially highest scores first. Those scores correspond to the posting entries with larger $w_q(t)$ and $w(d,t)$. As mentioned before, a very useful heuristic for both DAT and TAT systems is to begin the evaluation with the smallest postings first, i.e. with those terms with lower df. However, most $tf(t,d)$ values are likely to be small, especially in small postings. Most retrieval models assign scores increasingly with respect to $tf(t,d)$ and decreasingly with respect to df. It is clear that those higher tf values are to be scored first for taking advantage of the early stopping query evaluation strategies mentioned above. Unfortunately, a traditional document-id sorted ordering of the postings does not guarantee that those values come first.

There are two solutions for tackling this issue. The most straightforward option is to organise postings using a frequency-based ordering. As an example, consider the following sequence of document identifiers-term frequency pairs, sorted by document identifier:

$$< (1;5), (5;2), (10;1), (12;2), (14;8), (22;3), (25;1), (33;1) > \tag{2.25}$$

and factoring out the pointers with the same frequency would result in the triples (count, frequency, identifier):

$$< (1;8,14), (1;5;1), (1;3;22), (2;2;5,12), (3;1;10,25,33) > \tag{2.26}$$

It is possible to further apply gap based encoding in order to allow for effective compression. This prevents the re-arranged posting from paying any storage overload in comparison with the associated traditional document-identifier ordered postings.

$$< (1;8,14), (1;5;1), (1;3;22), (2;2;5,7), (3;1;10,15,8) > \tag{2.27}$$

Early stopping with TAT processing becomes simpler. Every list is fetched in turn and evaluation can stop for the term whenever the $w_q(t) \cdot w(d.t)$ is lower than a threshold S. It is possible to

perform DAT evaluation with interleaved evaluation. This implies that the first block of every list is frequency-sorted whilst the rest is identifier sorted.

Posting lists can be sorted in a different fashion. Anh and Moffat [2002] propose that pointers inside postings should be *impact* ordered for further enhanced performance. The basic principle is that $w_q(t) \cdot w(t,d)/W_d$ stands for the real contribution of the term to the final score. Hence, it makes sense to sort posting entries in decreasing $w(t,d)/W_d$ (called impact) order. Since those contributions are not integer values, a suitable transformation and truncation is needed, for retaining compression. Experiments show that around 10 to 30 different impact values suffice for effective evaluation. At query time the system only needs to multiply the impact stored in the index by the corresponding $w_q(t)$ value.

The drawback of both strategies is that they make Boolean matching and the evaluation of passage queries difficult, as the whole posting has to be scanned to determine whether a document belongs to a posting or not.

2.6 Evaluation

There are several retrieval models based on different assumptions, some of them outlined in the previous section. Usually, those models are controlled by parameters, or take into account different types of evidence. Then, *how do we compare between different approaches?*. There exists a bunch of metrics developed to rank retrieval models, and this is actually an active and interesting area of research. None of those metrics is completely satisfactory, because *relevance* is user-dependent and multidimensional, whilst the result of these metrics is a single value.

Evaluation is performed by using a document collection, a set of topics describing a user's information needs and a set of relevance judgements, indicating, for each topic, which documents are (manually annotated) *relevant* for each topic. This judgement is binary (relevant or not relevant) and incomplete, as not every document is classified into the relevant and non-relevant classes. For early collections, such as the Cranfield, the judgement was complete; however, the sizes of modern collections make complete assessment prohibitive.

Such assessments are made in the context of the Text REtrieval Conference Voorhees and Harman [2005]. The basic idea (Sparck-Jones and van Rijsbergen [1976]) is to use the top-ranked results given by several search engines as a document pool for topics. This pool is then manually annotated, instead of doing a full-assessing on the whole document collection.

In order to describe retrieval performance, the metrics calculate which portion of the relevant documents have been retrieved and how early in the rank. The two basic metrics are *precision* and *recall*. The precision P_r of a ranking method at some cutoff point r is the fraction of the top r ranked documents that are relevant to the query. On the contrary, the recall R_r of a method at a value r is the proportion of the total number of relevant documents retrieved in the top r.

In order to provide a fair comparison across multiple precision levels, it is standard to use Average Precision (AP). AP is defined as the arithmetic mean of the precision at all the levels where the relevant documents occur. On the other hand, Reciprocal Rank (RR) is defined as the inverse of the rank of the first relevant document. Those measures can be averaged across a set of topics, resulting in Mean Average Precision (MAP), and Mean Reciprocal Rank MRR. Another measure is R-precision, which stands for the precision when the R relevant documents in the dataset (for a given query) have been retrieved.

Chapter 3

Index Compression

This chapter is devoted to general data compression. After an introduction of the general benefits of compression (section 3.1.1) we present what are the main problems and characteristics of data compression algorithms (section 3.1.2). The following sections move onto traditional compression mechanisms for information retrieval. Specifically, we focus on the particular case of text (section 3.1.3), and on compressing the index structures of a retrieval system: the dictionary (section 3.2) and postings file (section 3.3).

3.1 Compressing Data

3.1.1 Compression In Information Retrieval Systems

Compression mechanisms address the problem of the efficient representation of information through algorithms that transform or *code* information.

IR systems deal with cumbersome amounts of information that need suitable and efficient processing. Compression comes at play in two different stages:

- Compression of the indexed documents.
- Compression of the index itself.

The first benefit of compression is the reduction of consumption of resources, in the case of IR, memory and bandwidth transmission. As shown in the following sections, suitable coding algorithms reduce the size of an inverted file significantly.

There are other advantages for using compression in an IR system, but all of them are based on the fact that it is faster to decompress information on RAM memory than to fetch it from a hard drive. We discussed briefly the drawbacks of modern secondary storage 2.4.4, which is the major bottleneck for any computer program that makes use of huge amounts of data. Disk seek times are related to the performance of the read header when it is to be positioned over a disk platter, which is a mechanical device. Also, increasing platter's bit-per-inch density affects the effective throughput of the disk. In order to avoid the header's movement penalty disks read blocks (contiguous volumes of data) instead of bits.

Modern general-purpose computer systems possess a hierarchy of memory levels, each level with its own cost and performance characteristics. At the lowest level, CPU registers and caches are built with the fastest but most expensive memory. For internal main memory, dynamic random access memory (DRAM) is typical. At a higher level, slower magnetic disks are used for external mass storage, and even slower but larger-capacity devices such as tapes and optical disks are used for archival storage. This growing performance gap between different types of memory access has to be carefully taken into account by the design of computer programs. Even if the program organises the structure of its pattern of memory access (to exploit locality and block-based fetching), the access

gap is still a key issue. Early developments in parallel processing further widen the gap, whilst RAID systems consisting of multiple disks with stripped or replicated data allow for additional bandwidth.

Given a tiered memory scheme, the effectiveness of an IR system that relies on magnetic storage somehow pays through the number of disk operations. Compression is helpful in this case: fetching blocks of compressed data will likely reduce the amount of such operations. Decompression speed is a key factor here: if decompression times are low enough, then I/O savings amortise the cost of decoding data.

Also, compression helps to keep more information in the memory cache. An IR system uses caches at different levels, like RAM, disk, or shared resources in a distributed environment. Their usefulness relies on their ability to exploit frequently or recently used data. The principle is the same as before, the more compressed the data, the more it will fit in the cache.

As a brief summary, compression helps greatly not only to cut off the cost of extra storage, but also to reduce substantially query answering times of an IR system.

Although this work focuses only on index compression, almost every other form of computerised data can be compressed, and hence benefit from reduced sizes. There exist several widespread compression formats. It is common for instance, to find audio encoded in MP3 format, images as PNG and video as MPEG. Lastly, the case of link compression is of interest to web search engines. Although the descriptions presented here can be applied for general compression of integers, the distributions of links in the web graph follows a general Zipfian law. For this particular case, some variations of Elias' δ coding (described below), named ζ codes are discussed by Boldi and Vigna [2005b].

3.1.2 Models and Codes

Compression methods need models of the data to be coded. They are inputted a stream of data in any format and output a set of codewords representing the compressed or coded form of the data. A model predicts *symbols*. The total universe of input symbols is called the *alphabet*. Compression needs a probability distribution for predicting the next symbol to be coded. The process of estimating the probability distribution is known as *modelling*, whereas the process of converting the symbols into bits (codewords) is known as *coding*.

The decoding process reverses the coding operation: it is inputted the stream of codewords and reconstructs the original data using the model. If the data recovered is identical to the original data, the compression method is called *lossless*. If we allow for distortions, or loss of data, the compression method is called *lossy*. We will refer to lossless compression mechanisms in the rest of this chapter, whereas lossy compression mechanisms for IR systems are further discussed in chapter 5.

The number of bits per symbol s, is referred to as the *information content* of the symbol, and it is directly related to the probability distribution p that predicts its occurrence:

$$I(s) = -\log p(s) \tag{3.1}$$

The average amount of information per symbol over the alphabet is known as the *entropy* of the probability distribution:

$$H(s) = \sum_{s} -p(s) \cdot \log p(s) \tag{3.2}$$

Shannon's (Shannon [1948]) theorem states that $H(s)$ is a lower bound in bits for compression, assuming independence between symbols and given a fixed p.

The probability distribution can be estimated in many different ways. A first naive option is to use the same fixed static distribution for every set of input data. That is likely to render very poor performance. It is more accurate to estimate the probabilities from a sample and extrapolate them to the whole dataset. If the compression method uses the same distribution regardless of the data being coded, it is called *static*. If the model is calculated separately for every input dataset to be compressed the method is said to be *semi-static*. The main drawback of semi-static models is that they need to transmit the model to the decoder every time, and that they need a first pass over the data to calculate the model, and a second one to perform the actual coding. Finally, *adaptative* models generate a model on-the-fly, while scanning the data. Adaptative models soften the probabilities of the input symbols on the basis of their number of occurrences.

3.1.3 Text compression

Text compression is a particular case of general data compression. In some situations, static models are a good choice: they are fast (it is not necessary to calculate the model each time) and perform reasonably well if the variability of text files to be coded is not high. However, a model in one language may not perform well in other language, or in a file containing numbers only.

Generally, text compression methods are normally classified into:

- Symbolwise methods
- Dictionary methods

Symbolwise methods work one symbol at a time, estimating the probability of every symbol, and usually assigning shorter codewords to the most frequent symbols. Those symbols can be characters, words, or sequences of n characters/words (also known as n-grams). The size of the context used to encode those symbols decides the *order* of the encoder. A *Zero* order modeller assigns probabilities to isolated symbols, whilst a k-th order modeller assigns those probabilities as a function of the k preceding symbols. The popular Huffman (Huffman [1952]) and arithmetic coding (Rissanen [1979], Witten et al. [1987]) fall into this category. On the other hand, *dictionary* methods replace words and other fragments of text with an index to an entry in a dictionary. They use simple representations to code references to entries in the dictionary. They obtain good compression ratios by conflating several symbols into a single output codeword. Symbolwise methods rely on the good estimation of the symbol's probabilities; for this reason they are also known as *statistical* methods. To compress a symbol, these estimations are likely to use some information about the context in which it occurs. Contrarily, dictionary-based methods create an implicit context by grouping several symbols into a single one.

Symbolwise methods are based on Huffman or arithmetic coding, the former being a bit faster than the latter. Huffman coders construct a coding tree for a given symbol set, given their probability distribution. The same tree is used in the decoding phase. Arithmetic coding is a more sophisticated model that achieves excellent compression ratios, close to the entropy bound. It operates by translating the entire message into one number represented in base b, calculating a lower and upper bound of the sequence of symbols.

The most successful dictionary methods are based on Ziv-Lempel coding. They are a family of adaptative models that share the idea of replacing strings of characters with a reference to a previous character in the string. This controls the growth of the dictionary size, whilst retaining good compression rations and decompression times. Mainly, Ziv and Lempel developed two algorithms, namely LZ77 (Ziv and Lempel [1977] and LZ78 (Ziv and Lempel [1978]), which achieve adaptivity by replacing a substring of text with its previous occurrence. In general, the algorithm maintains a window of size N compressing the last N processed characters, and in each step it reads the longest possible character sequence that also appeared in the window. Lempel-Ziv variants differ on how the pointers are represented, and in the limitations imposed on what the pointers are able to refer to.

The choice of a compression mechanism for text depends on the flexibility/speed imposed. Lempel-Ziv compressors combine fast compression and decompression times with efficient compression ratios. However, as every adaptative method, they do not provide random access to the compressed text (due to the model-learning phase). Character-based Huffman is reported to be poor performing. On the contrary, word-based Huffman (Moffat [1989]) is a zero-order model that provides good compression ratios and efficient random decompression times. For retrieval systems there is an extra beneficial characteristic: the text and index source alphabets are exactly the same.

3.2 Compressing the Lexicon

The lexicon (or dictionary file) of an inverted file records at least the index terms along with a pointer to the terms' on-disk posting. Some of the efficient query processing strategies presented in section 2.5.2, require the terms to be ordered by frequency. In order to allow such an ordering, the lexicon has to store as well those frequency values.

The dictionary has to remain in memory for providing efficient query evaluation. Small space requirements need suitable compression mechanisms that both reduce the size of the structure and allow fast term look-ups. Section 2.4.2.1 presented several data structures for storing the dictionary. If we are not interested in suffix search, and assume that the lexicon fits in memory, dictionary as a string or hashing tables are adequate choices.

Dictionary as a string represents the lexicon as a sequence of characters with delimiters. The structure is accessed in a binary search fashion. A simple structure supporting binary search is a simple in-memory vector with the actual strings and integer values for the frequencies and on-disk posting pointers. This is, however, very space consuming. Further space savings can be achieved if all the terms are concatenated into a long contiguous string accessed through 4-byte pointers.

Taking advantage of the fact that terms are sorted lexicographically, it is possible to get rid of shared prefixes between terms. Front coding is based upon the intuition that contiguous terms are likely to share a common prefix. Thus, it is possible to store two short integers for decompressing a term:

- one indicating the length of the shared prefix
- one indicating the length of the remaining non-shared suffix

In general, front coding reduces space storage requirements in about 40% with respect to full string based dictionaries. This is collection (and language) dependent, as it obviously depends on the prefix distribution (some prefixes are more common than others) and string lengths. Compressing every term in the dictionary with front coding eliminates the possibility of binary searching, because every term needs the preceding term for being decompressed. It is possible to further modify the front coded structure to allow fast look-ups: one term out of four can be stored uncompressed. Binary search is performed over the uncompressed terms, which provide fast access to the rest of the compressed terms. In practice, storing the lexicon in this 3-out-of-4 fashion leads to very compact dictionary representations while still allowing for fast look-ups.

Hashing is a mechanism for mapping a set of keys into a set of integer values. The final representation serves as a *fingerprint* of the data. Let $L = \{x_1, x_2, \ldots, x_n\}$ be the set of keys, or string in the particular case of the dictionary. A hashing function h maps the x_i as:

$$h : L \rightarrow S = [0, m-1] \tag{3.3}$$

$$x_i \rightarrow h(x_i) \tag{3.4}$$

Note that the input of the function is restricted to L, the vocabulary of the collection, although in practise a hash function may have an infinite domain. For that reason, hash functions are usually *not injective* which leads to collisions, i.e., two input values can have the same final representation. The likelihood of collisions in standard hashing function is determined by a load factor $\alpha = n/m$; the smaller α the less likely two keys would collide into the same hash value. Indeed, if n keys are hashed randomly into m slots independently, the probability of no collisions is:

$$\prod_{i=1}^{n} \frac{m-i+1}{m} = \frac{m!}{(m-n)!m^n} \tag{3.5}$$

Equation 3.5 implies that for any reasonable α loading ratios, keys are going to collide, leading to collision-resolving policies, which would degrade the performance of the hashing function. However, it is possible to take advantage of the fact that, in static indexing situations, L is known in advance. Hence, the collisions can be avoided by constructing carefully the hashing function.

A hashing function is said to be *perfect iff* $\forall x_i, x_j \in L, h(x_i) = h(x_i) \Rightarrow i = j$. Furthermore, a hashing function is said to be *minimal iff* $\alpha = 1$ which implies that $m = n$. Minimal perfect hashing functions, guarantee access to a set of keys in $O(1)$ time, and that there are no unused slots. Finally, a hashing function is *order preserving iff* $\forall x_i, x_j \in L, x_i < x_j \Rightarrow h(x_i) < h(x_j)$.

There is a description of how to build a OPMPH (Order Preserving Minimal Perfect Hash) function by Czech et al. [1992], and the MG software, built after Witten et al. [1999], includes a C

Table 3.1: Example unary, γ, δ and Golomb code representations

x	Coding method				
	Unary	γ	δ	Golomb	
				b = 3	b= 6
1	0	0	0	0 0	0 00
2	10	10 0	100 0	0 10	0 01
3	110	10 1	100 1	0 11	0 100
4	1110	110 00	101 00	10 0	0 101
5	11110	110 01	101 01	10 10	0 110
6	111110	110 10	101 10	10 11	0 111
7	1111110	110 11	101 11	110 0	10 00
8	11111110	1110 000	11000 000	110 10	10 01
9	111111110	1110 001	11000 001	110 11	10 100
10	1111111110	1110 010	11000 010	111 00	10 101

implementation[1]. A subtle variation on the algorithm allows for identifying terms that do not belong in the lexicon. The function returns a special value if the key $x_i \notin L$. Those functions are known as *signed* OPMPH and the MG4J software[2] provides a java implementation for computing them. OPMPH are suitable if enough main memory is available for the dictionary, or in a setting where computers are dedicated to the retrieval process. As well, some indexing strategies that require the vocabulary in memory can make use of hashing functions (2-pass in-memory inversion for instance, section 2.4.3).

Perfect hashing functions that allow for dynamic updates are difficult to implement, and need to extra space requirements that could degrade performance. This is not the case for most applications in IR, but if dynamic updates are needed, the method known as *Cuckoo hashing* is a simple substitute for Perfect Hashing (Pagh and Rodler [2001]).

Finally, in some situations it is not possible to hold the lexicon in memory, due to high memory limitations. Some examples are search based on desktop computers, or small mobile devices, like PDAs, or cell phones. In order to reduce the number of terms to be kept in memory it is possible to keep the first term of every on-disk block. This way the size of the dictionary is reduced, at the payload of one disk access for fetching the information of each term. More general structures, like B-trees with leafs on disk can further enhance term look-ups. The techniques presented in section 5.3 reduce the number of indexed terms, and are especially suited to address this kind of situations.

3.3 Compressing the Postings

Posting lists comprise the majority of the data in an index. The size of the postings file can be cut down greatly if it is compressed by suitable models and coding methods.

Recall that all the components in a posting (equations 2.2, 2.3) are integer values. A first obvious choice would be to consider every possible document identifier, term frequency, or position as an integer in the range $[0, N]$, and treat those values as a symbol set and apply standard modelling/coding. This approach has two main drawbacks: the space-cost of storing the model itself and the time-cost of decompressing the postings with the model. Even though a mixed approach could be feasible for some kind of indexes (like schema independent indexes, where everything is treated as a within-collection offset), specific compression algorithms for the postings work much better in most situations. We focus now on document-identifier (d_i) compression, which uses differences or *d-gaps* between consecutive identifiers; however the case of compressing positions is identical. The term frequencies are also compressed with model-fixed algorithms (just like the document identifiers or positions, but without taking into account the differences). However, the fact that most frequency values are very low somehow flattens the differences of effectiveness of different compression mechanisms.

[1] `http://www.cs.mu.oz.au/mg/` (accessed on 03-04-08)
[2] `http://mg4j.dsi.unimi.it/` (accessed on 03-04-08)

Without any loss of generality, it is possible to rearrange the posting entries in an ascending manner, as stated by equation 2.2. This is the format considered in the rest of the chapter. Almost every compression method presented next codes an integer n in $O(\log n)$ bits, so coding the differences between document identifiers will yield better compression performance. The aforementioned differences are also know as *document gaps*, or in short *d-gaps*. As an example, the list

$$< 12; 1, 2, 5, 6, 8, 10, 13, 17, 20, 50, 60, 70 > \tag{3.6}$$

results in

$$< 12; 1, 1, 3, 1, 2, 2, 3, 4, 3, 30, 10, 10 > \tag{3.7}$$

There is extensive literature on how to describe the document gap distribution. In section 4.1 we analyse some of the properties and assumptions in real data, that shed some light upon the inverted file rearranging techniques of chapter 4.

Next are described many popular choices for information retrieval systems. In general, models can be grouped into two broad classes: *local* and *global*. The difference between them is that global methods share the compression model for every posting list and local methods use information on the term's posting for modelling. Local methods tend to outperform global methods in compression effectiveness.

3.3.1 Global Models

Global models can use some information coming from the collection, like number of pointers, number of documents, and so on (*parametrised* models), or be totally fixed and collection-independent (*non parametrised* models).

Recall that by Shannon's law, equation 3.2 the relationship between ideal code length l_x and the probability distribution over the alphabet p:

$$l_x = -\log p(x) \tag{3.8}$$

In our case, $p(x)$ could be stated as a short version of $p(gap = x)$. If N is the number of documents in the collection, encoding the d-gaps in binary would require $\lceil \log N \rceil$ bits per gap exactly. This implies a uniform gap distribution over the postings:

$$p(x) = \frac{1}{N} \ , \tag{3.9}$$

which is not an accurate distribution because in reality small gaps occur frequently (see section 4.1 for a detailed explanation). The first simple compression method is the *unary* code, which codes every integer x with $x - 1$ 1s followed by a 0. The probability in this case is

$$p_u(x) = 2^{-x} \ , \tag{3.10}$$

which implies that $l_x = x$. Although the bias towards short gaps is too extreme for coding d-gaps in realistic situations, unary coding is part of other compression schemes (described next) and can be a suitable choice for coding term frequencies, or at some stages in the indexing process.

The next codes imply probability distributions that draw d-gap occurrences somewhere between the uniform (binary coding) and exponential (unary). Elias' γ codes (Elias [1975]) are among the first of them, which represent an integer x in two parts:

- $\lfloor \log x \rfloor + 1$ is coded in unary
- $x - 2^{\lfloor \log x \rfloor}$ is coded in binary using $\lfloor \log x \rfloor$ bits

Decompression operates as follows: first it reads the number of bits needed to code x, in unary, and then the actual value for x, coded in binary. The estimated d-gap length $l_x(\gamma)$ is $1 + 2\lfloor \log x \rfloor$ bits, and the probability implied is approximated by:

$$p_\gamma(x) = \frac{1}{2x^2} \tag{3.11}$$

Another variant on unary coding are Elias' δ codes (Elias [1975]), which are γ codes where the prefix is coded with γ instead of unary. In this particular case,

$$l_x(\delta) = 1 + 2\lceil \log(1 + \lceil \log x \rceil) \rceil + \lceil \log x \rceil = 1 + 2\lceil \log \log 2x \rceil + \lceil \log x \rceil \tag{3.12}$$

and the associated probability distribution is

$$p_\delta(x) = \frac{1}{2x(\log x)^2} \tag{3.13}$$

A first way of introducing a parameter into the model is to make use of the density of pointers. If P is the total number of posting entries, and T the number of posting (terms) in the collection, the probability that a randomly selected document contains a randomly selected term is $p = P/(N \cdot T)$.

Assuming a random draw of terms into documents the probability of a gap of size x for a term t, is the probability of $x - 1$ non-occurrences of t followed by an occurrence with probability p. Then,

$$p_G(x) = (1 - p)^{x-1} \cdot p \tag{3.14}$$

This particular modelling is assuming that every term-document pair occurs randomly, they are *Bernoulli trials*, and the probability distribution implied is called *geometric*.

The coding scheme that incorporates a geometric distribution is called *Golomb* coding (Golomb [1966]).The method requires a parameter, b, that controls the value p. The algorithm codes:

- $q + 1$ in unary, where $q = \lfloor \frac{x-1}{b} \rfloor$
- $x - qb - 1$ in binary, with $\lceil \log b \rceil$ or $\lfloor \log b \rfloor$ bits.

Hence, the expected codeword length for an integer x is

$$l_x(G) = 1 + \left\lfloor \frac{x - 1}{b} \right\rfloor + \lfloor \log b \rfloor \tag{3.15}$$

Gallager and van Voorhis [1975] proved that if b is chosen such as

$$(1 - p)^b + (1 - p)^{b+1} \leq 1 < (1 - p)^{b-1} + (1 - p)^b \,, \tag{3.16}$$

then Golomb coding generates an optimal prefix-free code for the geometrical distribution corresponding to Bernoulli trials with probability of success given by p. Solving the equation,

$$b = \left\lceil -\frac{\log(2 - p)}{\log(1 - p)} \right\rceil \tag{3.17}$$

Assuming that $p = P/(N \cdot T) \ll 1$, then finally

$$b \approx \frac{\log_e 2}{p} \approx 0.69 \cdot \frac{N \cdot T}{P} \tag{3.18}$$

If b happens to be a power of 2, then Golomb codes are known as Rice codes (Rice [1979]) which are convenient for use on a computer, since multiplication and division by 2 can be implemented using a bitshift operation, which can be performed extremely quickly.

It is possible to generalise the unary, γ and Golomb codes as follows: given a vector $V =< v_i >$, there is a first component which codes $k + 1$ in unary, with k such as

$$\sum_{i=1}^{k} v_i < x \leq \sum_{i=1}^{k+1} v_i \,, \tag{3.19}$$

followed by a binary code of r using $\lceil \log v_k \rceil$ bits, where

$$r = x - \sum_{i=1}^{k} v_i - 1 \tag{3.20}$$

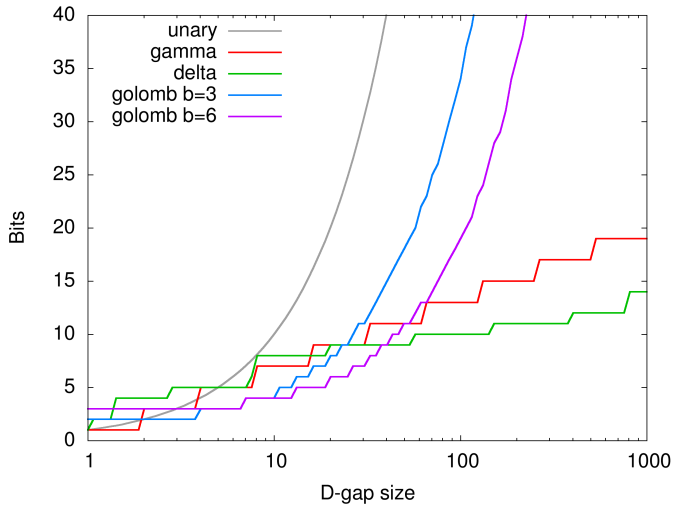


Figure 3.1: Bits required by unary, γ, δ and Golomb coding with $b = 3$ and $b = 6$

Under the vector framework just presented, the unary code uses the vector:

$$V_u =< 1, 1, 1, 1, 1, \cdots > \tag{3.21}$$

γ encoding uses the vector:

$$V_\gamma =< 1, 2, 4, 8, 16, \cdots > \tag{3.22}$$

and Golomb coding uses the vector:

$$V_g =< b, b, b, b, b, \cdots > \tag{3.23}$$

Table 3.1 exemplifies some of the coding methods mentioned in this section. Figure 3.1 is a plot of the number of bits required to code a given integer, for several global coding schemes. Finally, figure 3.2 plots the number of bits required by Golomb coding when the b modulus varies.

3.3.2 Local Models

Compression schemes can make use of per-term information, more concretely the term's document frequency df, which is effectively the number of pointers in the term's posting.

A first option is to draw a different Bernoulli model for every posting list, instead of using the same global model for all of them, based on the observed term frequency. Then, for every term t with document frequency df the code calculates b_t as in equation 3.17, using the local frequency information df. Given so, $p = \frac{df}{N}$ and thus

$$p_{lg}(x) = \left(1 - \frac{df}{N}\right)^{x-1} \cdot \frac{df}{N} \tag{3.24}$$

A safe approximation for b in this case is

$$b \approx \frac{df}{N} - 1 \tag{3.25}$$

The code needs a slight overhead for decompression, as most local models, due to the fact that the Golomb modulus b needs to be calculated for every posting.

Local Golomb only performs marginally better then γ or δ, because terms tend to appear in clumps of documents. Thus, there are other functions that capture better this notion of *burstiness*. It is

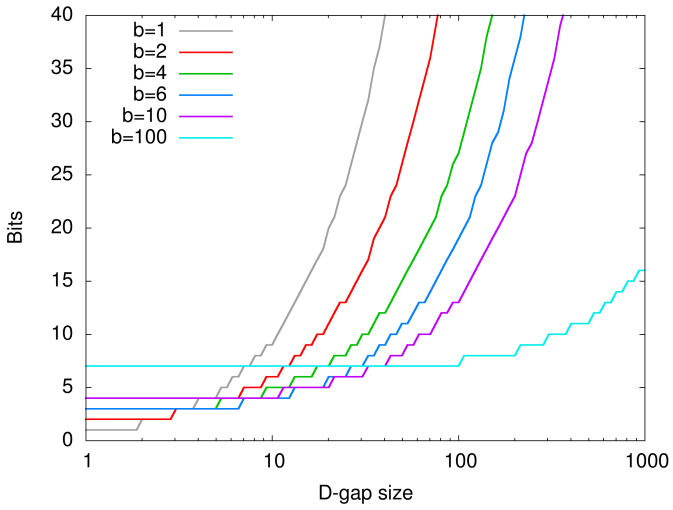


Figure 3.2: Bits required by Golomb coding for different values of the modulus b

possible to distort somehow the geometric distribution to take into account the fact that small gaps appear more often. This goal is easily achieved with a certain mix between γ and Golomb. Using the vector notation, it is easy to see that this distribution creates larger buckets for the selector, hence, creating a *skewed Bernoulli* distribution.

$$V_s =< b, 2b, \cdots, 2^i b, \cdots > \tag{3.26}$$

A good value for b is the median gap size in each posting. This implies that half of the gaps will fall in the first bucket (compressed with just one bit). The selector must be stored in each posting for decompression times to be acceptable.

Another local model uses a *hyperbolic* distribution:

$$p_h(x) = \frac{1}{(\log_e(m+1)+\phi)\cdot x}, \text{ for } x = 1, 2, \cdots, m \tag{3.27}$$

and where ϕ is Euler's constant, $\approx 0.577215...$ The value of m can be set to the maximum gap occurring in a posting, but it is more robust to determine m by the average expected size:

$$\sum_{x=1}^{m} x \cdot p(x) = \frac{m}{\log_e(m+1)+\phi} = \frac{N}{df} \tag{3.28}$$

The last equation can be solved iteratively. The main drawback of this model is that it requires arithmetic encoding, as there is no known Huffman representation for it.

3.3.2.1 Interpolative coding

Interpolative coding by Moffat and Stuiver [2000] is a context-sensitive model that takes into account explicitly that terms in a collection appear as *clumps* or *clusters*. Unlike methods presented before, which are zero-order, interpolative coding is able to code sequences of consecutive identifiers with a few bits. Instead of considering the d-gaps sequence from left to right, compression and decompression are performed in a recursive fashion. Consider a d-gap list $< a_1, a_2, ..., a_n >$. Interpolative coding works by splitting the list into two sublists $< a_1, ..., a_{\lfloor \frac{n}{2} \rfloor - 1} >$ and $< a_{\lfloor \frac{n}{2} \rfloor + 1, ..., a_n} >$. The middle element, $a_{\lfloor \frac{n}{2} \rfloor}$ is compressed using standard binary encoding, and using the first and last elements of the list as a bound for limiting the number of bits. A minimal binary encoding would require $\lfloor \log(a_n - a_1 + 1) \rfloor$ bits. The process continues recursively with both sublists.

```
    Variable Byte Encode(integer x)
1:  while x ≥ 128 do
2:      write( (x & 127) + 128))
3:      x = x ≫ 7
4:  end while
5:  write( (x & 127) + 128))
```

Figure 3.3: Variable byte coding algorithm

```
    Variable Byte Decode
1:  byte shift = 0
2:  byte readByte = read()
3:  integer result = readByte & 0x7F
4:  while readByte < 0 do
5:      readByte = read()
6:      result = result + (readByte & 127) ≪ shift
7:      shift = shift + 7
8:  end while
9:  return result
```

Figure 3.4: Variable byte decoding algorithm

The behaviour of interpolative coding is nearly as efficient as the Golomb code in the worst case, although it leads to greater gains in terms of compression when coping with contiguous sets of integers. The cost of coding f integers in the range $1 \dots N$ is never more than $f(2.5783 + \log \frac{N}{f})$ although this is an underestimation and generally this number is lower (Moffat and Stuiver [2000]).

3.3.3 Byte-aligned coding schemes

All the coding methods presented so far produce outputs whose lengths vary in number of bytes. They need bit level operations, which sometimes are not so gracefully handled by CPUs. Hence, several methods that read/write whole bytes have been proposed. The main claim is that decompression is faster because operations over bytes require less CPU cycles that bit based operations.

All the methods presented next are based upon such assumption. Their main drawback is that bit based compression schemes produce more compact indexes; however this fact is usually eclipsed by the lower decompression times of byte-based coding methods.

The most well-known byte-based method is variable byte, by Scholer et al. [2002]. It is the compression standard for many retrieval systems, due to its claimed better decompression performance. Variable byte compresses an integer using 7 bits as binary data and 1 bit as a continuation flag, indicating whether more data is to follow or not. The algorithms for compressing data in variable byte form are detailed in tables 3.3 and 3.4.

Another family of word-aligned coding algorithms has been proposed by Moffat and Anh [2005]. In brief, they applied the byte-based encoding to larger or smaller units than bytes: 32-bit words, 16-bit words, and 4-bit words. They combine several postings, compressed in 32, 16 or 4 bits, into a single group which is finally encoded using 32 bits (or the number of bits of the CPU's registry). Larger words imply less bit manipulation for compressing/decompressing than shorter words. This approach provides better compression ratios than plain variable byte encoding, while giving approximately the same decompression time.

In general, byte-based encoding mechanisms offer a good compromise between compression ratio and speed of decompression.

We validate the performance of some of the methods disscussed in this section presenting its

Collection	Size	Terms	Documents	Lexicon Size
Disks 4&5	1.9G	521,469	528,155	29M
WT2G	2G	1,002,586	247,491	40M
WT10G	10G	3,140,837	1,692,096	138M
.GOV	18.1G	2,788,457	1,247,753	128M

Table 3.2: Description of the collections employed for measuring compression effectiveness

Method	Disks 4&5		WT2G		WT10G		GOV	
	bpg	size	bpg	size	bpg	size	bpg	size
Unary	1,772	24.36G	1,882	16.68G	1,0753	513G	6,557	288G
γ	6.59	93M	6.21	57M	8.13	333M	8.16	368M
δ	6.24	88M	5.90	54M	7.37	302M	7.43	335M
Global Golomb	12.74	180M	12.80	117M	14.89	610M	13.92	628M
Variable Byte	9.03	128M	9.05	83M	9.65	395M	9.48	428M
Local Golomb	5.89	83M	6.44	59M	7.04	288M	6.22	281M
Skewed Golomb	5.96	84M	6.27	57M	7.07	290M	6.50	294M
Interpolative	5.04	73M	4.88	46M	6.03	252M	5.98	273M

Table 3.3: Average number of bits per document gap and index file sizes for different TREC collections

compression figures obtained in four standard TREC collections, varying in size and number of documents. Table 3.2 describes the collections: size refers to the collection size in raw-text format, and lexicon size refers to the size of the dictionary when every term is represented in an uncompressed form.

Table 3.3 and table 3.4 show the values for the document gaps and frequencies respectively. As expected, most frequency values are low (most of them being 1) and simply non-parameterised codes perform very well. D-gaps behave differently, and local/skewed Golomb codes (the cost of storing the selector b for each posting is not included) and interpolative coding are the most effective methods. Global Golomb is not able to capture the *true* distribution of d-gaps among postings, losing in performance to δ and γ. Variable byte is a bit wasteful with respect to other codes, but the increase in bits-per-gap is allegedly counterbalanced by its fast decoding times.

3.3.4 Compression for superscalar computers

Variable byte or word aligned codes are considered as a standard for decompression effectiveness and widely spread over commercial and free databases (DB) and IR engines. However, it is possible to take into account design of modern computers and CPU architectures to push forward decompression speed. Indeed, some algorithms are grounded on the fact that superscalar CPUs issue more than one instruction per clock cycle (IPC), which are dispatched into several pipelines. A parallel processor achieves maximum performance when none of the following happens:

- one instruction A does not need the data generated by other instruction B (or A would have to wait for B to finish)

Method	Disks 4&5		WT2G		WT10G		GOV	
	bpg	size	bpg	size	bpg	size	bpg	size
Unary	2.29	33M	3.44	32M	2.22	91M	3.76	170M
γ	2.03	29M	2.36	22M	1.90	78M	2.46	111M
δ	2.64	38M	2.91	27M	2.52	104M	2.97	135M
Variable Byte	8.00	113M	8.01	73M	8.00	328M	8.02	362M

Table 3.4: Average number of bits per frequency pointer and index file sizes for different TREC collections

Table 3.5: Variables controlling the query throughput

Symbol	meaning	Typical value
B	I/O bandwidth	0.3 GB/s (modern RAID systems)
r	compression ratio	4
Q	query bandwidth	0.6 GB/s
C	decompression bandwidth	200-500 MB/s
R	results bandwidth	-

- usage of control instructions is minimised, for instance `if A then B else C` control sequences or dereferenced calls (to improve parallelism, modern CPUs employ *branching prediction* techniques to predict the outcome of branch `A` based on past behaviour).

Why and when are faster decompression algorithms *that* necessary? In the introduction of this chapter, we presented briefly some of the factors that affect decompression performance, mostly the I/O bottleneck introduced by a layered memory architecture and secondary storage. Compression helps to alleviate this bottleneck, but only if the cost of decoding does not exceed the disk I/O transfer time. That situation can be issued by a simple analytical model (Zukowski et al. [2006]) that takes into account the variables presented in table 3.5. The model tries to reflect the expected throughput R of an IR engine in terms of the number of Gigabytes per second of uncompressed data a ranking algorithm is able to process. This would ideally frame the situations where whether disk transfer rate (I/O) or CPU usage bounds the outcome of the system. Hence, the model allows to figure out which are the situations where faster compression mechanisms are really needed. The numbers reported are related to a typical IR scenario where I/O transfer stands as the main bottleneck.

It is assumed that queries are entered the system at rate Q, and data is compressed at a r ratio (25% typically). A particular decompression algorithm is able to decompress data at rate C and results are delivered at rate R.

$$R = \begin{cases} B \cdot r & \text{if } \frac{B \cdot r}{C} + \frac{B \cdot r}{Q} \leq 1 \quad \text{(I/O bound)} \\ \frac{Q \cdot C}{Q + C} & \text{if } \frac{B \cdot r}{C} + \frac{B \cdot r}{Q} > 1 \quad \text{(CPU bound)} \end{cases} \tag{3.29}$$

We are interested in the first case, that is, when the I/O transfer ratio is the main bottleneck for query throughput. $\frac{B \cdot r}{C} \cdot 100$ is the percentage of time spent by the CPU on decompression. For the figures reported, it would be necessary 2GB/s $<$ C $\leq$ 6GB/s to only use about 20-50% CPU time for decompression. Note that B is usually greater than $0.3GB/s$ in commercial settings, where disks data is usually stripped in RAIDs[3]. In a modern desktop computer running linux OS without RAID, this is about $B \approx 0.05$.

In summary, there are some situations where hardware configuration advocates for decompression algorithms that operate as fast as possible (instead of maximising the compression ratio). Moreover, ranking schemes involve several operations that can be CPU-intensive, and thus the decompression algorithm should ideally leave the processor available enough time. High decompression algorithms for superscalar computers are based on the following guidelines (Zukowski et al. [2006]), in order to create *vectorised* decompression algorithms:

- decompress small arrays of values in a tight loop
- avoid `if then else` instructions
- keep loop iterations independent of each other

PFOR and PDICT (Zukowski et al. [2006]) are two examples of such an algorithm, which also come as a refinement from the prefix-suppression compression framework (Goldstein et al. [1998]). Both algorithms outperform word-based encoding in both compressing and decompressing performance, when ran on highly parallelisable CPUs.

[3] In linux B can be measured using the *bonnie* command

Chapter 4

Assignment of Document Identifiers

This chapter presents several contributions to a novel technique for compressing an index without any loss of information. Chapter 3 provides an overview of the standard index compression techniques and some insights on why they are suitable for IR indexes. The mechanism presented here, *assignment of document identifiers*, is based on the fact that for most applications, the document identifiers (d_i in equation 2.2) are *meaningless*. Given so, it is possible to *map* every identifier d_i to a new one so that the index is more compressed and retrieval quality is not affected. How to perform such a map and why it should work, depends on characteristics of text collections and how terms are scattered through documents.

The remainder of this section is organised as follows: section 4.1 presents some analytical and experimental results to try to shed light on index d-gap and terms' distributions. Section 4.2 presents the document reassignment problem. Section 4.3 characterises a simpler case of problem and proves its NP completeness. The rest of the section is devoted to novel techniques and combinations based on the insights just presented in order to devise effective/efficient solutions to the problem. Section 4.4 presents prior work on the problem. Section 4.5 describes a graph-oriented solution based on the *Travelling Salesman Problem* (TSP). Section 4.6 shows the way of reducing the dimension of a document similarity matrix by computing its *Singular Value Decomposition* (SVD) and its application to the reassignment problem. Section 4.7 applies the SVD technique again, but right after dividing the original problem into subproblems. Section 4.8 introduces clustering solutions and a technique that combines TSP and clustering. Implementations, experiments and results of each technique are included in their corresponding section. The chapter continues with the discussion and future work in section 4.9, and ends with a conclusions section.

4.1 Analytical form for the d-gap distribution

Lossless encoding mechanisms are of utmost importance for compression, in order to achieve reasonable sizes for index structures. Understanding how a compression model works has led to development of higher performing methods, like interpolative coding (section 3.3.2.1), because they exploit particularly well certain properties of the data at hand.

Section 3.3.1 presented Golomb coding, a parametric scheme that is build upon the fact that occurrences of (term,document) pairs inside a posting are random. However, this only an assumption. The basis of Golomb coding are revisited next, to exemplify the properties of inverted lists' organisation. The insights presented will lead to the formalisation of the assignment problem and its precise characterisation.

For every term t in the collection, Golomb coding makes an independence assumption on draws of occurrences of terms into documents. Consider a collection of N documents.

For every term t, occurring in df documents, let X_i be the random variable describing event of t

occurring in the *document with identifier* i, d_i. More formally

$$\begin{cases} Pr(X_i = 0) = 1 - p & \text{if } t \text{ not occurs in } d_i \\ Pr(X_i = 1) = p & \text{if } t \text{ occurs in } d_i \end{cases} \tag{4.1}$$

The model assumes that every two events X_i and X_j are independent

$$Pr(X_i \cap X_j) = Pr(X_i) \cdot Pr(X_j) \; \forall i, j < N \tag{4.2}$$

This independence leads to considering (term,document) pair occurrences as independent Bernoulli trials, which can be drawn at posting level (local) or collection level (global). The d-gap probability in a posting list for a given term, is directly linked to the number of trials needed to get one success, which is governed by a geometrical distribution:

$$Pr(dgap = x) = (1 - p)^{x-1} \cdot p \tag{4.3}$$

The geometric distribution is memoryless, which means that the conditional probability distribution over the trials after the first success is independent: the number of failures does not affect the odd of the next outcome.

The gap probability p may be estimated by considering statistics global or local to the posting, but assuming a random model with independence (Bernoulli trials) implies that the resulting probability distribution is always geometrical. It follows a simple formulation to calculate the estimations according to a global and local geometric distribution. In the latter we assume a Zipfian model of the term frequency distribution. Finally, we compare the divergence from the estimated geometric model and the actual distributions in a test collection, and explain the reason of this divergence.

The probability of a gap of size x, can be calculated using the total probability law and summing across all the terms (the total number of terms is T) in the collection $\mathcal{C}$:

$$Pr(dgap = x) = \sum_{i=1}^{T} Pr(g = x | n_i) \cdot Pr(n_i) \; , \tag{4.4}$$

where $Pr(n_i)$ is the probability of the i-th term in the collection.

Considering p the same for every document (global model), and approximating p using equation 3.18, the final estimation is determined by equation 4.3.

Estimating p_i in a different way (local model) for every term will still assume that the d-gaps follow a decreasing exponential function,

$$Pr(dgap = x) = \sum_{i=1}^{T} (1 - p_i)^{x-1} \cdot p_i \cdot Pr(n_i) \; , \tag{4.5}$$

because p_i and $Pr(n_i)$ are taken to be independent of x.

Such an estimation typically assumes that the distribution of frequencies of terms in the collection follows Zipf's law, in which the relative document frequency of the n-th ranked item is given by the Zeta distribution $1/(n_i^{\alpha}\zeta(\alpha))$, where $\zeta(\alpha)$ is Riemann's zeta function, and the parameter $\alpha > 1$ indexes the members of this family of probability distributions. Reasonable values of α for indexed natural language text are between 1 and 1.4.

We assume without any loss of generality that the identifier of the $i-th$ most frequent term in the collection is i. It follows that the expected document frequency for the i-th term is approximated as:

$$\hat{df}(n_i) \approx \frac{P}{\zeta(\alpha) i^{\alpha}} \tag{4.6}$$

where P is the total number of posting entries in the collection. Then, $p_i = \hat{df}(n_i)/N$ with N the total number of documents in $\mathcal{C}$. Finally, the probability of a term can be approximated by $\hat{df}(n_i)/P$, and thus $Pr(dgap = x) \in \Theta(a^x)$, with $a < 1$.

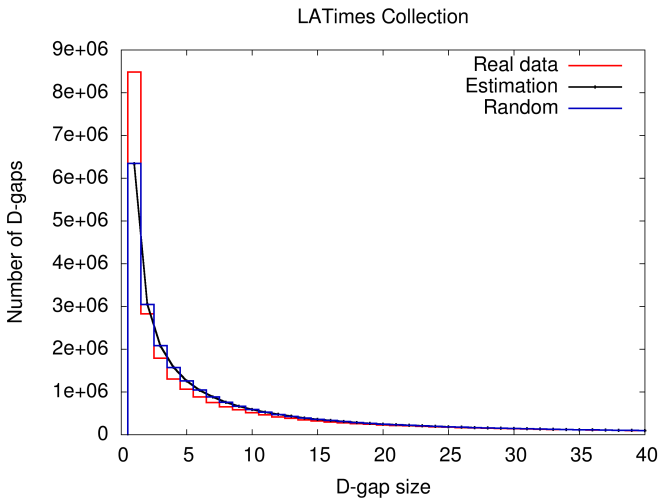


Figure 4.1: Estimated geometrical gap distribution (independent Bernoulli trials), real distribution and randomised distribution, LATimes collection

Note that it could be possible to try to find a closed form for the gap distribution, dependent on the Zipfian exponent α. This way we could have a predicted local distribution for any collection with just knowing T, P, N and α. However, this is not needed for this analysis. The inverted file already contains the real values for $df(n_i)$, which are used to compute the estimations.

Figure 4.1 shows the estimated total number of gaps and their real distribution for the LATimes collection. Figure 4.2 is a series of plots for several terms in the LATimes collection. The terms span a wide range of frequencies. Plots reveal a discrepancy between the real model and the geometrical model, which can be more notorious in mid-frequency terms. Figures also present a *random assignment* of the collection in which every document identifier has been mapped to a new random document identifier. The figure for the total gap values in the collection resembles an exponential curve. However, from the individual plots it is clear that not all the postings are suited to the distribution. The geometrical model is well-suited when the data, (term,document) pair occurrences inside postings, is randomly scattered through the whole document identifiers range, and hence the p_i probabilities are estimated accurately. For textual collections, this is not the case, as seen in the graphs.

The d-gaps are not totally randomly distributed over the posting lists because occurrences of terms in documents are obviously dependent on the d-gap assignment: some sets of d-gaps are more grouped than others in the postings. Before going further into that issue, it could seem reasonable that the *random reordering* of the collection would eliminate that effect. Figures 4.1 and 4.2 also show the values for the number of dgaps of a certain size, given the randomised assignment. Even though the distributions get closer to each other, the random-predicted distribution (local Golomb model) and the randomised distribution still diverge.

It is possible to quantify the divergence between both distributions using the well-known Kullback-Leibler divergence. For probability distributions P (representing the data) and Q (representing the theory, or the estimation over P) the KL-divergence of Q from P is defined to be:

$$D_{kl}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)} \tag{4.7}$$

KL-divergence is connected with Shannon's entropy, equation 3.2, as

$$D_{kl}(P||Q) = -\sum_i P(i) \log Q(i) + \sum_i P(i) \log P(i) \tag{4.8}$$

$$= H(P,Q) - H(P) \tag{4.9}$$

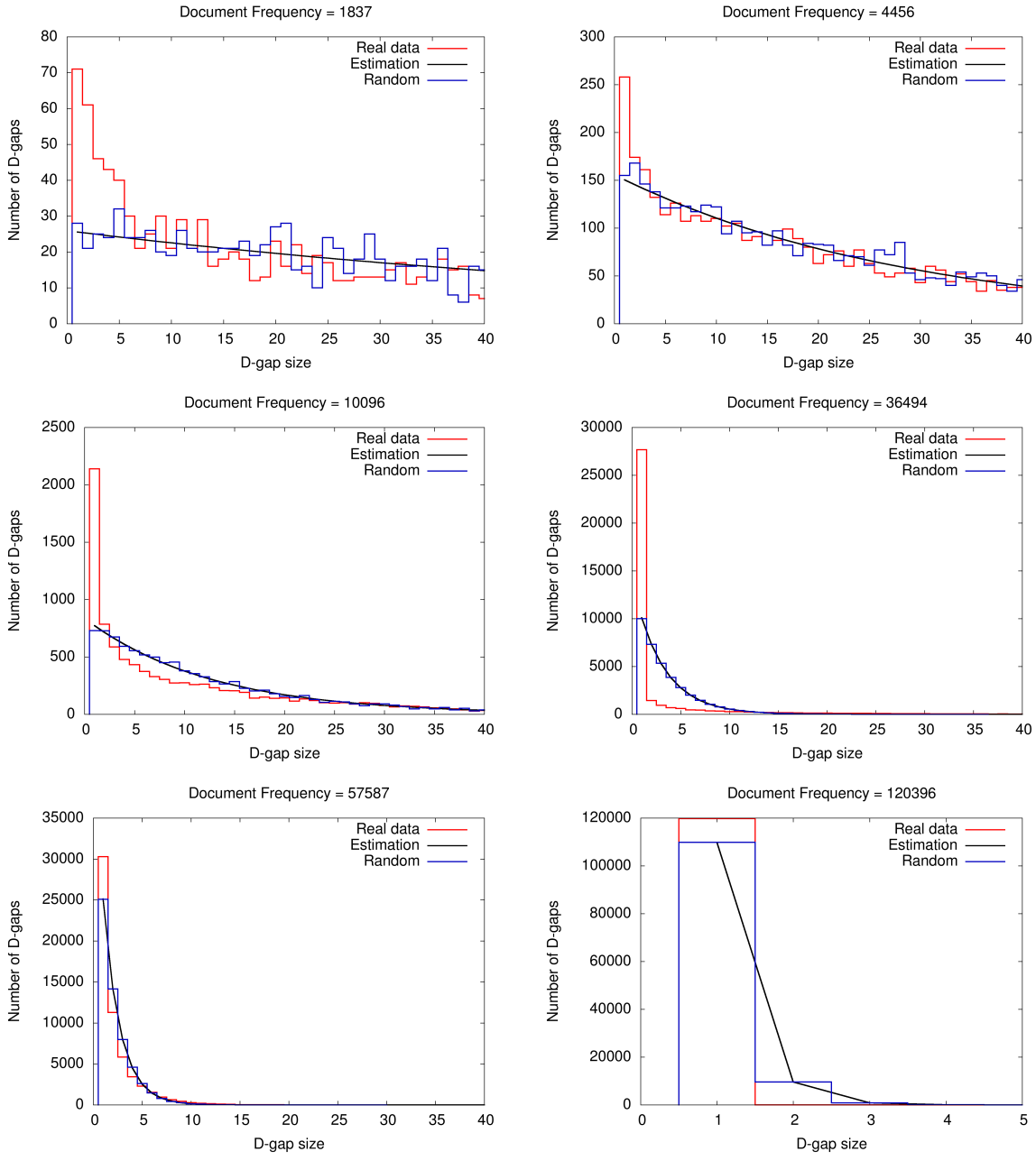


Figure 4.2: Estimated geometrical gap distribution (independent Bernoulli trials), real distribution and randomised distribution, single terms of varying document frequency (df)

Collection	Average KL	Variance	Correlation
LATimes	0.21	0.001	0.36
Randomised LATimes	0.13	$3.6 \cdot 10^{-4}$	0.14
.GOV	0.46	$7.8 \cdot 10^{-4}$	0.22
Randomised .GOV	0.41	$5.6 \cdot 10^{-4}$	0.17

Table 4.1: Average KL-divergence for the terms in the LATimes and .GOV collections, variance and correlation between divergence and posting list size. Statistics are calculated considering every posting as a different d-gap distribution, and averaging over all of them.

where $H(P, Q)$ is called *crossed entropy*. Table 4.1 summarises the values for the LATimes and .GOV collections and randomised LATimes collection for terms with $df > 1000$, where average and variance refer to those of considering every posting list as a different d-gap distribution itself. The table also shows Pearson's correlation coefficient between posting list size (df) and KL-divergence (Q would be local geometrical model estimated with 4.5 and $df(n_i)$ is taken as the real document frequency of term n_i in the collection (instead of using a Zipfian approximation).

Randomising the collections lowers the divergence between the expected and real distribution; however, to some extent, the *non-randomness* is still present. Moreover, there is a slight positive correlation between divergence and posting size, although not being too strong (Pearson's coefficient between 0.1 and 0.4). This behaviour is better noticed in figure 4.3, where the distribution implied by the random document identifier assignment gets *closer* to the random predicted distribution. However, even the divergence versus d-gap size pattern of the original collection order and randomised ordering exhibit a similar pattern.

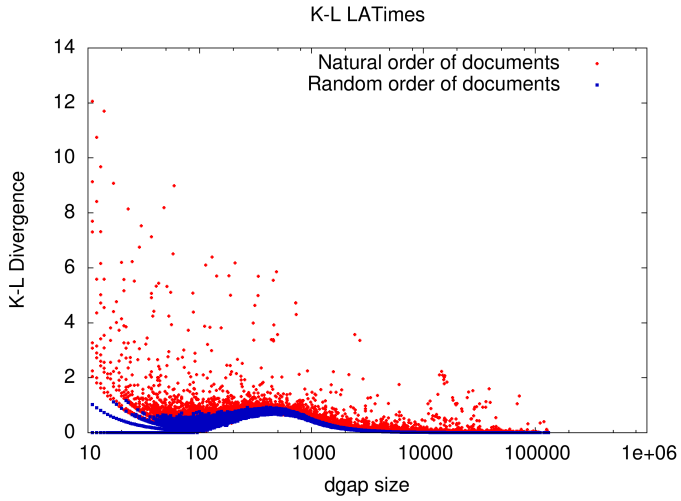


Figure 4.3: KL Divergence for the d-gap distribution, LATimes collection

Table 4.2 shows the most divergent terms from the geometric distribution, for the LATimes and .GOV collections.

Interestingly, the stemmed forms for the months are very divergent with respect to a geometric distribution. Intuition suggests that these terms may occur in *bursts* in the text: the document identifiers are indeed reflecting the fact that data is bounded in time. Also, the term `kuwait` is likely to appear subsequently in documents sorted by date, when the newspaper reported on some news related to 92's war in Iraq.

The burstiness of the data appears in both a newspaper collection and a Web collection. Top terms are reduced forms of the names of the days of the week and numbers very likely to appear in dates. This suggests that even the .GOV exhibits some kind of ordering, mostly due to crawling strategies. Finally, figure 4.4 is a plot of two small terms ($df \approx 2000$) in the LATimes collection, with respect to

LATimes ($df > 10^3$)	.GOV ($df > 10^4$)
bulldog	sat
calendar	sun
februari	feb
august	fri
octob	jan
decemb	5d
touchdown	thu
januari	wed
septemb	03
kuwait	31

Table 4.2: Stemmed forms of some of the most divergent terms from the geometric distribution in the LATimes and .GOV collections

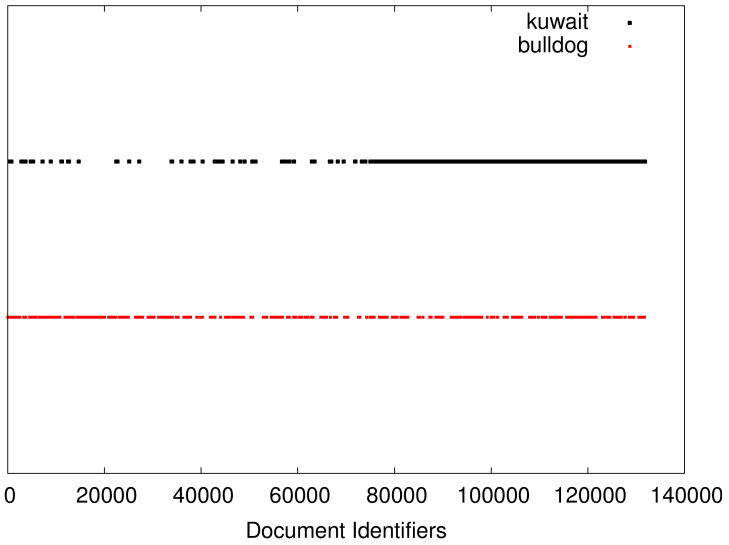


Figure 4.4: Occurrences for terms *kuwait* and *bulldog* in the LATimes collection

the document identifiers appearing in their postings. Their occurrences, in effect, come as grouped sequences, and are not randomly scattered through the whole document identifier range.

As a conclusion from this section, indexes from a text collection exhibit some kind of ordering, due to a temporary arrangement of documents, or crawling strategies. This has an impact in the distribution of terms, which forces most of them to deviate from a behaviour predicted random. Moreover, this *non-randomness* cannot be broken because it is tied to how terms occur in documents, which is unaffected by any document map (reordering). The main idea of the following sections, is to try to exploit this pattern, in such a way that it enhances the compressibility of the index. In other words, rearrange the d-gaps in posting lists so that compression methods reduce the number of bits required to encode them.

4.2 Document Assignment

The assumptions presented before can be summarised in one single concept: the identifiers inside postings reflect some kind of ordering. This ordering also reflects one fact: documents are clustered.

The explanation is that if a collection reflects a temporal order (like newspapers for instance) in its assignment of document identifiers, the number of terms co-occurring in consecutive or close-by

documents will be high. This fact has been the topic of some papers. In Bookstein et al. [1998], the authors characterise the *clumpiness* of terms for devising term content-detection techniques. Moffat and Stuiver [2000] exploited this contextual dependence of d-gaps in order to devise a context-sensitive compression methods. This effect is also referred to as the *clustering property* of a document collection (Silvestri et al. [2004]).

A definition of the reassignment problem follows. Ideally, one would like to find the optimum disposition of identifiers in postings, so that the d-gaps are minimal, and hence a compression scheme will squeeze more the data. This is clear bearing in mind some ideas presented before (section 2.4.2.2).

Consider a text collection $\mathcal{C}$ made up of N documents and T terms, an inverted file stores a set of T posting lists in following the format:

$$< t_i; f_{t_i}; d_{i1}, d_{i2}, \ldots, d_{if_{t_i}} >, d_{ik} < d_{ij} \forall k < j \ , \tag{4.10}$$

where f_{t_i} stands for the frequency of the term t_i (number of documents in which t_i appears), and d_{ik} is the k-th document identifier for the term i.

As stated in section 3.3, posting compression methods code the d-gaps $d_{ik+1} - d_{ik}$. Given so, an *assignment* of document identifiers in a document collection $\mathcal{C}$ is a bijective function π such as:

$$\begin{aligned} \pi : [1:N] &\rightarrow [1:N] \\ d_i &\rightarrow d_i^{'} \end{aligned} \tag{4.11}$$

Static codes exploit the fact that small document differences occur often, assigning shorter codes to them (see section 3.3). Indeed, it is clear that reordering the document identifiers in such a way that lower differences between document identifiers occur in all or most posting lists, the resulting total number of bits will also be reduced.

This assignment can reflect the fact that we aim to minimise the final representation of the index. In other words, which is the way this assignment produces a smaller index, given a compression method φ?

The problem can be re-stated as finding the minimum of the following function:

$$cost(\pi) = \sum_{t=1}^{T} \left(length(\varphi, \pi(d_{t,1})) + \sum_{i=1}^{f_t} length(\varphi, \pi(d_{t,i+1}) - \pi(d_{t,i})) \right) \tag{4.12}$$

where π is the reassignment bijective function that maps each document identifier in the range $\{1, N\}$ to a new one in the same interval, and $length(\varphi, x)$ gives the total number of bits needed for coding an integer x using the method φ. Most static codes represent an integer x with $O(log(x))$ bits, so

$$length(\varphi, x) \in O\left(\log(x)\right) \ , \tag{4.13}$$

is a close call.

4.3 Characterisation of the problem

We now characterise a simple case of the document identifier reassignment (Blanco and Barreiro [2005b]) in order to formalise it and:

- Characterise the NP-completeness of the problem
- Bring the graphs and index compression worlds together

The first point allows for the justification of heuristics that may be helpful to tackle this particular problem. The second one opens a wide new area for further exploration of techniques, that have proved to be particularly effective for graphs, to approach the assignment problem. Section 4.5 is an example of one of those techniques.

Consider a binary matrix B where the columns are document vectors and the rows represent the presence/absence of each term in every document. Working with this binary matrix, the problem of minimising the average d-gaps consists of finding the permutation of the matrix columns that minimises a function cost ϕ that computes the total d-gap length.

First, let us consider the posting list format in equation (4.10), where for each term t_i the ordered identifiers fall in the range $[d_{i1}, d_{if_{t_i}}]$.

The formalisation considered next considers the case of unary coding. This is important as most of the codes explained in section 3.3 are built upon unary coding, and it allows an easier representation for the cost function 4.12.

The cost function for unary coding, measures the average d-gap in every posting list, as $length(unary, x) = x$. It is possible to express the cost function as follows, if we avoid the first offset

$$\phi = \sum_{k=1}^{T} \frac{1}{f_{t_k} - 1} \sum_{i=2}^{f_{t_k}} d_{ki} - d_{k(i-1)} \tag{4.14}$$

$$= \sum_{k=1}^{T} \frac{1}{f_{t_k} - 1} \left[(d_{k2} - d_{k1}) + \cdots + (d_{kf_{t_k}} - d_{k(f_{t_k}-1)}) \right] \tag{4.15}$$

$$= \sum_{k=1}^{T} \frac{1}{f_{t_k} - 1} \left(d_{kf_{t_k}} - d_{k1} \right) \tag{4.16}$$

In this situation, given a permutation π of the matrix B, B_π, the cost function ϕ is the difference between the column number for the last and first **1** (document occurrence) in each row, divided by the total number of **1**s minus one (in the row):

$$\phi(\pi) = \sum_{k=1}^{T} \frac{1}{f_{t_k} - 1} \left(\mathbf{max}\{j|_{B_{\pi_{j,k}}>0}\} - \mathbf{min}\{j|_{B_{\pi_{j,k}}>0}\} \right) \tag{4.17}$$

$$= \sum_{k=1}^{T} \frac{1}{f_{t_k} - 1} \sum_{i=\mathbf{min}\{j|_{B_{\pi_{j,k}}>0}\}}^{\mathbf{max}\{j|_{B_{\pi_{j,k}}>0}\}-1} 1 \tag{4.18}$$

$$= \sum_{k=1}^{T} \frac{1}{\gamma_k} \left[\sum_{i=\mathbf{min}\{j|_{B_{\pi_{j,k}}>0}\}}^{\mathbf{max}\{j|_{B_{\pi_{j,k}}>0}\}} \alpha_{\pi_i} \right] - \sum_{k=1}^{T} \frac{1}{\gamma_k} , \tag{4.19}$$

if γ_k stands for the inverse term frequencies minus one, and $\alpha_{\pi_i} = 1, \forall i : 1 \leq i \leq N$.

In the last form of $\phi(\pi)$ in 4.19, the first term is the expression of the function cost in the Actor Costs (PSP-AC) of the shooting schedules problem. PSP-AC is a generalisation of the Average Order Spread (PSP-AOS) problem (Fink and Voss [1998]), and both are pattern sequencing problems.

General pattern sequencing problems have the goal to find a permutation of predetermined production patterns, useful for scheduling or planning.

Note that it is also possible to consider the document identifier reassignment problem as a PSP-AOS in the particular case that we measure the global average d-gap instead of the per posting list average d-gap. Actually, coding a posting list implies taking into account the first document identifier appearing in the sequence. It is possible to include this offset into the equation by simply adding it into the sum. Therefore, considering the first offset in each posting list, the cost function is

$$\phi_2 = \sum_{k=1}^{T} \frac{1}{f_{t_k}} d_{kf_{t_k}} \tag{4.20}$$

The PSP-AOS is closely related to a particular case for graph-layout problems, surveyed by Díaz et al. [2002]. They are a class of combinatorial optimisation problems where the goal is to find

an specific labelling of the nodes in the graph in such a way that some objective cost function is optimised. This connection is appealing, because even if most graph layout problems are NP-complete, there exists several heuristics provide feasible solutions with an almost optimal cost.

The problem is thus connected with the *optimal arrangement problem* which is stated as follows. Given a graph $G = (V, E)$ and a positive integer k, find a bijective function $g : V \rightarrow \{1, 2, ...|V|\}$ such as

$$\psi(g) = \sum_{u,v \in E} |g(u) - g(v)| \leq k \tag{4.21}$$

Recall that there is an almost immediate connection between this graph and the B matrix. Let $V = \{1, ..., T\}$ and $E = \{e_1, e_2, ..., e_N\}$ be the set of vertexes and edges of the graph (the columns are the terms and the rows are the documents). The bitmap B is constructed as $B_{ij} = 1$ iff $(e_i, e_j) \in E$. Consider a *lighter* version of the identifiers assignment problem, where $\gamma_k = 1$:

$$\phi'(\pi) = \frac{1}{T} \sum_{j=1}^{T} \left[\sum_{i=\mathbf{min}\{j|_{B_{\pi_{j,k}}>0}\}}^{\mathbf{max}\{j|_{B_{\pi_{j,k}}>0}\}} \alpha_{\pi_i} \right] - T \tag{4.22}$$

This in indeed a transformation as the function g exists if and only if there is a permutation B_π with $\phi'(\pi) \leq \frac{k}{T}$:

$$\psi(g) = \sum_{(e_x,e_y) \in E} |g(x) - g(y)| \tag{4.23}$$

$$= \sum_{j=1}^{T} |\mathbf{max}\{j|_{B_{\pi_{j,k}}>0}\} - \mathbf{min}\{j|_{B_{\pi_{j,k}}>0}\}| \tag{4.24}$$

$$= \sum_{j=1}^{T} \left[\sum_{i=\mathbf{min}\{j|_{B_{\pi_{j,k}}>0}\}}^{\mathbf{max}\{j|_{B_{\pi_{j,k}}>0}\}} \alpha_{\pi_i} - 1 \right] \tag{4.25}$$

$$= T \cdot \phi'(\pi) \tag{4.26}$$

The solution to the problem formalised here is also a solution to the reassignment of document identifiers using unary codes for d-gap encoding. However, in the real case the minimisation must take into account the coding scheme. In order to obtain good compression ratios, a better goal would be to minimise the d-gap products. This way, the log of the products and the sum of the logs would be minimal, with practical consequences in the final coded posting lists.

In this section, we formalised a particular a case of the document identifier assignment problem, defining the real cost function and identifying the problem as a PSP. This formalisation shows the NP-Completeness of the problem. This fuels the development of suitable heuristics to tackle the problem, which are presented next.

4.4 Prior work on document identifier reassignment

The problem presented so far has been named in the IR literature indistinctly as either *assignment* and *reassignment*, depending on the case. The former suggests that the identifiers can be associated to documents on indexing time, or before the actual index is built. The latter suggests that the identifiers come with a certain order, as stated in section 4.1. In the following, we will refer indistinctly to both nomenclatures.

The reassignment of document identifiers is a technique developed recently to enhance static compression in inverted files. As the intuitions presented so far suggest, some work demonstrated that it is possible to lower the number of bits required to code each posting list by reassigning the document identifiers of the collection. The reassignment of document identifiers has been addressed

by three different strategies. A first family of algorithms split the collection with graph-partitioning or clustering algorithms and obtain a new order which exploits locality (Blandford and Blelloch [2002], Silvestri et al. [2004]). A second kind, considers the task as a graph-layout problem (Shieh et al. [2003], Blanco and Barreiro [2005a]). Graph-layout problems try to find the optimum labelling for the vertices in a graph that minimise a certain cost function (dependent on the particular vertex labelling). To apply heuristics designed to solve these kind of problems to the reassignment problems it is necessary to transform the view of the inverted file into a graph or adjacency matrix. A more detailed connection between these two worlds has been presented in section 4.3. Finally, Silvestri [2007] demonstrated that sorting the document identifiers by their *urls* yielded excellent compression improvements. The technique is extremely appealing, given that the algorithm is linear, does not require to inspect the documents (or a pre-built index, for the case) and scales up to collections of any size. This fact was also noticed by Boldi and Vigna [2005b] to improve the compressibility of the Web-graph.

In section 4.6.2 we extend upon the work presented in Blanco and Barreiro [2005a], where the heuristic that considers the problem as a *Travelling Salesman Problem* (TSP) introduced by Shieh et al. [2003] allowed an efficient solution of the reassignment after reducing the input data dimensionality using a *Singular Value Decomposition* (SVD) transformation. The TSP-based strategy considers the reassignment of document identifiers as a graph-traversal problem and finds a path according to a global minimum. Although the strategy is not explicitly based on locality, it has obtained very appealing results in practice. As it is shown later in section 4.5, this path is the one that maximises the bitwise intersection of the document vectors. In section 4.9 we further give an explanation for the good behaviour of the TSP heuristic for this problem, based on the formal characterisation presented before in section 4.3.

The main reason why a SVD reduction is useful in this context, is that the TSP technique has to compute several similarity values between documents in order to choose the edges that will connect the nodes in the final path. Thus, SVD computation is helpful given the high-dimensionality space of terms by documents, while also being the best choice for matrix transformation, as it obtains the best similarity measure in a reduced-dimensionality space.

In the following, we extend the work presented in Blanco and Barreiro [2005a] by reviewing and reimplementing two clustering algorithms previously adapted to the reassignment problem (Silvestri et al. [2004]), in order to evaluate the two different approaches within the same conditions. In the proposal presented by Silvestri et al. [2004] the assignment of document identifiers is done *on the fly*, i.e. without the existence of a prior built inverted file. In fact, this might be the main reason why the baseline for the evaluation of the approach is a random order and not the original collection ordering. In this work, we measure the improvements after the reassignment of document identifiers with respect to the random order and the identity (original) collection. Lastly, we present a new technique that combines the two strategies in such a way that the best tradeoff between compression and efficiency is achieved. The main bulk of the following section is published in Blanco and Barreiro [2006].

4.5 The TSP approach to the document identifiers reassignment problem

Given that this work will illustrate the use of the dimensionality reduction technique for document reassignment with the TSP-based solutions, we briefly review the work by Shieh et al. [2003].

4.5.1 The document identifiers reassignment problem as a TSP

The formulation presented so far considered an inverted file first as a posting list set and then a binary matrix. Each posting list coming from the IF contains the information for a single term appearing in the document collection, expressed as a sequence of encoded d-gaps $G_t = \{g_1, \ldots, g_{f_t}\}$. Drawn from equation 4.19, it follows that the document reassignment problem is to find the bijective function f that

Greedy-NN algorithm

Input:
The Graph G
The Vertex set V
The weighted Edges set E
Output:
A global path P maximising the similarity between vertexes
Select the edge $e(v_i, v_j) \in E$ with the largest weight;
Add v_i and v_j to P;
$v_{last} \leftarrow v_j$;
while ($|P| \neq |V|$) **do**
Choose $v_k \in V$ and $v_k \notin P$ such that $e(v_{last}, v_k)$ is maximal;
Add v_k to P;
$v_{last} \leftarrow v_k$;
end while
return P;

Figure 4.5: Greedy-NN algorithm: it computes an approximation to the TSP over a given graph using a greedy heuristic

- maps each document identifier into a new identifier in the range $[1 : N]$
- minimises the cost in bits of coding the posting lists

Similarity between documents is defined as the number of common terms, and maintained in a similarity matrix Sim, where Sim_{ij} represents the similarity between the document i and the document j.

Shieh et al. [2003] proposed a gap-reduction strategy based in the transformation of the problem into a *Travelling Salesman Problem* (TSP), stated as follows:
given a weighted graph $G = (V, E)$ where $e(v_i, v_j)$ is the weight of the edge from v_i to v_j, find a minimal path $P = \{v_1, v_2, \ldots, v_n\}$ containing all the vertexes in V, such as if $P' = \{v'_1, v'_2, \ldots, v'_n\}$ is another path in G, $\sum_{i=2}^{n} e(v_i, v_{i-1}) \leq \sum_{i=2}^{n} e(v'_i, v'_{i-1})$.

This brings us back to the view of the IF as a graph, just like in the *optimal rearrangement* case. For the particular case we are interested in, the graph devised from building a huge adjacency matrix containing the document-to-document similarities. In fact, considering Sim a weighted adjacency matrix, it is possible to build a Document Similarity Graph (DSG) expressing the aforementioned similarities. This graph can be traversed by a gap-reduction strategy based on the similarity factor between documents. The idea is to assign close document identifiers to similar documents, as this will likely reduce the d-gaps in common terms postings. This traversing problem can be transformed into TSP just by considering the complement of the similarity as the weight in TSP. The solution of TSP is the path that minimises the sum of the distances between documents, therefore the algorithm is an appropriate strategy to the document reassignment problem.

4.5.2 Heuristic approximations

TSP is an *NP*-complete problem, so some polynomial-time heuristic approximations were modified for the reassignment problem. These algorithms were classified as greedy algorithms and spanning tree algorithms. The heuristic considered in this section is the Greedy-NN algorithm, which is described next.

When describing the heuristic approximation, TSP is considered as the one that obtains the path that maximises the similarity sum between consecutive documents. The Greedy-NN (Nearest Neighbour) expands the path by adding the closest vertex to the tail of the current path. In each

Table 4.3: Statistics of the pure TSP approach on two TREC collections as reported by Shieh et al. [2003]

Collection	FBIS	LATimes
Size of the Collection	470 MB	475 MB
Number of distinct terms	209,782	167,805
Number of distinct documents	130,471	131,896
Temporal Cost	19h37'	23h16'
Space Cost	2.10 GB	2.17 GB

iteration the algorithm adds a new vertex (document), by choosing the vertex that is most similar to the last vertex in the path. This approximation is high time consuming. Each vertex is inserted only once in the path P and at iteration i the algorithm does $N - i$ comparisons (the remaining documents) involving the term size t of both documents. Therefore the overall complexity is $O(N^2t)$.

Figure 4.5 shows the role of the Greedy-NN algorithm in the reassignment process. The algorithm takes the input collection and then computes a ordering of the documents, that is used for recompressing the inverted file via doc. id. reassignment.

4.5.3 Implementation considerations

The TSP approximation for the identifier reassignment problem was evaluated by Shieh et al. [2003]. The solution demonstrated significant improvements in the compression ratio, although it presented some design challenges and an extremely poor performance time and space results.

First, this approach requires a big amount of space. The similarity matrix is symmetric ($Sim_{ij} = Sim_{ji}$) and the elements in the diagonal are not relevant, so it is easy to prove that we need to store $\frac{N(N-1)}{2}$ *similarity pointers* ($O(N^2)$). Even with a suitable coding scheme this amount can become unmanageable, so a matrix partitioning technique has to be developed. Second, building this matrix can be very expensive if it does not fit into memory, as each update has to access the disk twice, involving big delays.

Experimental results were presented for two medium sized collections (FBIS and LATimes in TREC disk 5), to prove the effectiveness of this mechanism. These tests are summarised in table 4.3.

It is important to remark that the work by Shieh et al. [2003] provides bar graphs that show an approximated gain of one bit per gap when reassigning with the Greedy-NN for δ and γ coding. The temporal costs include the process of building the similarity matrix, greeding and re-compressing the inverted file. However, the results show that this full TSP approach may be unacceptable for very large collections, as it takes 23 hours and 2.17 GB to process a 475 MB collection.

4.6 Document identifiers reassignment by dimensionality reduction and the TSP strategy

The TSP strategy achieves very good compression ratios. For this reason, we present a new approach based on dimensionality reduction in which the TSP algorithm can operate efficiently. This technique is based on the application of a SVD transformation to the input data, in order to obtain a new representation of the data structures that allows the reordering of the document identifiers in a dimensionality-reduced space. Once the new ordering is obtained, the posting lists in the inverted file are re-compressed, using the document identifiers given by the assignment function. It is worth noting the different nature of this SVD application in IR with respect to other typical SVD IR usages, like the LSI retrieval model.

In-memory graph approximations involve very high computational costs (see section4.5.3). The assignment technique gives good time-performance in Web collections with good compression ratios.

However the amount of memory employed by the Greedy-NN algorithm is overwhelming. In fact, very large collections should be split and partitions reordered separately. The overall behaviour can become worse when the average terms per document ratio increases.

4.6.1 Singular Value Decomposition

Singular Value Decomposition (SVD) is a well known mathematical technique used in a wide variety of fields. It is used to decompose an arbitrary rectangular matrix into three matrices containing singular vectors and singular values. These matrices show a breakdown of the original relationships into linearly independent factors. The SVD technique is used as the mathematical base of the Latent Semantic Indexing (LSI) IR model (Deerwester et al. [1990]).

Analytically, we start with X, a $T \times D$ matrix of terms and documents. Then, after applying SVD to X, the matrix is decomposed into three sub-matrices:

$$X = F_0 S_0 D_0' \tag{4.27}$$

F_0 and D_0 have orthonormal columns, and S_0 is diagonal and, by convention, $s_{ii} \geq 0$ and $s_{ii} \geq s_{jj} \forall i \geq j$. F_0 is a $T \times m$ matrix, S_0 is $m \times m$ and D_0' is $m \times N$ where m is the rank of X. However it is possible to obtain a k-ranked approximation of the X original matrix by keeping the k largest values in S_0 and setting the remaining ones to zero, thus obtaining the matrix S with $k \times k$ dimensions. As S is a diagonal matrix with k non-zero values, the corresponding columns of T_0 and D_0' can be deleted to obtain F, sized $T \times k$, and D', sized $k \times N$, respectively.

This way we can obtain $\hat{X}$, which is a reduced rank k approximation of X:

$$X \approx \hat{X} = FSD' \tag{4.28}$$

$\hat{X}$ is the closest rank k approximation of X in terms of the Euclidean or Frobenious norms, i.e. the matrix which minimises $||X - \hat{X}||_N^2$ where $|| \cdot ||_N^2$ is the involved norm.

The i-th row of DS gives the representation of the document i in the reduced k-space and the similarity matrix $\Theta(X)$ is k-approximated by $\Theta(\hat{X})$:

$$\Theta(X) \approx \Theta(\hat{X}) = \hat{X}'\hat{X} = DS^2D', \tag{4.29}$$

where $\hat{X}'$ is the transposed matrix of $\hat{X}$ and D' is the transposed of D.

If $D_{N \times k} = \{z_{ij}\}$ and $\{s_i\}$ is the set of diagonal elements of S, it is easy to prove that

$$\Theta(\hat{X})_{ij} = \sum_{\gamma=0}^{k-1} z_{i\gamma} z_{j\gamma} s_\gamma^2 \tag{4.30}$$

Therefore it is possible to calculate $\Theta(\hat{X})_{ij}$ solely by storing the k elements of the $\{s_i\}$ set and the $N \times k$ matrix D, instead of computing and writing the full rank matrix $\Theta(X)_{N \times N}$.

The output of the SVD of X, $\hat{X}$ has been used in the computation of $\Theta(\hat{X}) = \hat{X}' \cdot \hat{X}$. The same result could be obtained by calculating the SVD of $\Theta(X) = X' \cdot X$ due to the uniqueness property of SVD (Bartell et al. [1992]). Since SVD computes the best rank k approximation, it is proved that the best rank k approximation of $\Theta(X)$ is obtained starting from X and without the need of computing $\Theta(X)$.

4.6.2 SVD in the document reassignment problem

Figure 4.6 describes the system built for testing this approach. The inverted file builder mechanism outputs the X data matrix to a SVD module. This module produces the matrices $D_{N \times k}$ and $S_{k \times k}$ that allow the computation of $\Theta(\hat{X})$, therefore it is no longer needed to store the similarity matrix $\Theta(X)_{N \times N}$. The reassignment module uses the SVD output matrix to compute the TSP approach described in section 4.5.2. As k is a constant factor, we can conclude that the space usage of the algorithm is $O(N)$ now, i.e. linear in collection size and not dependent on document size. The output

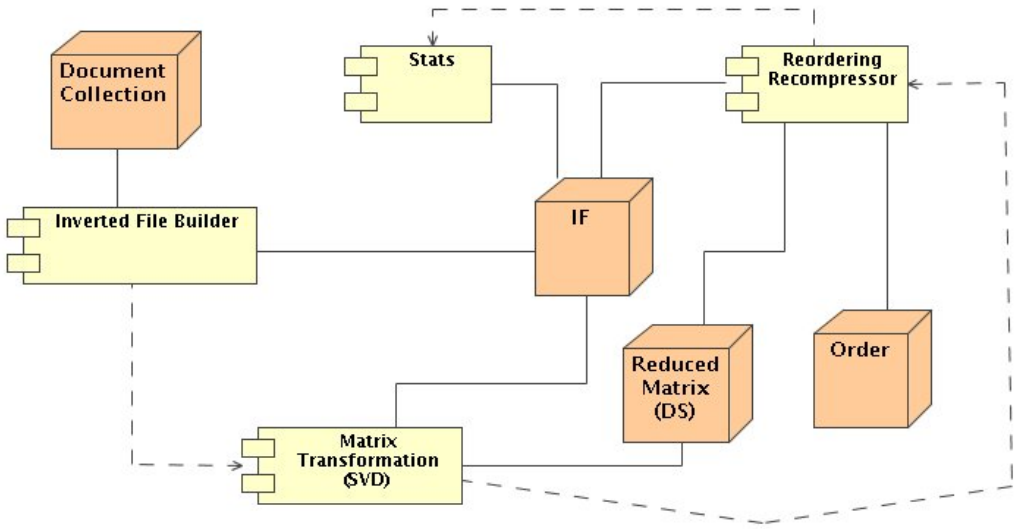


Figure 4.6: Block diagram showing the interaction between the indexing and reassignment systems

SVD-based reassignment of document identifiers

1: **Input:**
   The Document Collection C
   The k SVD parameter
2: **Output:**
   A reordered Inverted File
3: $originalIndex \leftarrow$ **createIndex(C);**
4: $reducedMatrix \leftarrow$ **SVD(**$originalIndex, k$**);**
5: $documentOrder \leftarrow$ **SVDGreedyNN(**$reducedMatrix$**);**
6: $newIndex \leftarrow$ **re-compress(C,** $documentOrder$**);**
7: **return(**$newIndex$**);**

Figure 4.7: General process for re-compressing an index after performing an SVD reduction and TSP reordering

of the TSP reassignment module is used by an inverted file recoding program which exploits the new locality of the documents to enhance the d-gaps compression. Finally, some statistical information is taken into account to make suitable comparisons between compression ratios achieved by the original encoding and those obtained after reassignment.

The summarised steps are detailed in figure 4.7, where SVDGreedyNN is a modified version of the Greedy-NN algorithm that uses the $\Theta(\hat{X})$ matrix for computing the similarity measure between vertexes v_i and v_j.

The main difference in this model is that computing the similarity between two documents d_i and d_j involves k operations ($\sum_{\gamma=0}^{k-1}(DS)_{i\gamma}(DS)_{j\gamma}$) and storing k real pointers per document, making a total of $k \times N$ for the full matrix. This representation can fit smoothly into memory by adjusting the parameter k and uses considerably less space than the original $N \times N$ matrix. Furthermore, the space usage can be precalculated so suitable scalable algorithms can be easily developed. Considering 32 bits per float (real number), the implementation proposed uses $4 \times k \times d$ bytes of main memory.

One point to consider is the heuristic for choosing the starting node on the Greedy-NN algorithm which was also employed to solve the TSP. The algorithm (section 4.5.2) first chooses the edge (v_i, v_j) that has the maximum value. This involves the computation of the similarity for every document pair

Table 4.4: TSP + SVD algorithm bits per gap and execution times, LATimes collection

			k						
	Random	Original	200	100	50	20	10	5	1
Gamma	8.15	7.77	6.71	6.75	6.79	6.89	6.99	7.13	7.44
Delta	7.65	7.25	6.29	6.36	6.39	6.48	6.57	6.73	7.02
Interpolative	6.08	5.88	5.25	5.26	5.28	5.29	5.33	5.44	5.57
Golomb	5.82	5.81	5.79	5.79	5.79	5.79	5.79	5.79	5.79
SVD	–	–	42'31"	20'37"	11'25"	4'05"	2'33"	1'55"	1'09"
Reorder	–	–	8h33'	4h24'	2h15'	58'18"	32'05"	18'31"	7'30"

(d_i, d_j). In the approach presented here, selecting the first node this way involves more operations than the rest of the algorithm itself. Hence, we propose a less time-expensive heuristic, consisting in calculating, after dimensionality reduction, each (d_i, d_i) self-similarity and choosing the document (node) with the largest value.

4.6.3 Experiments and results

Several experiments for testing the low-dimension approach on the two TREC document collections are described in table 4.3. These collections were not preprocessed, so indexing and reordering did include stop words and terms were not stemmed. The machine used was a 2.5 GHz AMD with 40 GB ATA disk and 1GB of main memory, running Linux OS. The original index file was built with MG4J[1] Managing Gigabytes for Java (accessed on 03-04-08) from the University of Milan, a free Java implementation of the indexing and compression techniques described in Witten et al. [1999] and originally implemented in the MG software[2]. The SVD module used was SVDLIBC[3], a C library based on the SVDPACKC library. The reassignment, recoding and statistical software was written in Java. It should be pointed out that the MG4J software had to be modified in order to output data directly to the SVDLIBC module. Also some modifications were made that allowed to encode document pointers with interpolative coding.

The first experiment assessed the performance of the system with the Greedy-NN algorithm, in terms of average bits per compressed document pointer (d-gap). The document collections were inverted, the IF was inputted to the SVD module and the program computed the Greedy-NN in the reduced dimension for the reassignment task. After reordering the collection, the inverted file was re-compressed. The software measured the average bits per gap in the inverted file, before and after reordering and re-compressing, which reflects the amount of compression gained by reordering the document collection. We ran several tests varying the following parameters:

- the parameter k which reflects the desired dimensionality reduction
- coding schemes for document pointers: δ coding, γ coding or interpolative coding

The best results are obtained considering X as a binary matrix in the reassignment process. The elements of X represent the presence or absence of a term in a given document. The re-compressing module acts over the original index file which contains within-document term frequency and frequency of the term in the collection. Results are given in bits per document gap because it is a measure independent of these indexing options. As stated in section 4.6, the memory usage leads to $4 \times N \times k$ bytes, concretely $0.497707 \times k$ MB for the FBIS and $0.503143 \times k$ MB for the LATimes (for $k = 200$ less than 101 MB in both collections).

In order to assess the performance of the reassignment algorithm, the experiments included results using several algorithms employed for posting list compression: two global non-parametrised methods (gamma γ and delta δ Elias [1975]), one global parametrised (Golomb code Golomb

[1] `http://mg4j.dsi.unimi.it/`(accessed on 03-04-08)
[2] `http://www.cs.mu.oz.au/mg/` Managing Gigabytes (accessed on 03-04-08)
[3] `http://tedlab.mit.edu/~dr/SVDLIBC/` (accessed on 03-04-08)

Table 4.5: TSP + SVD algorithm bits per gap and execution times, FBIS collection

			k						
	Random	Original	200	100	50	20	10	5	1
Gamma	7.84	6.74	6.20	6.24	6.31	6.46	6.63	6.81	7.07
Delta	7.35	6.35	5.80	5.86	5.92	6.07	6.23	6.42	6.69
Interpolative	5.83	5.25	4.98	4.99	5.01	5.06	5.17	5.21	5.33
Golomb	5.61	5.60	5.59	5.59	5.59	5.59	5.59	5.59	5.59
SVD	–	–	34'58"	15'01"	5'42"	3'18"	2'09"	1'33"	58"
Reorder	–	–	8h30'	4h20'	1h48'	58'20"	31'13"	17'55"	7'05"

[1966]), and one local context-sensitive (interpolative coding Moffat and Stuiver [2000]). All these coding methods have been described in section 3.3.

Tables 4.4 and 4.5 show the results for the different coding schemes. In the rows labelled with each of the coding methods, columns refer to bits per document gap results for: random reassignment, original document identifiers and reassignment after reducing the dimensionality with different k values. Actually, the default starting point for any rearrangement is the pre-existing order, which already includes an element of clustering because of the chronology of the documents. However, in previous studies (Blandford and Blelloch [2002], Silvestri et al. [2004]), the authors used the random reassignment as a baseline, and for comparison issues this column is included here as well. By assigning values to k similar to those used in retrieval (Dumais [1994]), the low-dimension algorithm operates with gains that produce improvements in bits per gap. As expected, the method behaves better as the k value increases. Also, the figures seem to have an asymptotic behaviour. With k=200, for the LATimes collection (FBIS collection) the method achieved a 13.65% (8.02%) gain in compression ratio with respect to the original document identifier order with γ encoding, 13.24%(8.66%) for the δ coding, and 11.32% (5.15%) for the interpolative coding. These values are 17.67% (21.92%), 17.80% (21.10%) and 13.66% (14.58%) respectively for both collections and the three encoding schemes, with respect to a random reassignment. The gains in the FBIS collection are worse than the ones in the LATimes, although starting from a randomised order the result is inverted. This is the expected behaviour if the FBIS collection exhibits a better original document order. One point to remark upon is that even in the case of interpolative coding, where the starting point is much better, the method is able to produce gains in bit per document gap. Regarding the compression ratios with Golomb coding, it performed the best for the random assignment, and it is clear that it does not benefit from the locality of the documents. The reason for this behaviour is that the Golomb code is not affected by the skewness of the document distribution, as it renders a geometric distribution, as discussed in the very beginning of this chapter. However, interpolative coding with reordering is the option that achieves best compression results.

The tests shown here did not include the computation of the full dimension solution as presented by Shieh et al. [2003], because it requires the development of matrix partition techniques and partial reading/writing, which are the tasks we want to avoid. The authors only provided bar graph results for γ and δ coding in the LATimes and FBIS collections. However, exact compression values depend on the indexing software and particular indexing options. This information is not explicitly provided, thus it is not possible to make exact comparisons between their published full-dimension results and the k-dimension solutions.

The last two rows of tables 4.4 and 4.5 state algorithm running times. Time measurement is divided in three parts: inverted file construction, SVD running time and reordering and re-compressing time. Times are given for δ coding only, as the tables illustrate the time variation within the k parameter. The time variance due to k, is only dependent on SVD and reordering times, so times with δ coding only are suitable for this purposes (re-compressing time is about 16 seconds for both collections). As the system was built upon different modules, the different software pieces employ a lot of temporal I/O transfer time, which also is measured, so results are given in *elapsed time*. Inversion takes 5' 20" for the FBIS collection and 6' 03" for the LATimes collection and it is not shown in the tables. Although the SVD software performs well for the collections and k values

Table 4.6: c-GreedyNN algorithm performance: bits per gap and running time (k=200 and δ coding), LATimes collection

	c								
	70	100	150	200	300	400	500	1000	2000
Bits per gap	6.68	6.72	6.77	6.81	6.87	6.92	6.95	7.02	7.09
c-GreedyNN & re-compress	18'08"	9'50"	5'08"	3'21"	1'57"	1'25"	1'07"	42"	47"

used, the TSP greedy algorithm running time still rises to high values. Hence, we conclude that it is possible to achieve good compression ratios with reasonable time performance with the technique just introduced. Next, we will show results for one compression method, as the experiments are aimed at clarifying if the different reordering variants give better bits per gap values, along with their respective temporal cost, without focusing on coding methods.

4.7 The c-blocks algorithm

In order to further exploit the advantages of the TSP strategy and the dimension reduction, we present next a simple new algorithm based on the division of the original problem in c subproblems, hereinafter c-GreedyNN. Later on, the performance between this algorithm and others based on collection partitioning will be compared.

The c-GreedyNN algorithm operates as follows: first, it divides the DS matrix (which represents the document similarities in the k space) in c blocks of $[N/c]$ documents each. Then, each block is reordered by running the greedy algorithm. Finally, a block order is decided by running another greedy with c documents, each one selected from different blocks. For a simpler explanation, consider N an exact multiple of c. Analytically, the Greedy-NN after dimensionality reduction does N comparisons to select the first document, and $\frac{N(N-1)}{2}$ for reordering, resulting in

$$\frac{N}{2} \cdot (N+1) \in O(N^2) \tag{4.31}$$

comparisons involving k multiplications each. The new approach chooses c block-representatives and then performs c greedy runs with d/c documents, resulting in

$$N + c \cdot \left(\frac{\frac{N}{c}(\frac{N}{c} - 1)}{2} \right) = \frac{N}{2} \left(\frac{N}{c} + 1 \right) \tag{4.32}$$

comparisons, so the overall number of operations is reduced in a $1/c$ factor. Experimental results with different values of the number of blocks c are presented in tables 4.6 and 4.7.

Results are provided for the LATimes and FBIS collections, $k = 200$ and δ coding. Tables 4.6 and 4.7 also show that the compression factor increases as the number of blocks decreases, with a goal value of 6.29 (5.80) for the LATimes (FBIS) collection, which is the value of considering the matrix as one single block.

The algorithm is efficient in running time, as expected from the analytical form, and it gives acceptable compression values. We ran all the experiments with $k = 200$, as it proved to drive the best performance results as well as the slowest reordering times, so lowering those times while keeping good compression ratios is a challenge for the c-blocks algorithm.

The method enhances the original compression ratio 7.25 (6.35) for the LATimes (FBIS) collections and δ coding. These improvements are higher when comparing to the randomly ordered collection compression ratio 7.65 (7.35). The tradeoff between the final compression achieved and the time usage, can be parametrised by selecting the c and k values.

Table 4.7: c-GreedyNN algorithm performance: bits per gap and running time (k=200 and δ coding), FBIS collection

	c								
	70	100	150	200	300	400	500	1000	2000
Bits per gap	5.98	6.00	6.02	6.05	6.09	6.12	6.14	6.22	6.30
c-GreedyNN & re-compress	17'37"	9'35"	4'59"	3'15"	1'53"	1'21"	1'05"	40"	45"

4.8 Clustering strategies

4.8.1 Clustering solutions to the document identifiers reassignment problem

Other work addressed the document identifiers reassignment problem with clustering techniques. The main idea is to enhance the locality in the collection by assigning similar documents to the same cluster, hoping that the d-gaps are lowered if documents on the same cluster are assigned close identifiers. The fact that this property comes with the original order of collections was presented in section 4.1, whereas the bulk of this work (all previous sections) has proved that *clustering* does not exactly imply *optimality*. Although the solution to the original problem is closer to graph-based techniques, these are especially hard to compute in IR environments, due to the huge amount of data at hand. Hence, enhancing the original clustering ordering of the collection is an appealing way for bringing up a collection order that makes standard lossless compression to behave better.

For the particular case of clustering, two families of algorithms were developed to compute an efficient document assignment: the *top-down assignment* and the *bottom-up assignment*. The top-down assignment schemes start with the whole collection and recursively split it into sub-collections, inserting similar documents into the same sub-collections. After this phase, the algorithm merges the sub-collections obtaining a single and ordered group of documents, which is used to compute the assigning order. Bottom-up schemes start from a set of documents, extracting disjoint sequences containing similar documents. Each sequence is ordered, and the final assignment is computed by considering an arbitrary order between sequences.

The first top-down algorithm was introduced by Blandford and Blelloch [2002], who developed a technique that improved the compression ratio in about 14% in TREC text collections. The technique employs a similarity graph as described in section 4.5.1, and operates in three different phases. The first phase constructs the document-document similarity graph from the original inverted file. The second part of the algorithm calls to a graph partitioning package which implements the Metis (Karypis and Kumar [1995]) algorithm for splitting recursively the similarity graph produced by the first part. Finally, the algorithm applies rotations to the clustered graph outputted by the second part in order to optimise the obtained order. The final assignment of the document identifiers is obtained by simply depth-first traversing the resulting clustering tree. As is stated by Blandford and Blelloch [2002], constructing a full similarity graph is $O(n^2)$, so the raw technique may not be suitable for very large collections. Nevertheless, the efficiency of the algorithm can be controlled by two parameters: τ and ρ. The first parameter, τ, acts as a threshold for discarding high-frequency terms, i.e., if a term t_i has a number of document occurrences $|t_i| > \tau$ it is discarded for the construction of the similarity graph. Actually the algorithm works with a sample of the full similarity graph. The parameter ρ stands for how aggressively the algorithm sub-samples the data: if the index size is n it extracts one element out of $\lfloor n^\rho \rfloor$. By tuning τ and ρ the technique may lead to a tradeoff between efficiency and time and memory usage.

The work by Silvestri et al. [2004] follows a different approach by *assigning* the document identifiers *on the fly* during the inversion of the text collection. This is done by calculating a *transactional* representation form of the documents, which stores for each document d_i a set of 4-byte integers representing the MD5 Message-Digest (Rivest [1992]) of each term appearing in d_i. Using this representation, a new top-down algorithm called *bisecting* is presented and evaluated, as well as two bottom-up algorithms: the *k-means* and *k-scan*. These techniques were tested in the Google Programming Contest collection. Both algorithms proved to be competitive with respect to a random baseline, with k-scan performing remarkably better and with reasonable computing times.

As the whole collection is ordered while indexing, the baseline chosen for measuring the performance of these algorithms was a randomised collection. The previous results with the FBIS and LATimes presented in section 4.6.3, showed that randomly ordering the collection always yields worse compression ratios than those obtained with the *natural order*, i. e. leaving the collection with its original numbering of documents. This motivated us to re-implement these algorithms, but in a scenario of reassignment of document identifiers, with the existence of an inverted file, rather than performing the assignment on the fly.

In the particular implementation of the clustering algorithms presented here, the construction of the inverted file makes unnecessary the transactional hashing of the terms. In order to improve the computational times of the document-to-document similarities, we worked not only with the inverted file, but also with the direct file. This structure can be seen as an inverted file where the index keys are the documents, and the posting lists contain term identifiers. It is worth noting that this direct file is also compressed with the techniques used to compress the inverted file.

4.8.2 Bisecting and k-scan

In this section, we adapt the bisecting and k-scan algorithms to the reassignment scenario. The bisecting algorithm (figure 4.8) belongs to the top-down family, and operates as follows: first, it randomly selects two documents as the centres of mass of a two-partition of the collection. Next, it assigns every document in the input data to one of the two partitions, according to the similarity between the document and the centre of mass. The technique keeps these two partitions equally sized, so many documents at the end of the input may fall into an undesired cluster. This partitioning scheme is called recursively until the input partition only contains one element. The merging phase is done by comparing the borders of the two partitions. If the similarity between the document at the right border of cluster D' and document at the left border of cluster D'' is less or equal than the similarity between the document at the right border of cluster D'' and document at the left border of cluster D', in the final order cluster D'' must precede D'. Otherwise D' must precede D''. In the pseudo-code, $\oplus$ represents the concatenation of two sets containing document identifiers.

The k-scan algorithm (figure 4.9) is a simplified version of the popular k-means clustering algorithm. The algorithm sorts the elements of the collection by descending document length, and chooses the largest unselected document as the cluster representative. Then it assigns the $|D|/(k-1)$ unselected documents which are most similar to the cluster representative. There is an internal order in each cluster, given by this similarity measure. The selection of the cluster representative and the assignment of the documents to that cluster is repeated k times.

The work in Silvestri et al. [2004] shows that the space storage and the computational cost of the bisecting algorithm are both superlineal $O(N \cdot \log(N))$ in the asymptotic analysis. For k-scan, the space occupied is lineal $O(|S| \cdot N)$ and the complexity is $O(N \cdot k)$, where $|S|$ is the average document length.

4.8.3 Experiments and results

The experiments were designed to assess the performance of the clustering techniques and compare them to the TSP-based approaches, measuring the improvements in compression ratios with respect to both the original and randomised collections. The similarities between documents were measured with the Jaccard distance, repeating the conditions described in Silvestri et al. [2004]. In order to exploit the SVD transformation previously computed, the experiments were repeated measuring the similarities between documents with the inner product in the reduced dimensionality space. In these cases, the k parameter in the SVD transformation was fixed to 200, as this value proved to be worthy (sections 4.6.3 and 4.7). Hereinafter, the k parameter appearing in the tables refers to the number of scans of the document set D in the k-scan algorithm. In this set of experiments the compression of the posting lists was done with δ coding.

The bisecting algorithm achieves different gains in bits per gap in each run, due to the random centre selection for the different executions, although they are stable around a certain value. In 10 runs of the algorithm, for the LATimes collection the values obtained fall in the range $[6.82, 6.87]$

Algorithm Bisecting(D)

Input:
The document set D
Output:
A list with the final order
if $|D| > 1$ **then**
  $c_1 \leftarrow$ randomDocument(D);
  $c_2 \leftarrow$ randomDocument($D \setminus c_1$);
  $D' \leftarrow \{c_1\}$;
  $D'' \leftarrow \{c_2\}$;
  for all $d \in D \setminus \{c_1, c_2\}$
    if $(|D'| \geq \frac{|D|}{2}) \vee (|D''| \geq \frac{|D|}{2})$ **then**
      Assign d to the smallest partition;
    else
      if similarity$(c_1, d) \geq$similarity(c_2, d) **then**
        $D' \leftarrow D' \cup d$;
      else
        $D'' \leftarrow D'' \cup d$
      endif
    endif
  endfor
  $D'_{ord} \leftarrow Bisecting(D')$
  $D''_{ord} \leftarrow Bisecting(D'')$
  if sim(right(D'_{ord}),left(D''_{ord}))
      $\leq$ sim(right(D''_{ord}),left(D'_{ord})) **then**
    $D_{ord} \leftarrow D''_{ord} \oplus D'_{ord}$
  else
    $D_{ord} \leftarrow D'_{ord} \oplus D''_{ord}$
  endif
  return D_{ord}
else
  return D;

Figure 4.8: Bisecting top-down clustering algorithm

k-scan algorithm

Input:
The set of documents D
The number of sequences k

Output:
k ordered sequences representing the final order

$result \leftarrow \oslash$
Sort D by descending lengths;
for $i = 1 \ldots k$ **do**
  $center \leftarrow$ first(D);
  for all $d_j \in D$ **do**
    $sim[j] \leftarrow$ similarity$(center, d_j)$;
  end for
  $M \leftarrow \frac{D}{k} - 1$ documents with higher sim value;
  $result \leftarrow result \oplus \{center\} \oplus M$
  $D \leftarrow D \setminus (M \cup \{center\})$
end for
return $result$;

Figure 4.9: k-scan clustering algorithm

for the Jaccard distance and $[7.24, 7.32]$ for the inner product. Comparing the worse values in these ranges to the original (random) collection order 7.25 (7.65), the bisecting method obtains a gain in compression ratio of 5.24%(10.19%) for the Jaccard Distance and - 0.97%(4.30%) for the inner product.

Table 4.8: k-scan algorithm performance in average bits per gap, LATimes collection

	k										
	100	200	300	400	500	1000	2000	3000	30000	...	131896
Inner product	6.79	6.72	6.70	6.67	6.66	6.62	6.60	6.60	6.70	...	7.65
Jaccard	6.82	6.74	6.70	6.68	6.67	6.64	6.63	6.62	6.71	...	7.65

Table 4.9: k-scan algorithm performance in average bits per gap, FBIS collection

	k										
	100	200	300	400	500	1000	2000	3000	30000	...	130471
Inner product	6.46	6.39	6.36	6.34	6.33	6.29	6.23	6.21	6.26	...	7.35
Jaccard	6.54	6.47	6.44	6.41	6.39	6.29	6.23	6.22	6.27	...	7.35

The main point to consider is that the gains with respect to the natural order are very small, or they may not exist. This observation was ratified by the results of the bisecting algorithm in the FBIS collection, which we skip here. Comments on the difference between the results obtained with the two different measures of similarity for every experiment can be found at the end of this section.

Table 4.8(4.9) shows the final compression results measured in bits per gap for the LATimes (FBIS) collection. The first row presents the data obtained measuring the similarity with inner product and 200 dimensions in the reduced space, which are exactly the same conditions as those of the experiments with the c-blocks (section 4.7) and the bisecting algorithms. The second row shows the results with the Jaccard distance. The k parameter stands for the number of scans of the algorithm. Overall, the results give good compression values for a large enough k value (1000). However,

Table 4.10: k-scan + TSP combination bits per gap (δ coding), LATimes collection

	k				
	100	200	300	400	500
Inner product under 200 dimensions	6.34	6.35	6.36	6.37	6.37
Jaccard	6.25	6.28	6.29	6.30	6.30

Table 4.11: k-scan + TSP combination bits per gap (δ coding), FBIS collection

	k				
	100	200	300	400	500
Inner product under 200 dimensions	5.93	5.94	5.95	5.96	5.97
Jaccard	5.84	5.87	5.87	5.89	5.89

considering the variation of k, there is a point where the bits per document gap stops decreasing, and slightly increase until reaching the limit value where $k = N$, which is just ordering the collection by document length.

It is remarkable that the best values obtained with the k-scan algorithm in the LATimes collection (6.60) with the values given by the c-blocks algorithm are similar (6.68). This is not true for the FBIS collection, where the c-blocks (5.98) algorithm obtains a better value than the k-scan (6.23). In terms of gain with respect to the original compression ratio (6.35) this would be 1.9% vs 5.9%. These comparisons are fair, as the metric used by the two algorithms in this measure is exactly the same (inner product in the $k = 200$ space).

4.8.4 TSP and clustering combination

Finally, we present a technique that combines the best of the previous approaches: the real performance in bits per gap of the TSP and the efficiency in time consumption of clustering. The basic idea is the same as in the c-blocks algorithm (see section 4.7), namely split the collection into several parts and run the TSP in each part, so the overall running time is lowered from several hours to just minutes. The difference lies in the way the blocks are chosen; instead of driving a *raw* partition of equally-sized blocks, we select each sub-collection with the k-scan algorithm. This way, it is expected that each partition holds similar documents, so the TSP would improve an already good result.

Tables 4.10 and 4.11 show the results obtained for the LATimes and FBIS collection for this technique, respectively. The performance of the algorithm is approximately the same as running a Greedy-NN (TSP for the whole collection) with a 200 dimension reduced space (the best results obtained so far). One point to consider is that this technique obtained better results than the c-blocks in both collections. This proves that the TSP-based techniques perform well when the collection brings a *natural order* with a good compression ratio, which is harder to improve with any algorithm, but even more with clustering approaches tested before. Also, the reordering times for the values of k shown are in the order of the c-blocks, because the partitions are size balanced. Of course, this also applies to the algorithms described in the previous section, although this particular technique performs a reorder of each sub-collection in addition to computing the clusters.

It is possible to notice that the Jaccard measure performs better than the inner product in the bisecting and k-scan + TSP algorithms, although the inner product behaves slightly better for the k-scan, especially with low values of k. The problem with the Jaccard distance is that it is an undefined measure in the SVD reduced space, because it is related to the intersection of the original terms. However, the results with the inner product are useful for comparing fairly the clustering and c-blocks techniques.

4.9 Discussion

The experiments and results conducted through this chapter show empirically that the TSP heuristic is a good reordering strategy. This is supported by the implementations, tests and comparisons with clustering algorithms and in particular by the combination of the two main techniques. As a summary of the results presented in this chapter, figures 4.10 and 4.11 show some of the results obtained by the algorithms described here. The plots give some details on the effectiveness and efficiency tradeoff that can be achieved with the TSP and cluster-based combinations. The x-axis shows reordering and re-compressing time in a logarithmic scale, and the y-axis represents the bits per gap achieved for δ coding. Each line in the plot stands for a different reordering technique, namely TSP (section 4.5), c-blocks (section 4.7), k-scan (section 4.8.2), bisecting (section 4.8.2) and the TSP and clustering combination using k-scan (section 4.8.4).

For example, it is possible to achieve reduction in the compression ratios of 13.24% (6.29 bits per gap) (LATimes, δ coding) with a time cost of more than 8 hours driving a TSP algorithm only, while the TSP+k-scan combination gives a ratio of 12.55% (6.34) with much less computational effort (around 10 minutes).

In terms of reordering time vs compression ratio, the TSP and kscan combination is the method that brings a better performance in both collections, whereas TSP-only running times make the technique less attractive in real settings. The results of k-scan and c-blocks are acceptable (but worse than k-scan combined with TSP). K-scan performs better than c-blocks in the LATimes collection but worse than c-blocks in the FBIS collection. Finally, bisecting does not provide any significant reduction in bits per document gap with respect to the original ordering of documents.

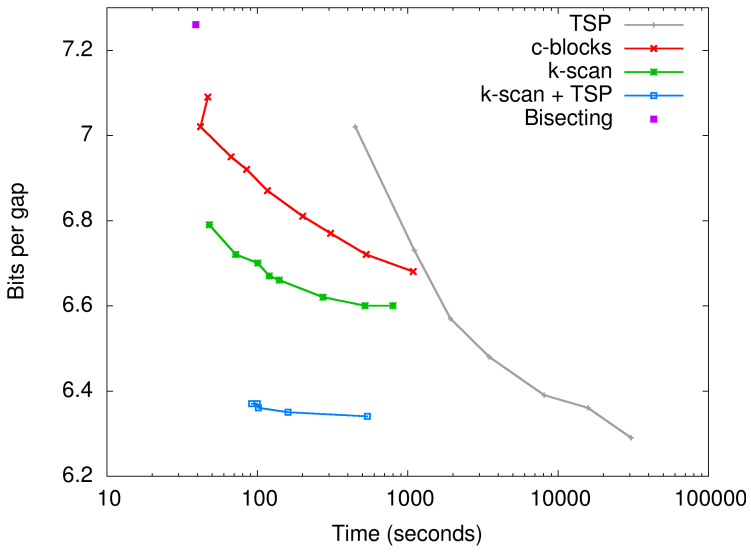


Figure 4.10: Compression vs reassigning time summary results (δ coding), LATimes collection.

Despite of the fact that the TSP-based technique performs better than other approaches and that it can be adapted to other situations to fit time consumption and memory usage (SVD reduction, collection partitioning, and combination with clustering) it does not necessarily give the optimal solution to the problem of reassignment of document identifiers.

In order to find better solutions than those provided by the TSP strategy, it will be necessary to discover heuristics that take into account the exact cost function to be minimised, just as in section 4.3. For a precise characterisation of this cost, see equation 4.12.

In this context, considering the problem a TSP, although producing good results, is only a strategy to address the document identifier reassignment problem. However, it is possible to give an explanation of why it performs well in reducing the d-gap log.

Consider the distance matrix $L_{d\times d}$ where each l_{ij} measures the distance between documents

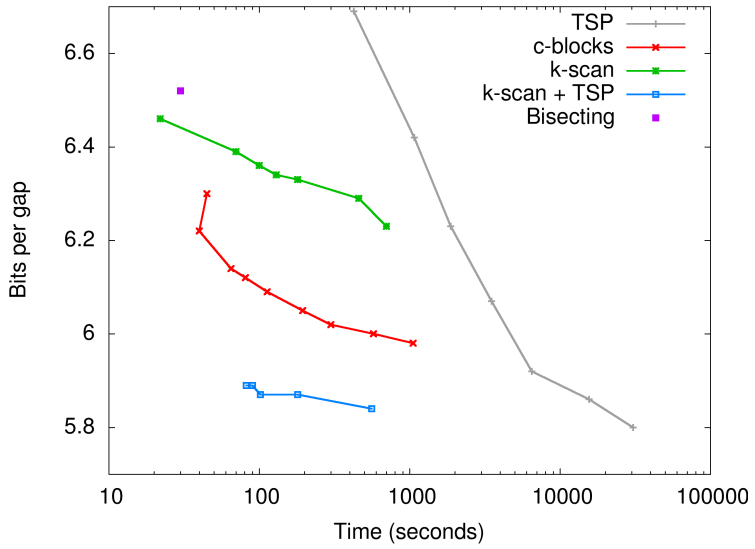


Figure 4.11: Compression vs reassigning time summary results (δ coding), FBIS collection

i and j. Considering the documents as sets of terms, the distance can be measured as $|(d_i \cup dj) \setminus (d_i \cap d_j)|$. The traversal found by the exact solution of the TSP minimises the function $sumdist(\pi) = \sum_{i=1}^{D-1} l_{\pi(d_i)\pi(d_{i+1})}$. Imagine the inverted file as a $t \times d$ binary matrix B, where a 1 in position b_{ij} represents an occurrence of the term t_i in document j. For each row of this binary matrix, each time a 0-1 or 1-0 switch appears, $sumdist(\pi)$ increases by one, so the order that minimises $sumdist(\pi)$ is the one which minimises this number of switches. This would maximise the bitwise intersection of document vectors, but it does not imply minimising the average d-gap logarithm sum. However, this ensures that the final configuration of the binary matrix has a big number of consecutive 1s in each row, so it is reasonable to expect a low average logarithm d-gap sum (as the logarithm does not penalise bigger d-gaps as much as it benefits from smaller ones).

In fact, it follows an example where the exact solution provided by the TSP does not minimise the d-gap products.

If we use as the similarity measure the inner product, then a solution to the TSP is the one that minimises the distance between document in consecutive positions, i.e., that which maximises the similarity between consecutive documents. Consider the following 4×4 matrix, where the columns are document vectors and the rows represent the presence/absence of each term in every document.

$$\begin{pmatrix} d_1 & d_2 & d_3 & d_4 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

The traversal $tr_1 = \langle d_1, d_4, d_2, d_3 \rangle$ is a solution to the TSP as it maximises $\sum_{i=1}^{N-1} e(v_i, v_{i+1})$, the similarity sum, obtaining a value of 4.

$$\begin{pmatrix} d_1 & d_4 & d_2 & d_3 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

However, the traversal $tr_2 = \langle d_2, d_4, d_1, d_3 \rangle$, which is not a TSP solution, is better than tr_1 with respect to the logarithm d-gap sum. For the traversal tr_1 the sum of the d-gaps logs is 3.59, and for tr_2 is 2.

$$\begin{pmatrix} d_2 & d_4 & d_1 & d_3 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

It is possible to trace a new line of active research, coming up with new heuristics based on the optimal cost function and/or the characterisation of the distance that a particular solution achieves with respect to the optimum. This also holds for TSP solutions: as long as this algorithm in the reduced dimension space performs well, we may pursue a formal characterisation to the distance of the optimal solution reached, with this sort of heuristic solutions.

The combination of clustering and TSP would also hold the same scalability problem as the TSP-only approaches described in Shieh et al. [2003] in very large collections. The combination of TSP, SVD and clustering may be suitable for these cases, and it would be interesting to measure the gains of applying the TSP in each cluster, instead to the whole collection. Moreover, this is especially indicated for collection partitioning techniques that may produce unbalanced clusters, such as the *k-means* algorithm (Jain et al. [1999]). The negative effect of large clusters would probably be softened with the SVD, so the TSP technique may run with acceptable times.

Another scalability issue is the efficiency of the SVD step. As we have seen, the dimensionality reduction allows efficient implementations of the TSP-based approaches, but when dealing with Web scalability, when several Gigabytes or even Terabytes are involved, it becomes necessary to come up with more efficient SVD implementations, as Kokiopoulou and Saad [2004], or other different matrix transformations, with similar properties as the SVD, but more efficient in computing time.

4.10 Summary and future work

This chapter gave many insights on the organisation of indexes of textual collections. In particular, we looked into the disposition of document identifier in a real testbed, where the notions of *clustering* and *bursts* of identifiers appear. Based on this, we formalised and characterised the document identifier reassignment problem, which is a form of rearranging the index in order for the compressibility to be maximised. The problem indeed falls in the NP-hard class. On the basis of those insights, many graph-based heuristics that may proved useful for other tasks can be applied to tackle this very same issue. Next, we presented a smart approximation for the document identifier reassignment problem by using a previous dimensionality reduction with SVD. The results presented allow to report time-efficient methods that yield good inverted file reduction gains. Specifically, we implemented the TSP Greedy-NN approach in the reduced dimension space and one variant, that applies this solution to sub-collections of the original data, reordering them next. We also presented an implementation of two clustering techniques to the problem of reassignment and evaluated and compared them with TSP-based reordering. Finally, in the discussion section we showed a comparison of the different techniques and gave insights into both theoretical aspects and scalability issues of the proposed work, as well as future research lines.

Chapter 5

Static Pruning of Inverted Files

This chapter is devoted to a lossy index compression mechanism, known as *static pruning*. Usually, techniques for reducing query times try to maintain retrieval effectiveness whilst optimising the accesses and operations over the data. This process can result in considerable time savings. The main rational behind static pruning is that most users only browse a short amount of top results, just like dynamic pruning (max-score for instance). However, static pruning techniques use this claim off-line, without any query information.

The first advantage is that the index size can be reduced without precision loss in the top results, which is the claim in the techniques presented so far. However, the fact that the algorithm can operate before querying time, opens ground for developing a more detailed index analysis. In fact, static pruning can be beneficial for retrieval effectiveness, if handled with care.

This chapter begins by revisiting prior work on index pruning (section 5.1), with particular focus on a well-known standard pruning technique, which we will be referring to as *Carmel's method* (section 5.1.1). The algorithm poses a series of issues on its own, which can be addressed to enhance retrieval performance. Some of these issues concerning its algorithmic design are revisited in section 5.2 . Although the original approach is able to retain precision when a considerable part of the inverted file is removed, we show that it is possible to improve precision in some scenarios if some key design features are properly selected.

Section 5.3 presents an alternative way for static pruning. It addresses the problem of identifying collection dependent stop-words in order to reduce the size of inverted files. We present four methods to automatically recognise stop-words, analyse the tradeoff between efficiency and effectiveness, and compare them with a previous pruning approach. The experiments allow us to conclude that in some situations stop-words pruning is competitive with respect to other inverted file reduction techniques.

Finally, section 5.4 presents a novel pruning technique based on the probabilistic model of IR. More specifically, we employ the Probability Ranking Principle as a decision criterion over which posting list entries are to be pruned. Moreover, this approach requires the estimation of three probabilities and combines them in such a way that we gather all the necessary information to apply the aforementioned criterion. In brief, it defines a framework in which the *goodness* of pruning is characterised based on the belief of some estimations, instead of being fixed to any specific scoring function.

5.1 Static and Dynamic pruning

IR systems need to be efficient to keep low response times. Efficient performance is usually achieved by employing suitable data structures (inverted files Witten et al. [1999]) and policy-driven memory caches (Baeza-Yates et al. [2007]) and by using adequate algorithms to access these data structures (e.g. posting file ordering Anh and Moffat [2002]).

Index pruning can be dynamic or static. Dynamic index pruning aims to improve system efficiency by *stopping* the inverted file scanning during query time, when some criterion is satisfied. Some

```
      Prune(k,ϵ)
1:    for every term t do
2:        {docs} = posting(t)
3:        if |{docs}| > k
4:            ∀D ∈ {docs}
5:                A(t, D) = score(t, D)
6:            z_t ← k-th highest entry in row t of A
7:            τ_t ← ϵ · z_t
8:            ∀D ∈ {docs}
9:                if A(t, D) < τ_t
10:                   remove entry D from posting(t)
```

Figure 5.1: Carmel et al. Top-k algorithm. The procedure takes two parameters, ϵ and z_t and discards the posting entries that score less than $\epsilon \cdot z_t$, where z_t is the k-th highest score in the whole posting.

examples of dynamic pruning are the MAXSCORE optimisation (Turtle and Flood [1995]), or its more recent incarnation described in Strohman et al. [2005]. Anh and Moffat [2006] presented a different approach in which document pointers have to be sorted according to *impacts*. These impacts reflect the partial contribution of the term occurrence in a document, and provide fast index scanning. The work by Theobald et al. [2004] presents a dynamic pruning method that preserves the top-k results (when using the pruned or unpruned index) using a probabilistic score prediction for each query term and managing efficiently priority queues. Dynamic pruning is a very efficient strategy for reducing answering times (Zobel and Moffat [2006]) without compromising the retrieval performance.

Static pruning consists of compressing the index size, by pruning entries from the postings file, according to some criterion. Unlike dynamic pruning, this is done off-line and is query-independent. Static pruning is a *lossy* compression technique, because it is not possible to bring the compressed data back to its exact uncompressed form.

Static pruning has two advantages: it reduces not only query time, but also disk occupancy, and it is query-independent, hence it can be done off-line. The trade-off between efficiency and effectiveness is of outmost importance for static pruning approaches.

Static index pruning attempts to discard the least important data from the index. This data is estimated according to some relevance criteria (Carmel et al. [2001b]). A number of proposals have been made to address static pruning, which can fall into two different categories: term-based and document-based.

Document-based pruning discards the terms from a document that are less representative of the document's content. The main advantage of document-based pruning is that it can be applied on-the-fly while indexing the collection (Büttcher and Clarke [2006]). However, on-the-fly pruning requires estimating collection statistics using only partial subcollections, which may not be accurate enough. Term-based pruning reduces the index size by discarding the term occurrences that are less likely to affect the retrieval performance of a specific retrieval model.

Index pruning is uniform when it is applied to all the terms or documents in the same way. A first detailed overview of static index pruning methods is given in Carmel et al. [2001b], where an experimental overview of uniform and term-based pruning is presented. The work presented in Carmel et al. [2001b] is extensively used in all throughout this chapter and presented in detail in section 5.1.1.

Other static pruning techniques are discussed next. Blanco and Barreiro [2007a] presented a comparison on four collection-dependent algorithms for pruning low-contribution terms and their impact in the efficiency and retrieval performance. Büttcher and Clarke [2006] proposed a document-centric pruning approach which uses a query expansion technique based on Kullback-Leibler (KL) divergence. Their algorithm produces excellent outcomes in terms of query processing speed gains. Results were reported on the .GOV2 collection, which is a 428 Gigabytes collection, and effectiveness figures were reported on 50 short queries. Another approach for performing document-based pruning

Collection	size	Topics	# queries
TREC disks 4&5	2G	301-450+601-700	250
WT2G	2G	401-450	50
WT10G	10G	451-550	100

Table 5.1: Collections and topics employed in the *boosting static pruning* experiments

is to replace the documents by their respective summaries. Summary indexing seems to improve precision while incurring in a large recall loss (Brandow et al. [1995], Sakai and Sparck-Jones [2001]). The work by Ntoulas and Cho [2007] addresses the issue of resource handling for the pruning of terms and posting entries, by keeping the unpruned index on disk and determining the conditions and pruning level necessary to keep the top results the same, under a uniform pruning regime.

This work proposes a different approach, based solely on relevance estimations and not retrieval models. It is shown that our proposed algorithm is both theoretically sound and also beneficial to retrieval.

5.1.1 Carmel et al.'s method

The algorithm of Carmel's method (see figure 5.1) aims at preserving the top results when either the original or the pruned index is used by a retrieval system. It is an *idealised pruning algorithm* that ensures the similarity of the top k returned results for queries of less than $r = 1/\epsilon$ terms, where k and ϵ are parameters. This property holds if the scores assigned by the IR system are the same before and after pruning. The procedure operates on a term-by-term fashion, selecting which term-document pairs are ruled out from the index on the basis of their scores (in the original paper TF-IDF). For every term in the dictionary, the algorithm computes the scores of the documents contained in its posting list: this reflects the partial contribution of the term in any query it appears. Then, the method selects the k-th highest score z_t and sets $\tau_t = \epsilon * z_t$. Every document-term pair in that posting that scores lower than τ_t is discarded from the index.

Finally, although the algorithm preserves the similarity of the top k returned results, which is a nice theoretical property, experiments reported on the LATimes collection showed that the pruning levels achieved in practice are negligible Carmel et al. [2001b]. The most probable reason for this behaviour could be the narrow range of values of TF-IDF; the lowest scored entry in each posting is almost always larger then the cutoff threshold z_t.

In order to obtain any significant index reduction, it is necessary to shift every term-document score in the index, by subtracting the minimum score of the index from every document score. In practice, the pruning algorithm is applied after this ad-hoc modification of the inverted file. This accomplishes excellent results but the property of preserving the top-k results despite of the index used (pruned or unpruned) is not guaranteed.

Carmel's pruning method has been further investigated repeatedly Carmel et al. [2001a], Büttcher and Clarke [2006], Blanco and Barreiro [2007b], and is considered state of the art in index pruning.

In the following sections, we demonstrate how not imposing any similarity preserving conditions can be beneficial for retrieval if done with care. In any case, for the approaches presented next, retrieval quality is measured in terms of precision and not by document ranking similarity.

5.2 Boosting static pruning

The algorithm just presented in section 5.1.1 yields excellent results at keeping the top-k answers for a given query and scoring function (Smart's TF-IDF), making this approach suitable for high-demanding environments. As seen, it involves two parameters, k and ϵ, that set the number of top-documents (k) that are scored the same (within an error of ϵ) whether the original or the pruned inverted file is used, for any query less than a certain size ($\frac{1}{\epsilon}$). Hence, the rankings produced are similar. Results

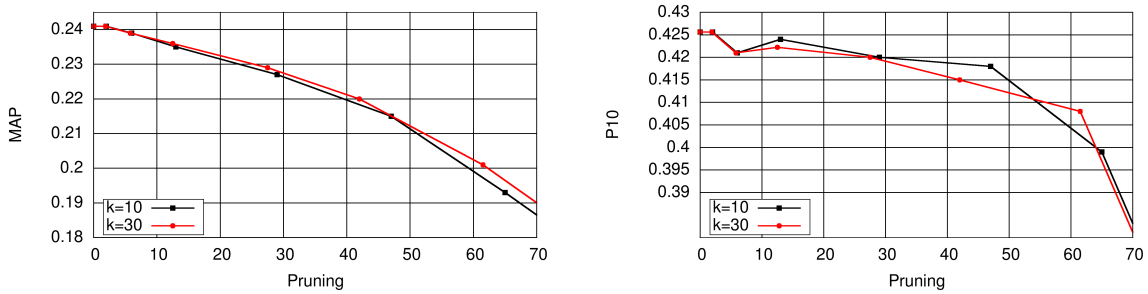


Figure 5.2: Behaviour of the pruning algorithm with respect to the choice of different k values, TREC Disks 4 & 5, MAP and P@10

presented by Carmel et al. [2001a] and Carmel et al. [2001b] prove that the technique conveys excellent results for maintaining precision when a high quantity of the data is removed.

However, by issuing some key features, the technique can go further than just faithfully preserving the document ranking, and is able to increase precision values for some pruning levels and under certain conditions (Blanco and Barreiro [2007b]).

Next, we address some features not considered in the original model and assess their impact in static pruning performance through TREC-style intensive evaluation. The following experiments use a probabilistic-based scoring function (BM25) (section 2.5.1.3) instead of Smart's TF-IDF (section 2.5.1.2). The parameters for the BM25 function are set to their recommended values (Robertson et al. [1995]): $k_1 = 1.2$, $k_3 = 1000$ and $b = 0.75$.

5.2.1 Prior Experiments and Results

The first batch of experiments was carried out with three different document/topics sets, described in table 5.1, and measured using P@10 and MAP. Terms were stemmed using Porter's algorithm. As well, three types of queries were considered: short (title only), medium (title + description) and long (title + description + narrative).

Following the original description of Carmel's pruning algorithm, k and ϵ must be selected carefully, as they hold important properties regarding to ranking similarity before and after pruning. In particular, ϵ should be low.

This section considers k and ϵ just as parameters that determine the number of pointers removed (percentage of pruning), without focusing on theoretical guarantees.

Choosing different k values yields very correlated results, although higher values ($k = 30$) are preferable for obtaining better MAP values whereas lower ($k = 10$) values seem more adequate if a good behaviour at P@10 is desired, which goes accordingly with the top-k preserving property construction of the algorithm. Figure 5.2 is an example of such behaviour, although in general any reasonable choice of k values (< 50) is likely to yield very similar results.

Some of the other considerations taken into account are summarised next.

5.2.1.1 Updating document lengths

Following the description of the *idealised* pruning algorithm by Carmel et al. [2001b], document lengths (*dl*) should be the same in both the original and the pruned inverted file. This is needed for the ranking similarity guarantees of the algorithm to hold. In particular, the score of the matching function should be the same before and after pruning in order to assure the ranking-preserving theoretical guarantees. Pruning affects collection statistics (term document frequency most notably) and the updated new document sizes (which should remain the same in order to preserve the aforementioned guarantees).

On the contrary, these series of experiments are performed under the belief that collection and document statistics should be updated when pruning. From a certain point of view, it makes sense to

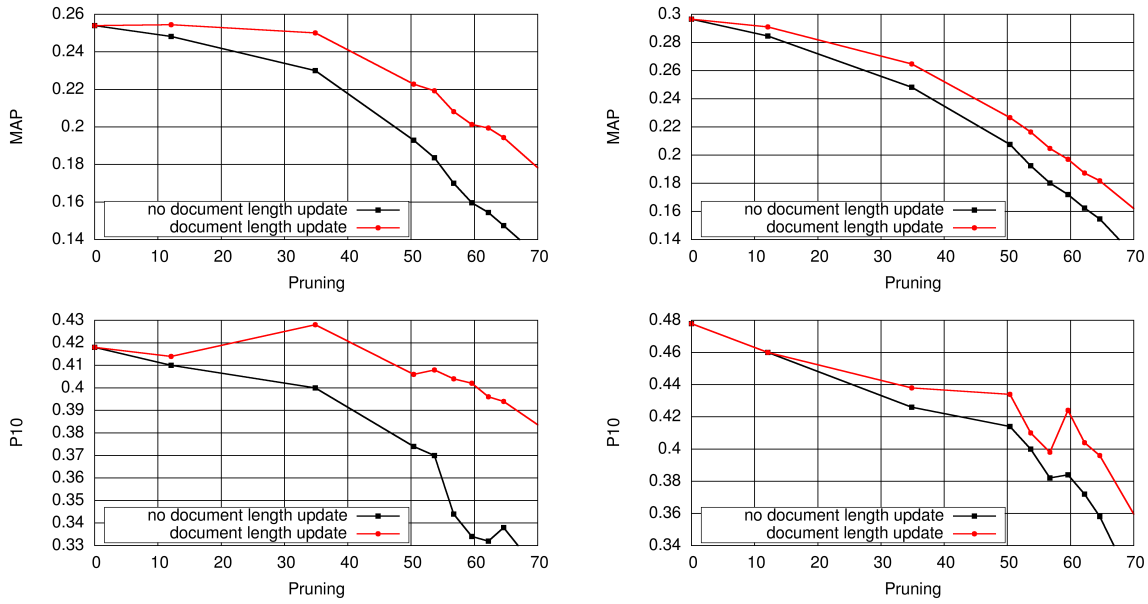


Figure 5.3: Carmel et al's. method, WT2G, MAP for short(left) and long(right) queries BM25 retrieval updating and not-updating the document Lengths.

update them in order for the inverted file to be *coherent*. Weighting functions are modelled under some assumptions over those statistics, which might be breaking up if the information contained in the index does not reflect the statistics in use. For instance, one document might be pruned much more than others (with many non-relevant keywords), and not updating its document length would imply not retrieving it even if the remaining terms match some query.

Experiments demonstrated that the best pruning and precision tradeoffs are obtained when the document lengths are updated.

Figure 5.3 presents a simple experiment comparing the performance of Carmel's pruning method while updating document lengths and not updating them. It is clear that updating the collection statistics in every pruning method brings better results, for both short and long queries. Note that the top-k similarity preserving condition is therefore no longer guaranteed, but it performs better, with respect to MAP and P@10.

5.2.2 Additional considerations

Terms with document frequency $df > N/2$ can be discarded automatically from the inverted file, where N is the total number of documents. This value comes naturally from BM25's idf formula as those terms have a negative score for every document.

$$\log\left(\frac{N - df + 0.5}{df + 0.5}\right) , \tag{5.1}$$

It turned out that ruling out terms with $df > N/2$ is a good option. This holds true especially for long queries, where those terms are more likely to appear.

In the original work (Carmel et al. [2001b]) it is stated that every score must be shifted, otherwise the pruning level obtained is negligible. This shifting is done by subtracting the global minimum score. As we considered ϵ as just a parameter, we omitted the shifting step. In any case, the shift produced negligible effects in all the experiments carried out.

A last design consideration was whether to update the average document length *avgdl* in the pruned inverted file or not. Correcting this value portrays a new term frequency (*tf*) normalisation factor. Updating the average document length performs slightly better than not doing it for long

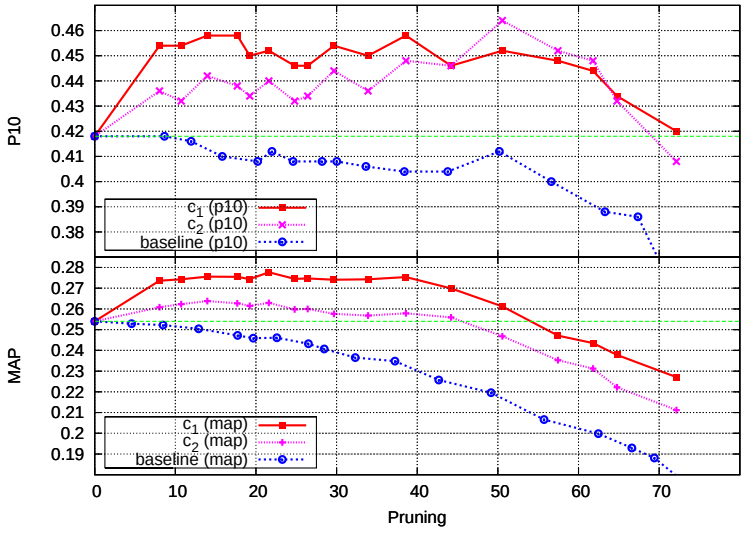


Figure 5.4: MAP and P@10 for short queries at different pruning levels, baseline and different settings (WT2G collection)

queries, whereas for short queries updating is outperformed by not doing it. Therefore, the new term frequency normalisation introduced by the algorithm seems adequate for a short-queries scenario. Section 5.2.3 introduces a formal analysis of this fact.

In section 5.4 an algorithm that follows the considerations presented in section 5.2.1.1 is evaluated thoroughly and employed as a baseline.

To exemplify some of the previous claims, we comment next some of the precision vs. pruning results using standard MAP and P@10 measures. In order to illustrate the effects of some of these design considerations figure 5.4 shows three precision curves obtained using the WT2G collection and short queries. The baseline is the result using BM25, not updating the document lengths, not updating the average document length (*avgdl*) and not removing terms with $df > N/2$ (this setting is the one that retains better the original ranking). The other two curves are obtained removing terms, updating the *dls* (the best setting found empirically), and c_2 updates the *avgdl* while c_1 does not. With the baseline settings, the method is better for maintaining the top-k results, which is easier for shorter queries, but the precision curves are decreasing. On the other hand, a different parameter selection improves significantly over the original precision values, up to a 50% pruning level for MAP and 70% for P@10.

Overall, using a suitable combination of parameters, pruning is able to improve MAP and P@10. Considering the precision values obtained with the original not pruned inverted file as our baseline, it can be retained up to a 35-50% pruning level in most cases, although the method tends to favour short queries and the P@10. In that case, MAP(P@10) is over the baseline up to a 30-50%(60-70%) pruning level. Maximum improvements go up to 12% for MAP and 10% for P@10. These values are collection-dependent and lower with long queries. The behaviour is better in web collections than in disks 4&5, where MAP improvements are very small, even though the original precision values can still be maintained up to a 40-60% pruning level.

Parameter selection is crucial, and different algorithm settings lead to totally different performances. P@10 presents a good behaviour with any query size, whereas MAP behaves better with short queries. Some parameter combinations (document length update, selective term removal) are consistently better than the rest, although the improvements are more noticeable in the web collections (WT2G especially). Finally, the curves are not always monotonically decreasing, and

high pruning values may increase precision, probably due to the score function ranking high *bad* document-term pointers, which are removed at those pruning levels.

5.2.3 Document normalisation effect

This section shows that the effect of pruning updating the *dl* and without updating the *avgdl* in BM25's score, is to alter the *tf* contributions in a particular fashion. The *tf* normalisation in BM25 is

$$tf_n = \frac{tf}{nf}\ , nf = (1-b) + b\frac{dl}{avgdl}\ , \tag{5.2}$$

and $b \in [0:1]$ is a constant (typically 0.75). If α terms are removed from a document by the pruning algorithm, the new normalisation factor would be

$$nf' = nf - b\frac{\alpha}{avgdl} \tag{5.3}$$

This has the global effect of softening the document length contribution, and it can be compared with selecting a lower b value. In this case, $b' = b - \beta$ which implies that

$$nf' = nf - \beta\left(\frac{dl}{avgdl} - 1\right)\ , \tag{5.4}$$

and hence both normalisation factors have the same analytical form for long documents ($dl > avgdl$), and different otherwise.

5.2.4 Summary

This section outlined and experimented with several new variants of the most well-known inverted file pruning algorithm by Carmel et al. [2001b], and stated how it is possible to tweak the technique so that the precision is improved when some information is removed selectively. As a main conclusion, discarding pointers from an inverted file off-line and independently of the queries is able to devise satisfactory results, efficiency and effectiveness-wise, for ad-hoc retrieval. Also, carefully addressing the algorithmic issues involved brings some new intuitions regarding weighting schemes or term frequency normalisation.

5.3 Static pruning of terms

This section presents several techniques for reducing the size of the inverted file by identifying a stop-words set dependent on the collection. The main difference between this method and the one described in section 5.1.1 is that the whole term is removed from the index instead of deleting single occurrences. We introduce several techniques based on the terms' *informativeness value*, in particular inverse document frequency (*idf*) and residual inverse document frequency (*ridf*), and a novel method based on the *term discriminative value*. Discarding a whole term implies that the index term is not useful in every possible context (query). Although this claim may seem too aggressive (or naive), except for a predetermined and well-known set of function words, we found out that in some scenarios these algorithms prove to be competitive or even better than the methods based on the pruning of term-document occurrences. Other work (Carmel et al. [2001b],de Moura et al. [2005]) size the amount of pruning as the percentage of pointers removed from the inverted file, but Carmel et al. [2001b] advanced that it is not known how static pruning would behave in conjunction with the traditional lossless compression methods, and that further research was needed in order to clarify this issue. This section also presents the experiments and results assessing the relationship between the amount of pointers and the real space savings, for five well known coding algorithms. In summary, it is possible to predict a good and stable behaviour of the static pruning methods for every coding scheme tested. Experiments also report on query times in a real retrieval platform.

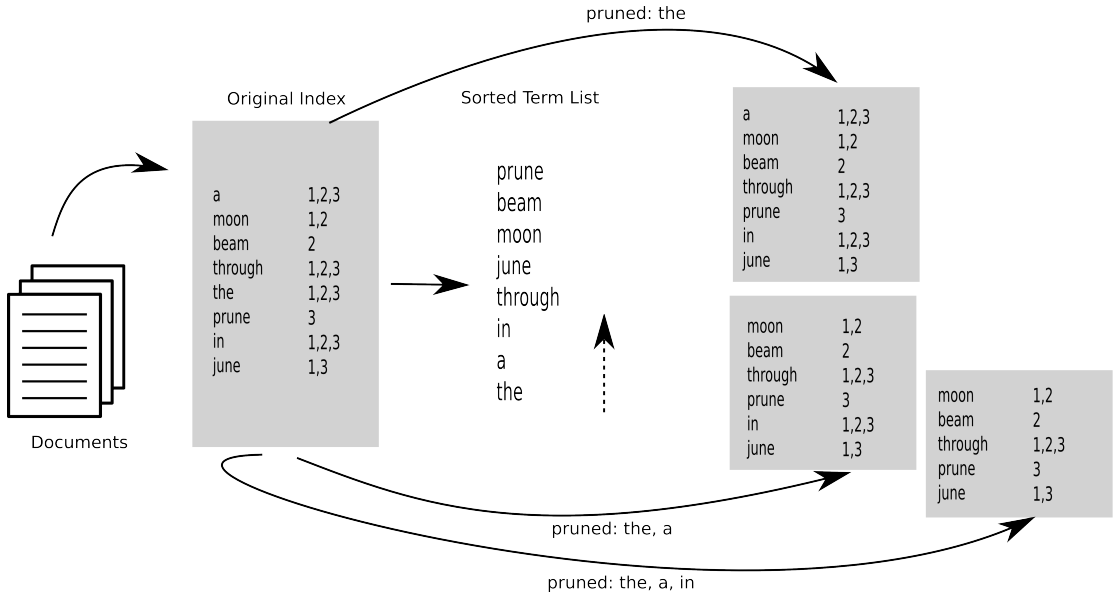


Figure 5.5: Procedure for pruning terms in an index. First the terms are ranked according to some method and then the pruned indexes are obtained by removing the tail terms

The rest of the section is organised as follows: section 5.3.1 introduces the term pruning methods, section 5.3.2 presents techniques for term pruning based on *informativeness* and section5.3.3 based on *discriminativeness*. The experiments and results are presented in section 5.3.4 and the section ends with a conclusions and further work section 5.3.5.

5.3.1 Static index pruning of term posting lists

Traditionally, stop word removal aims at identifying noisy terms that may hurt precision, and to the best of our knowledge it has not been used for efficiency purposes.

It is clear that removing high-frequency terms from an uncompressed inverted file may lead to substantial space savings, as they tend to engross most of the occurrences (according to Zipf's law). How this may affect to compressed inverted files is disccussed by Witten et al. [1999]. The claim is that the higher the frequency of the word, the better a parametrised compression model such as Golomb will adapt to it, so the less space it will consume in a compressed form. In general, it is a commonly accepted idea that stop-words should be in the inverted file since removing high-frequency words would result in very small space savings. However, we believe that if it is possible to obtain a good ranking of terms according to their *importance*, it would be interesting to establish the tradeoff between retrieval accuracy and the index reduction implied by the removal of the *less important* terms. In fact, some authors (Robertson and Walker [2000]) report that building a manual extended stop-list speeds searches. We propose to study this effect with techniques that obtain *informativeness* (5.3.2) and *discriminative* (5.3.3) rankings.

In general, all these techniques operate following the same procedure, which is depicted in figure 5.5: first the terms in the lexicon are sorted according to some criteria, and a specific pruning level is obtained by removing the n tail terms in the list.

5.3.2 Stop-words list based on idf and ridf

The inverse document frequency is a term informativeness measure, therefore it can be used to produce a ranking of bad terms (those with lower idf values). We used a common *idf* normalisation introduced by Robertson and Sparck-Jones [1976a] that performed well for identifying dynamic stop-words by Lo et al. [2005]. If D is the total number of documents in the collection, and df the

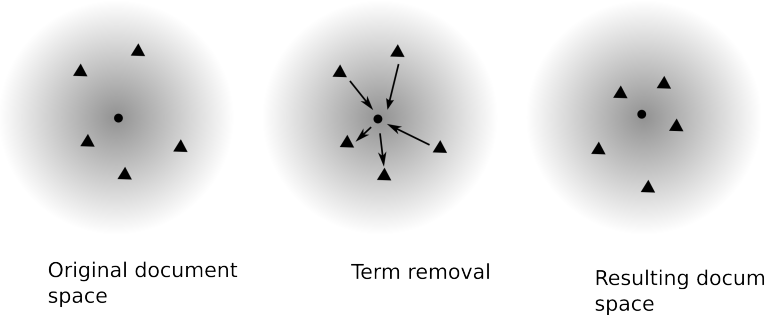


Figure 5.6: Effect of the removal of one term in the term-document space: distances between documents and centroid vary

number of documents the term t appears in (document frequency), then the *idf* for term t is:

$$idf = \log\left(\frac{D - df - 0.5}{df + 0.5}\right) \tag{5.5}$$

Residual idf is defined by Church and Gale [1995] as the difference between the observed idf (IDF) and the idf expected under the assumption that the terms follow an independence model, such as Poisson ($I\hat{D}F$). To the best of our knowledge it has not been used for identifying collection-dependent stop-words, although in Rennie and Jaakkola [2005] it is employed successfully for named entity recognition. If tf is the total number of tokens for a term t, then the ridf devised by a Poisson distribution is

$$RIDF = IDF - I\hat{D}F = -\log(\frac{df}{D}) + \log(1 - e^{-\frac{tf}{D}}) \tag{5.6}$$

Church and Gale [1995] claim that the more a term deviates from Poisson, the more dependent on hidden variables, and more useful the term is to discriminate between documents containing it on the basis of the hidden dependencies. In order to compute the *idf* and *ridf* values for every term appearing in the collection, it is only necessary to traverse the lexicon file once.

5.3.3 Stop-words list based on Salton's Term Discrimination Model

Salton's Term Discrimination Model (TDM) (Salton et al. [1975b]) is one of the first computationally attractive attempts to find an effective ranking of words, based on the analysis of the *Discriminative Value* (DV) of a term and it was used for automatic indexing. The model is embodied into the vector-space framework for Information Retrieval and its use has been limited to small collections (Cranfield, Medlars, Time). However, the usefulness of the model has not been clearly stated in the following years, nor it has been applied in large TREC collections. This section proposes to revisit the original model and to determine to which extend it may be worthy as a tool for finding stop-words.

The Term Discrimination Model measures the importance of every index term based on the influence it has on a document space. The main assumption is that a document space with distant vectors is preferable for retrieval. A good document space is one that maximises the average separation between every pair of vectors, because it would be easier to distinguish among the retrieved documents. Under this claim, and given that terms act as dimensions of the document space, it is possible to rank the index terms according to how much each term affects the *density* of the vector space, i.e. how good as *discriminators* they are. The DV of a term t is defined as how much the removal of t from the vector space decreases the total space density. Figure 5.6 is an graphical example of the effect TDM measures.

Let $\{t_1 \dots t_T\}$ and $\{d_1 \dots d_D\}$ be the term and the document set respectively, where every document d_i is represented by a term frequency component vector $\langle tf_{i1}, tf_{i2} \dots tf_{iT}\rangle$. The calculation of every document-to-document distance as a measure of the space density is computationally unaffordable for very large collections. One possible variation could be a definition of the density measure

related to documents-to-centroid distances. In this case, the DV for a term t_k is

$$DV_k = \sum_{i=1}^{D} \text{distance}(d_i^k, c^k) - \sum_{i=1}^{D} \text{distance}(d_i, c) = Q_k - Q, \tag{5.7}$$

where Q is the space density, Q_k is the space density after the term t_k is removed, d_i^k is the document obtained after removing the term t_k from d_i, c is the document centroid and c^k is the document centroid resulting after the removal of the term t_k.

A straight implementation of equation 5.7 is very time consuming. For every term, it requires the computation of the similarities between every document and the centroid, forcing to traverse T times a direct file of D documents. Next it follows a reformulation of equation 5.7 that allows to save most of the operations by storing some data in main memory and reducing drastically the total computation time. First, let TF_j^k (TF_j) be the j-th component of the centroid c^k (c):

$$TF_j = \frac{1}{D} \sum_{i=1}^{D} tf_{ij} \tag{5.8}$$

$$TF_j^k = \begin{cases} TF_j & \text{if } j \neq k \\ 0 & \text{if } j = k \end{cases} \tag{5.9}$$

Equation 5.7 can be rewritten as follows, where tf_{ij}^k is the j-th component of d_i^k.

$$DV_k = \sum_{i=1}^{D} \sum_{j=1}^{T} \frac{tf_{ij}^k \times TF_j^k}{|d_i^k| \times |c^k|} - \sum_{i=1}^{D} \sum_{j=1}^{T} \frac{tf_{ij} \times TF_j}{|d_i| \times |c|}, \tag{5.10}$$

Let $w_i = \sum_{j=1}^{T} tf_{ij} TF_j$, which is a value that can be precomputed for each d_i, then

$$\sum_{j=1}^{T} tf_{ij}^k \times TF_j^k = \begin{cases} w_i & \text{if } t_k \notin d_i \\ w_i - tf_{ik} TF_k & \text{if } t_k \in d_i, \end{cases} \tag{5.11}$$

and Q_k can be expressed as

$$Q_k = \sum_{i \setminus t_k \in d_i}^{D} \frac{w_i - tf_{ik} TF_k}{|c^k| \times |d_i^k|} + \sum_{i \setminus t_k \notin d_i}^{D} \frac{w_i}{|c^k| \times |d_i^k|} \tag{5.12}$$

Taking into account that

$$|d_i^k| = |d_i| \quad \text{if } t_k \notin d_i \tag{5.13}$$

and that

$$\sum_{i \setminus t_k \notin d_i} \frac{w_i}{|d_i|} = \sum_{i=1}^{D} \frac{w_i}{|d_i|} - \sum_{i \setminus t_k \in d_i} \frac{w_i}{|d_i|}, \tag{5.14}$$

then Q_k can be finally rewritten as:

$$Q_k = \frac{1}{|c^k|} \left(\sum_{i \setminus t_k \in d_i}^{D} \left(\frac{w_i - tf_{ik} TF_k}{|d_i^k|} - \frac{w_i}{|d_i|} \right) + \sum_{i=1}^{D} \frac{w_i}{|d_i|} \right) \tag{5.15}$$

Since Q is constant, the Q_k values will suffice to compute the rank produced by the TDM. The reformulation of Q_k introduced in equation 5.15 allows the computation of this rank with just one single pass to a direct file to calculate the w_i and $|d_i|$ values, and another one to the inverted file to recalculate every single term contribution. If we use the cosine normalisation, then $|d_i| = \sqrt{\sum_{j=1}^{T} tf_{ij}^2}$, $|c| = \sqrt{\sum_{j=1}^{T} TF_j^2}$, implying that $|d_i^k| = \sqrt{|d_i|^2 - t_{ik}^2}$, $|c^k| = \sqrt{|c|^2 - TF_k^2}$. Finally

we propose another last modification to this model, in which the contribution $\frac{1}{|c^k|}\sum_{i=1}^{D}\frac{w_i}{|d_i|}$ is dropped out from equation 5.15. This factor is dominant in the final value of Q_k and very dependent on the $|c^k|$ value. This is a problem in large collections because the method is too biased for high frequency terms (concretely on the factor TF_j appearing on $|c^k|$), ranking them higher.

This efficient implementation of the Term Discrimination Model requires $2|D| + |T|$ extra pointers to store the document lengths, the w_i (for each document) and the TF_j (for each term) values. Considering 16-byte double precision floats, these amounts sum up to approximately 12 MB for the 2 Gigabyte TREC web collection.

The approach described here will be referred as *tdm1* and we denote as *tdm2* another variation that employs a term frequency normalisation factor in the fashion of BM25 (Robertson et al. [1995]):

$$\hat{tf}_{ij} = \frac{(k_1+1)tf_{ij}}{tf_{ij} + k_1\left((1-b) + b\frac{len(d_i)}{avglen}\right)} \tag{5.16}$$

In equation 5.16, $len(d_i)$ stands for the number of tokens in the document d_i, $avglen$ is the average document length in the collection and we used the recommended values for $k_1 = 1.2$ and for $b = 0.75$. In the implementation of *tdm2* we considered the simplification of not recomputing the average document length every time a term is removed from the collections. Once the term frequencies are computed according to equation 5.16 the process follows as described for *tdm1*.

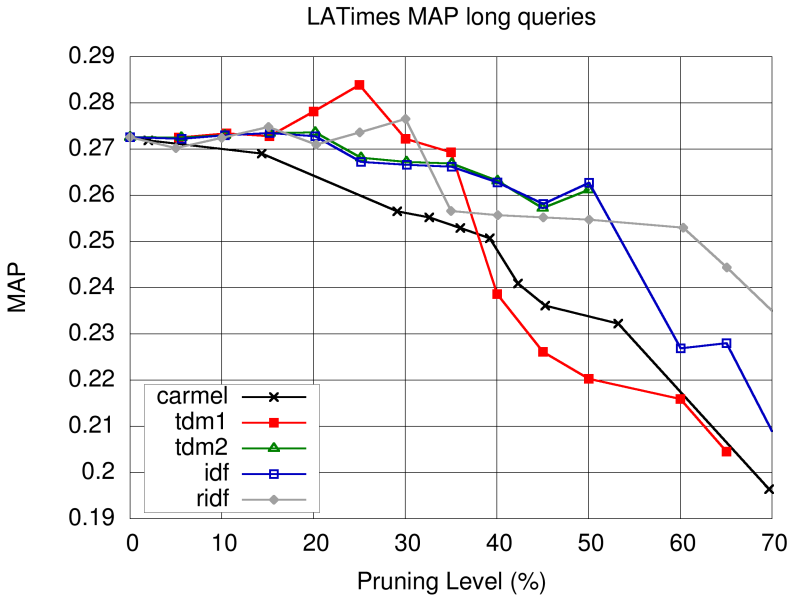


Figure 5.7: Term-based static pruning: MAP vs. %pruning, LATimes collection & long queries

5.3.4 Experiments and Results

Next we report some empirical findings using the five pruning methods described in sections 5.1.1 and 5.3.1. The evaluation tries to assess how the mean average precision (MAP) and precision at ten (P@10) (section 2.6) vary as the number of deleted occurrences from the inverted file increases. Intentionally, we chose settings that devise high precision values in order to measure the decrease in precision when augmenting the pruning level. We used Porter's algorithm for stemming. BM25 (equation 2.10) was selected as the scoring function for every method, as it has proved to be robust in the IR literature (see section 2.5.1.3, equation 2.10). In this section we employ the recommended values (Robertson et al. [1995]): $k_1 = 1.2$, $k_3 = 1000$ and $b = 0.75$.

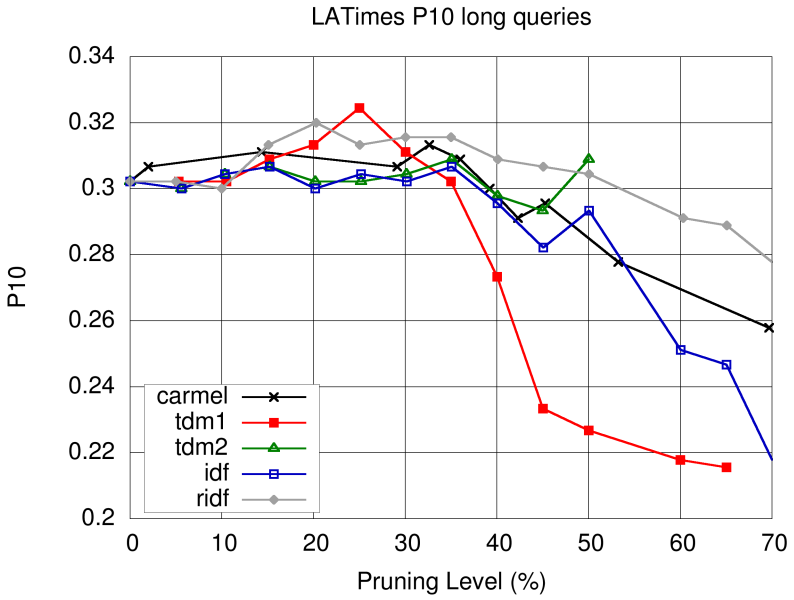


Figure 5.8: Term-based static pruning: P@10 vs. %pruning, LATimes collection & long queries

This experimentation was carried out with TREC topics from 401 to 450 in the LATimes and WT2G collections, short queries (title) and long queries (title plus description). Note that the narrative field was discarded as it hurts precision using these settings. Regarding Carmel's method, the k value was set to 10, and the different pruning levels were obtained by modifying ϵ.

For the TDM-based methods, another condition was taken into account in order to smooth the correlation between the frequency range and the discrimination value. We introduced a document frequency threshold based on the size the collection: only terms with document frequency in the collection greater than 400(2000) where pruned for the LATimes(WT2G) collection.

A second class of experiments try to assess the real tradeoff between the pruning level and the disk space occupied by the inverted file, using different posting-list compression methods. We experimented with five different coding algorithms (see section 3.3) for the document pointers: three non-parametrised methods (γ, δ, *variable byte*), a local parametrised method (*Golomb coding*), and a context-sensitive method (*interpolative coding*). Within-document term frequencies were coded with unary code, except for the case of variable byte where they were coded with variable bytes as well.

Finally, a third experiment measured the real query time performance of the system for one term-based method (*ridf*) and Carmel's method, to try to determine the final speedup effect of pruning on a retrieval platform.

Indexing and retrieval was carried out using the Terrier IR platform[1] v1.0.0, developed at the University of Glasgow. The pruning and compression program suite was implemented on top of it.

5.3.4.1 Retrieval effectiveness vs Index pruning

Figures 5.7 to 5.10 show MAP and P@10 results for the LATimes collection, for both short and long queries. The precision curves end when the number of terms deleted forces any query to be empty. In general, all the methods that prune terms are able to increase initial MAP and P@10 values. Overall, *tdm1* achieved the highest values in precision with a pruning level around 20%-30%. If Fox's stop-list Fox [1990] is applied the results are: MAP 0.2695(0.2524) and P@10 0.2933(0.2911) for long(short) queries at a 26.7% pruning level. The best values achieved with the *tdm1*, MAP 0.2839(0.2544) and P@10 0.3224 (0.3022), are better than those attained by Fox's stop-list. Term pruning methods present a good behaviour at certain levels, with ridf remarkably stable and smooth and *tdm1* be very

[1] http://ir.dcs.gla.uk/terrier

Table 5.2: Term-based static pruning: precision vs. %pruning, WT2G collection & long queries

		pruning									
		0%	10%	15%	20%	25%	30%	40%	50%	60%	65%
tdm1	MAP	0.2966	**0.3006**	**0.3062**	**0.3074**	0.2892	0.2704	0.2473	0.2151	0.2054	–
	P@10	0.4780	0.4780	0.4780	**0.4860**	**0.4660**	0.4460	0.3980	0.3440	0.3143	–
tdm2	MAP	0.2966	0.2985	0.2942	0.2925	0.2755	0.2741	0.2602	0.2436	0.2166	0.2054
	P@10	0.4780	0.4620	0.4600	0.4680	0.4360	0.4400	0.4080	0.3780	0.3583	0.3208
idf	MAP	0.2966	0.2987	0.2945	0.2928	0.2749	0.2733	0.2599	0.2410	0.2163	–
	P@10	0.4780	0.4640	0.4620	0.4700	0.4380	0.4380	0.4100	0.3760	0.3204	–
ridf	MAP	0.2966	0.3000	0.3050	0.2970	**0.2922**	**0.2962**	**0.2881**	**0.2625**	**0.2325**	**0.2322**
	P@10	0.4780	**0.4800**	**0.4880**	0.4640	0.4600	**0.4640**	**0.4560**	**0.4320**	**0.3760**	**0.3653**

		pruning									
		0%	9.3%	14.0 %	19.1%	24.2%	30.3%	39.4%	52.1%	58.1 %	66.0%
Carmel	MAP	0.2966	0.2789	0.2779	0.2712	0.2634	0.2591	0.2606	0.2405	0.2283	0.2188
	P@10	0.4780	0.4480	0.4460	0.4540	0.4440	0.4480	0.4400	0.4280	0.4220	0.4200

good at increasing precision, although at the cost of being too aggressive. The other two methods, *tdm2* and *idf* are very correlated and perform slightly worse than ridf for most of the cases.

Tables 5.2 and 5.3 summarise the results for the WT2G collection. Results are analogous to the ones obtained in the LATimes, although short queries benefit more from precision gains. It is remarkable that Carmel's method is able to improve P@10 values in the WT2G collection at very high pruning levels (short queries only).

Every method presented needs to set some threshold in order to *stop pruning*, be it the ϵ parameter (Carmel's method) or the percentage of pruning (term pruning methods). We carried out a third experiment in order to find an automatic threshold using Fox's stop-list as *relevance* information, i.e. good stop-words. The procedure is as follows: the list of terms is sorted according to a first measure and split into several intervals bounded by the *relevant* (trusted) stop-words. For every term and using a second measure, its informativeness value v_1 and the value of the lower bound of its corresponding interval v_2 are compared. If $v_2 \geq v_1$ the term is pruned. Combining the *ridf* (first) and *tdm2* (second) measures this approach gives, for long(short) queries, MAP values of 0.2685(0.2490) and P@10 values of 0.3044(0.2889) at a 56% pruning level in the LATimes collection. These precision values are obtained automatically and comparable with the ones obtained by Fox's stop-list alone, but at a higher pruning level.

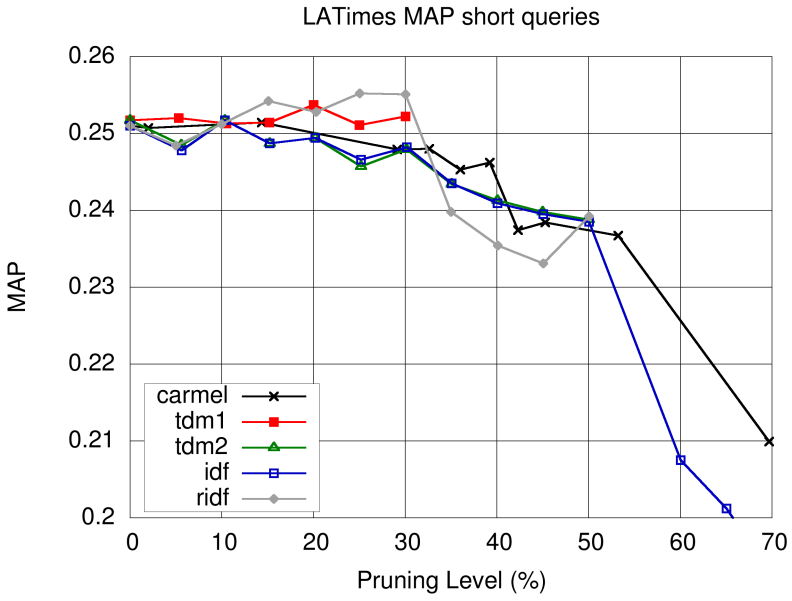


Figure 5.9: Term-based static pruning: MAP vs. %pruning, LATimes collection& short queries

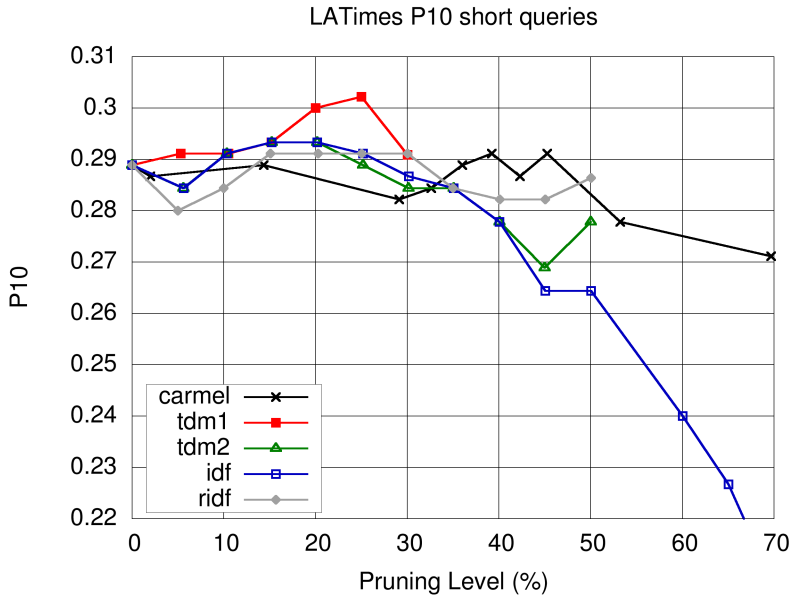


Figure 5.10: Term-based static pruning: P@10 vs. %pruning, LATimes collection & short queries

Table 5.3: Term-based static pruning: precision vs. %pruning, WT2G & short queries

		pruning									
		0%	10%	15%	20%	25%	30%	35%	40%	50%	55%
tdm1	MAP	0.2540	**0.2688**	**0.2719**	**0.2661**	0.2524	0.2470	–	–	–	–
	P@10	0.4180	**0.4540**	**0.4560**	**0.4620**	**0.4480**	**0.4271**	–	–	–	–
tdm2	MAP	0.2540	0.2635	0.2641	0.2600	0.2498	0.2490	0.2393	0.2351	0.2172	–
	P@10	0.4180	0.4360	0.4360	0.4300	0.4040	0.4060	0.3800	0.3620	0.3553	–
idf	MAP	0.2540	0.2635	0.2644	0.2602	0.2503	0.2495	0.2408	0.2351	0.2172	0.2109
	P@10	0.4180	0.4380	0.4380	0.4300	0.4080	0.4040	0.3780	0.3600	0.3480	0.3163
ridf	MAP	0.2540	0.2619	0.2640	0.2636	**0.2594**	**0.2572**	**0.2524**	**0.2509**	**0.2333**	**0.2254**
	P@10	0.4180	0.4400	0.4360	0.4204	0.4143	0.4204	**0.4020**	**0.3939**	**0.3633**	**0.3653**

		pruning									
		0%	9.27%	14.0%	19.1%	24.2%	31.0%	39.6%	45.2%	52.1%	58.9%
Carmel	MAP	0.2540	0.2634	0.2632	0.2622	0.2606	0.2558	0.2548	0.2526	0.2397	0.2301
	P@10	0.4180	0.4360	0.4360	0.4360	0.4380	0.4420	0.4400	0.4500	0.4580	0.4500

5.3.4.2 Index compression vs. index pruning

Figure 5.11 shows the real tradeoff between pruning level and disk space usage (WT2G collection). The graphs reflect how the inverted file size decreases when the number of pruned pointers increases using different coding methods. Sizes are relative with respect to the original inverted index except in the last graph, where the size is absolute. Only the posting list file is considered since the space reduction due to the lexicon file is not significant. The behaviour is stable for every compression algorithm, which proves that measuring the pruning level as the number of deleted occurrences is a valid indicator of the final compressed file, despite of the coding method used. The best reduction is obtained for the method based on *ridf* although with minor differences. The final figure shows the relative performance of the different coding algorithms, measured in megabytes (pruning values obtained with *ridf*).

It is possible to explain the values in figure 5.11 as follows. Real coding of posting lists is based on document gaps. A document gap is the difference between two consecutive document identifiers in the same list. For a given term with consecutive document identifiers a, b, c the cost of coding its postings would be $\phi(b-a)+\phi(c-b)$ and for bit-based coding methods $\phi(x) = O(\log(x))$. Carmel's method may prune the document occurrence with identifier b resulting in a coded posting list reduction from $\log(b-a)+\log(c-b)$ to $\log(c-a)$. Methods that prune every term occurrence do not leave this $\log(c-a)$ gap in the posting list when they operate, as they remove the whole list, thus

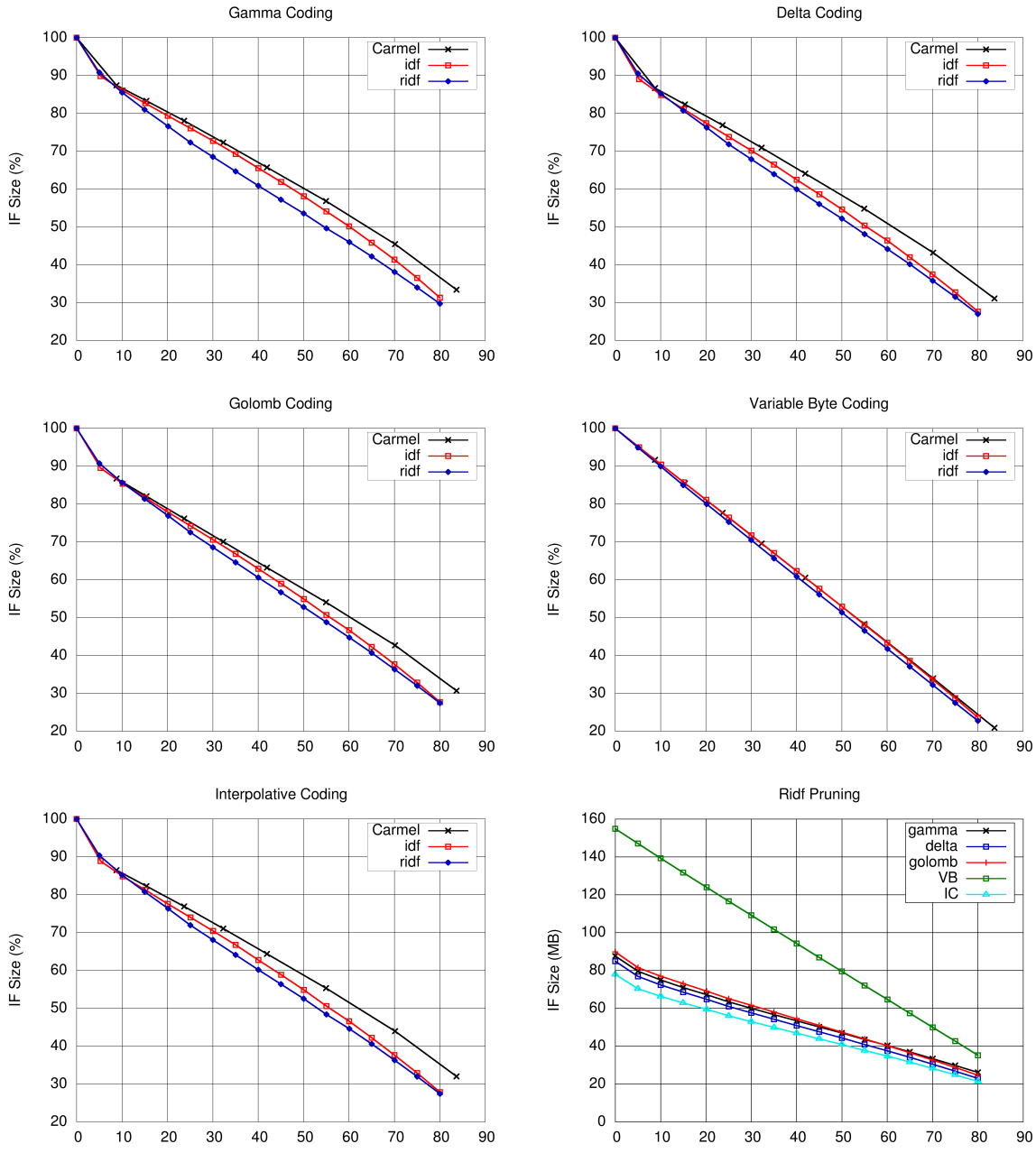


Figure 5.11: Effect of pruning on the final compressed inverted file size

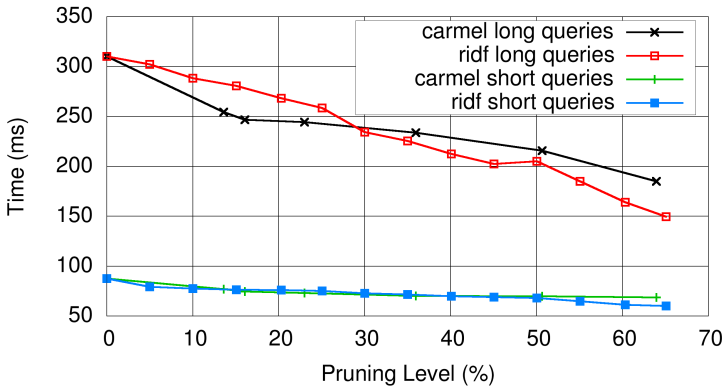


Figure 5.12: Average query processing time (ms) vs. %pruning

they may yield less average bits per gap values. The first slope in the graphs is due to the fact that the first terms being pruned are the ones with highest document frequency, which happen to be the ones with the highest within-document term frequencies. When those frequencies are coded in unary ($\phi(x) = x$) the space saved when they are removed is more noticeable. In fact, if the frequencies are coded with gamma, the slope softens. It is interesting to notice that Carmel's method follows this behaviour too, which indicates that if ϵ is low, it is only able to delete occurrences of terms with high document frequency.

In the case of variable byte coding, 90% of the pointers require just one byte and therefore there is no noticeable difference among the methods. Variable byte is clearly the worst method with respect to inverted file size, although it is interesting because of its faster decompression times.

5.3.4.3 Query times vs. pruning

Figure 5.12 reports on average query times for *ridf* and Carmel's method on the LATimes collection with fifty queries (topics from 401 to 450). There is a query processing time reduction which is more important in the case of long queries. The different behaviour between the methods is due to the number of disk accesses, main bottleneck for query evaluation in retrieval systems. Every query term is processed if the inverted file is pruned with Carmel's method, and this is the reason why query processing time varies smoothly with respect to the pruning level. In the *ridf*-based pruning method, query processing times can be drastically reduced at pruning levels that maintain or even improve the precision values.

5.3.5 Summary and future work

In this section, we implemented several pruning techniques based on the *informativeness* and *discriminative value* of terms. Next, we evaluated the behaviour of precision with respect to pruning, and the final effect in index file reduction and query processing times. Those methods have been compared with the well-known pruning method introduced by Carmel et al. [2001b]. As a general conclusion, *tdm1* is good if only high values of precision are desired, although it is very aggressive, and *ridf* is easy to implement and very stable. In general, pruning whole terms is better for maintaining or improving MAP, and it keeps precision values at high pruning levels with long queries, whereas pruning pointers is better with respect to P@10. In particular, Carmel's method behaved very well for P@10 and short queries in the WT2G collection. Therefore, methods that prune terms could be useful in applications such as indexing collections for PDAs and mobile devices, and desktop search.

One future research line is to design a pointer-based pruning method that operates selectively over posting lists, driven by a global term rank. Another topic of research is to address the problem of pruning while allowing for phrasal queries. None of the methods presented here is appropriate for processing phrasal queries. To tackle these problems it is necessary to develop an explicit pruning

method for this purpose (de Moura et al. [2005]) or to combine a pruned inverted file with a next-word index (Bahle et al. [2002]).

5.4 Probabilistic Static Pruning of Inverted Files

5.4.1 Derivation of Pruning Criterion from Probability Ranking Principle

In this section, we show how to formally derive a pruning mechanism based on the *Probability Ranking Principle* (PRP).

The *Probability Ranking Principle* (PRP) Robertson [1977], van Rijsbergen [1979] states: *If a [...] retrieval system's response to each request is a ranking of the documents in the collections in order of decreasing probability of relevance to the user who submitted the request, where the probabilities are estimated as accurately as possible on the basis of whatever data has been made available to the system for this purpose, then the overall effectiveness of the system to its users will be the best that is obtainable on the basis of that data.*

The PRP estimation can be used for the retrieval of documents, and that is the foundation of the probabilistic model of IR. The fundamental idea behind this work is to introduce this principle for index pruning and not for document retrieval.

Given a document and a query, represented by the random variables D and Q, a probabilistic IR model calculates the probability of D being relevant $p(r|D,Q)$ and non-relevant $p(\overline{r}|D,Q)$. It was shown in van Rijsbergen [1979] that the ranking produced by the odds-ratio of relevance and non-relevance is equivalent to the ranking produced by PRP.

PRP can be seen as a Bayes decision rule considering the classes of relevance r and not relevance $\overline{r}$. As such, the probability of a document being relevant (respectively non-relevant) to a query has an associated error (ξ). Formally,

$$p(\xi|D,Q) = \begin{cases} p(r|D,Q) & \text{if we decide } D \in \overline{r} \\ p(\overline{r}|D,Q) & \text{if we decide } D \in r, \end{cases} \tag{5.17}$$

Each type of error can be associated with a *cost*, so that our decision rule would minimise the *risk* of assigning a document to a wrong class. Consider the following definitions:

- Let c_{rr} be the cost of deciding $D \in r$ when $D \in r$
- Let c_{rn} be the cost of deciding $D \in r$ when $D \in \overline{r}$
- Let c_{nn} be the cost of deciding $D \in \overline{r}$ when $D \in \overline{r}$
- Let c_{nr} be the cost of deciding $D \in \overline{r}$ when $D \in r$

Under the risk-minimisation perspective, the risk of classifying non-relevant documents as relevant should be lower than the risk of classifying relevant documents as non-relevant:

$$c_{rr}p(r|Q,D) + c_{rn}p(\overline{r}|Q,D) < c_{nr}p(r|Q,D) + c_{nn}p(\overline{r}|Q,D) \tag{5.18}$$

Or equivalently,

$$\frac{p(r|Q,D)}{p(\overline{r}|Q,D)} > \frac{c_{rn} - c_{nn}}{c_{nr} - c_{rr}} \tag{5.19}$$

Equation 5.19 states that the ratio of the probability that a document is relevant to a query over the probability that a document is not relevant to a query should be greater than the cost ratio which acts as a threshold:

$$\epsilon = \frac{c_{rn} - c_{nn}}{c_{nr} - c_{rr}} \tag{5.20}$$

The basic idea in this section is to consider every term in the lexicon as a single-term query. We are not interested in the ranking produced by PRP for this query, but only in identifying which

document-query pairs satisfy equation 5.19. If a document-query pair satisfies equation 5.19 it is kept in the index, otherwise it is pruned. Therefore, equation 5.19 acts as our pruning criterion where ϵ (equation 5.20) acts as a pruning threshold. More simply, this is a way of deciding, for every term-document occurrence, if it would be acceptable to consider that entry as relevant, given a query. In essence, the odds-ratio states how characterising a term is in the context of a document.

The threshold ϵ can be set in several ways; some choices could be to estimate the collection difficulty or using relevance information. In this work, we set the basic costs to $c_{rn} = c_{nr} = 1$, and $c_{nn} = c_{rr} = 0$, and hence the starting threshold is 1. This is a typical assumption in retrieval scenarios. However, as it will be shown in the experimental section, we increase this threshold in order to assess the pruning versus precision trade-offs.

For the estimation of the left hand side of equation 5.19 we employ a decomposition of the odds-ratio of relevance and non-relevance (Lafferty and Zhai [2003]):Using Bayes rule, this becomes:

$$\begin{aligned}\frac{p(r|Q,D)}{p(\overline{r}|Q,D)} &= \frac{p(D,Q|r)\ p(r)}{p(D,Q|\overline{r})p(\overline{r})} \\ &= \frac{p(Q|D,r)\ p(D|r)\ p(r)}{p(Q|D,\overline{r})\ p(D|\overline{r})\ p(\overline{r})} \\ &= \frac{p(Q|D,r)\ p(r|D)}{p(Q|D,\overline{r})\ p(\overline{r}|D)}\end{aligned} \tag{5.21}$$

Equation 5.21 states that the ratio of the probability that a document is relevant to a query over the probability that a document is not relevant to a query can be decomposed into two odds: one document dependent and one document independent. Now assume that the document D is independent of the query Q under the conditioned event $\overline{r}$, $p(D,Q|\overline{r}) = p(D|\overline{r})\ p(Q|\overline{r})$. Therefore,

$$\frac{p(r|Q,D)}{p(\overline{r}|Q,D)} = \frac{p(Q|D,r)}{p(Q|\overline{r})} \cdot \frac{p(r|D)}{p(\overline{r}|D)} \tag{5.22}$$

or under a binary term-independence assumption Robertson et al. [1981] for a query of $|Q|$ terms:

$$\frac{p(r|Q,D)}{p(\overline{r}|Q,D)} = \prod_{i=1}^{|Q|} \left(\frac{p(q_i|D,r)}{p(q_i|\overline{r})} \right) \cdot \frac{p(r|D)}{1-p(r|D)} \tag{5.23}$$

Equation 5.23 decomposes the ratio of the probability that a document is relevant to a query over the probability that a document is not relevant to a query into three distinct probabilities. The first probability, $p(q_i|D,r)$, is the query likelihood. The second probability, $p(r|D)$, is the document prior. The third probability, $p(q_i|\overline{r})$, is the probability of a term given non-relevance. We show how we estimate each of these probabilities separately in Section 5.4.2.

In the formulation presented in equation 5.23, it is assumed that query terms occur independently of each other (*term independence assumption*). During the pruning stage we do not have any query information and it seems reasonable to follow the term independence assumption. This makes sense because typically, scoring functions are additive, which means that the partial score that a term produces has nothing to do with the contribution of the other, and thus to the final score.

Our proposed pruning method presented above is *uniform* because the threshold is the same for every term in the index. However, the probabilities to be estimated bias the amount of data to be pruned more towards some terms than others. This is a first procedural difference of this method from Carmel et al. [2001b]. The latter method decides the amount of pruning on the basis of the differences between the scores of a single term in different documents. A second difference is that the intention of this pruning mechanism is not to be faithful to the original ranking produced by a given scoring function, nor to make results imperceptible for a user, but to detect which terms are not significant in the context of a document, so those term occurrences can be discarded from the index. The pruning process is summarised in figure 5.13.

In the framework presented here, it is possible to incorporate additional information about which terms are more likely to be pruned, or equivalently, less *relevant*, by altering the estimation of the probabilities in use.

```
     Prune(ϵ)
1:   for every term q in the lexicon do
2:       {docs} = posting(q)
3:       p_q = p(q|r̄)
4:       ∀D ∈ {docs}
5:           p_d = p(r|D)
6:           p_r = p(q|D, r)
7:           s = (p_r/p_q) * (p_d/(1 − p_d))
8:           if s < ϵ
9:               remove D from {docs}
```

Figure 5.13: PRP-based uniform pruning algorithm. The procedure takes a parameter ϵ that acts as a threshold. It calculates three probabilities and combines them in a score. Finally, it discards the posting entries that score less than the threshold.

Finally, it is worth to state that this algorithm is well-founded, because it uses the PRP. If the conditions of PRP were to hold, indexes pruned using this technique would be optimal for retrieval tasks, (in the same way PRP is optimal for retrieval). In section 5.4.2, we present each probability appearing in equation 5.22, and show a feasible way of estimating them.

5.4.1.1 PRP and probabilistic retrieval models

The PRP Robertson [1977], van Rijsbergen [1979] is the foundation of probabilistic retrieval models, and it has been stated as a criterion of *optimum retrieval* since the 70s. BIR (Binary Independence Retrieval) is the probabilistic retrieval model that follows the term independence assumption and PRP Robertson and Sparck-Jones [1976b], van Rijsbergen [1977]. Robertson et al. [1981] developed a retrieval model based directly on the log-odds ratio and following PRP. The relevance of a document to a query was estimated using within-document term frequencies, modeled with a 2-Poisson mixture distribution. The parameters of the distributions were estimated directly by the within-document frequencies distributions, and its usage in weighting functions resulted in little performance benefits. Based on these same assumptions, Robertson and Walker [1994] developed a simpler model that incorporated some variables present in the 2-Poisson model in an ad-hoc fashion. This model further resulted in the successful BM25 retrieval model. The work in Manmatha et al. [2001] considers a probabilistic model where the main focus is to calculate the probability of relevance and non-relevance. The model employs the distribution of scores provided by a search engine in order to obtain the probability of relevance of a certain document. Assuming that different search engines provide independent rankings, those probabilities are combined. Probabilities and PRP allow for ranking fusion in a more principled way. Finally, an interesting connection between language and probabilistic models for IR using PRP was presented in Lafferty and Zhai [2003].

5.4.2 Probability Estimations

Generally, statistical methods rely on the goodness of their estimations. In this section, we present how we estimate each component of equation 5.22, namely

- the probability of a term given a document and relevance, or query likelihood (Section 5.4.2.1),
- the prior probability of relevance of a document (Section 5.4.2.2),
- the probability of a term given non-relevance (Section 5.4.2.3).

We present and discuss each estimation separately. The estimations proposed are not the *only ones* possible; however, they turned out to work very well for pruning purposes.

It is important that the estimates of the probabilities combine well with each other. Weighting schemes make explicit assumptions for assigning term weights like doing logarithmic transformations and smoothing probabilities in a particular way. Those assumptions do not affect the ranking process, because the final value of the score is taken to be meaningless in most cases: ranking only needs a final ordering of the documents. The scenario presented here implies a more complex modelling approach, as the probabilities to be estimated should combine well with each other (considering smoothed estimations).

5.4.2.1 Query likelihood

In equations 5.19, 5.20 and 5.22, we presented our pruning criterion, which consists of the combination of three probabilities and a threshold. The first of these probabilities is $p(Q|D,r)$, which is the probability of a document generating a given query. This probability corresponds to the well-known query likelihood probability, a fundamental part of the language modelling approach Zhai and Lafferty [2004].

The query likelihood probability refers to the probability of a document generating a given term q_i. This probability has to be smoothed for being of real use in retrieval. Some well known smoothing methods are Dirichlet prior and the linear interpolation scheme (or Jelinek-Mercer (JM) smoothing) Zhai and Lafferty [2004]. The performance of smoothing methods is dependent on some particular parameters. In this work we chose JM smoothing, because of its relative stability for short queries. Hence, for every term q_i, $p(Q|D,r)$ is approximated by $p(q_i|D)$, as follows:

$$p(q_i|D) = (1-\lambda)p_{mle}(q_i|D) + \lambda\, p(q_i|\mathcal{C}),\ \lambda \in [0,1] \tag{5.24}$$

where p_{mle} is the maximum likelihood estimator for a term q_i given a document D, $p_{mle}(q_i|d) = \frac{tf(q_i,d)}{|D|}$, tf is the frequency of a term in a document, $|D| = \sum_{w_i \in D} tf(w_i, D)$ (the document length) and λ is a parameter (we set λ to 0.6 in every collection, see section 5.4.3.1).

$p(q_i|\mathcal{C})$ is also the maximum likelihood estimator for a term q_i given a collection $\mathcal{C}$:

$$p(q_i|\mathcal{C}) = \frac{count(q_i,\mathcal{C})}{\sum_{q_j \in \mathcal{C}} count(q_j,\mathcal{C})}\ , \tag{5.25}$$

where $count(q_i,\mathcal{C})$ is the number of occurrences of q_i in collection $\mathcal{C}$.

In language models, the query likelihood component represents the probability of a document model generating a string of text (query): $p(Q|D)$. We approximate the probability of the document generating the query *given relevance* $p(Q|D,r)$ with a simpler probability estimate. A possible alternative way to tackle this issue are relevance-based language models Lavrenko and Croft [2001], which capture the notion of relevance in the language modelling framework. In practice, relevance LMs can be seen as a way of incorporating pseudo-relevance feedback to the language modelling approach in a principled way. A relevance language model would incorporate to the estimation information about terms related to the query (in our case, a single term). We did not follow this path in this work, due to the easiness of computing equation 2.20, and the high performance it brings on its own for pruning.

5.4.2.2 Document prior

The second of these probabilities to be estimated is $p(r|D)$, which is the prior probability of relevance of a document, without any query information. This corresponds to the likelihood that the document is relevant by just seen *query independent* evidence. A document prior can be based on certain knowledge on the document structure or content, and this is currently an active area of research. For instance, in Web IR, the typical sources for document priors information come from the link structure of Web pages Kraaij et al. [2002], such as the number of incoming links (in-links) and URL depth Westerveld et al. [2002], or even the Pagerank Brin and Page [1998] algorithm for ranking Web documents according to their popularity Upstill et al. [2003].

In most cases $p(r|D)$ is taken to be uniform Zhai and Lafferty [2004], therefore it has no effect on retrieval. However, there have been several studies where the document length and link structure have been encoded as a prior probability, for ad-hoc and some non ad hoc tasks Kraaij et al. [2002], Westerveld et al. [2002]. Overall, incorporating prior knowledge on documents into retrieval has been particularly effective on Web retrieval, namely *homepage* and *named page finding*, which refer to the retrieval of a single Web page.

Using the formulation presented here, even a *uniform* document prior would have effect on pruning. The only way of avoiding the prior effect over the pruning algorithm is to set $p(r|D) = p(\overline{r}|D)$ so that the document prior component cancels out.

We further explored the issue of estimating this probability, and developed a way for estimating the prior that worked well for every collection tested in the pruning framework. In this work $p(r|D)$ is estimated solely as a function of document length and without any linkage structure information, therefore our findings can also be applied to collections without link information.

The derivation of the length-based document prior is as follows. A previous approach for deriving a prior based on document length, considered $p(r|D)$ as a linear function on the number of tokens of a document, as it is supposedly reminiscent of the *true* shape of relevance Singhal et al. [1996]. The work by Kraaij et al. [2002] models the prior probability and document length dependency as a straight line, which tries to reflect this fact. The idea behind this prior is that longer documents are more likely to embody more topics, and hence should receive a higher prior probability. This agrees with the scope hypothesis Robertson and Walker [1994] that states that longer documents cover more topics than shorter ones. On the contrary, the verbosity hypothesis states that longer documents cover the same number of topics than shorter ones, but they just use more words to do it. Some effective retrieval models (like BM25 for instance) are also based on a parameterised combination of both hypothesis. The detailed analysis of the true shape of relevance by Singhal et al. [1996] shows that a sigmoid (pseudo-linear) function fits better the relevance curve. This also allows for incorporating more elements into the modelling, as shown next.

The hyperbolic tangent function is one of the hyperbolic transcendental functions and a member of the sigmoid family. It has some nice properties we shall exploit, like

- $tanh(0) = 0$
- Its inflexion point is $x = 0$.
- $tanh(x) \in [-1, 1]$

We employ linear transformations (in the form of $ax + b$) to adjust the hyperbolic tangent:

$$S(x) = a + b \cdot tanh\left(c \cdot (x - d)\right) \tag{5.26}$$

It is possible to model the behaviour of the sigmoid function $S(x)$ using four parameters, a, b, c and d. In particular, the *slope* is controlled with c, d is the inflexion point, and its bounding interval is determined by a and b.

These parameters should be adjusted in order to transform equation 5.26 depending on some particular assumptions, with the goal to assign a prior probability -$S(x)$- to every document based on its length -x-. A different set of assumptions/intuition on how the document prior probability should behave may lead to interpretations different than the one presented next.

First of all, we take the centre point to be the average document length $\overline{X_d}$. This decision goes accordingly to the study by Singhal et al. [1996], where the relation between document length and relevance followed a sigmoid function centred over the mean document length. The shape of the curve implies there are some bounding values x_1, x_2 in the x-axis ($x_1 < \overline{X_d}, x_2 > \overline{X_d}$) that decide for which document lengths the normalisation is quasi-linear and for which it is quasi-uniform. This would be the same as considering $if\ x < x_1 \Rightarrow S(x) \approx S(x_1)$ and $if\ x > x_2 \Rightarrow S(x) \approx S(x_2)$.

Now, let $S(x_1) = lo$ and $S(x_2) = hi$, and let μ be the point where we consider the tangent reaches its maximum, $tanh(\mu) \approx 1$ (we take $\mu \approx 2$), then the curve family results in:

$$p(r|D) \approx S(dl) = \frac{hi - lo}{2} \cdot tanh\left(\frac{\mu \cdot (2dl - (x_1 + x_2))}{x_2 - x_1}\right) + \frac{hi + lo}{2} \tag{5.27}$$

In particular, the sigmoid function is useful for bounding both the effect of the prior on the pruning score and the range of document lengths the quasilinear effect is applied to. In the following paragraphs we model the dependency of $p(r|D)$ by just using document length and with the shape of a sigmoid function (figure 5.14).

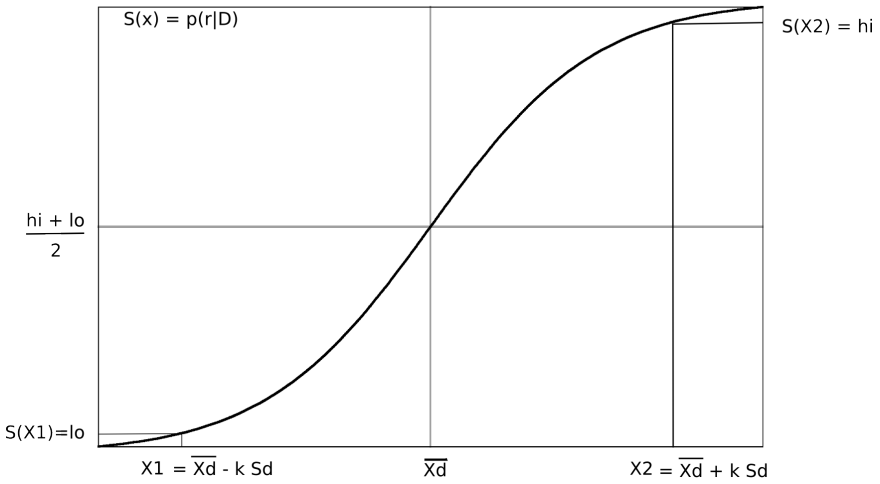


Figure 5.14: $p(r|D)$ estimation using document lengths and a sigmoid function. $\overline{X_d}$ is the average document length, S_d the typical deviation over the document lengths, and hi, lo and k are set to constant values

First, we set the bounds $[lo, hi]$ for the contribution of the document prior probability to the right-hand side of equation 5.23: $p(r|D)/(1 - p(r|D))$. Then, we discuss the document length range that the linear effect will be applied to.

Figure 5.15 is a plot of the $x/(1 - x)$ function. In our case, this function reflects the contribution of the document prior to the pruning score (y-axis), given a prior probability $p(r|D)$ (x-axis). Moving to the right side of the x-axis implies a greater contribution if that value is accepted as a probability estimation. After a certain point, the growth of the function is exponential. Moving to the left side of the x-axis results in a very low contribution to the pruning score (equation 5.23), therefore its effect on discriminating between documents diminishes. In other words, this plot reveals some interesting facts about the bounds of the prior: priors on the right of the x-axis will affect more the pruning score, and priors on the left side will result in a uniform effect for all the documents, regardless of their length. For these reasons, a reasonable interval for the prior (in this particular application) could be around 0.5, as this is the point where the prior contribution is 1: a prior value higher than 0.5 boosts the query-independent score, whilst it is decreased by values lower than 0.5. In this work we narrow the prior to the $[0.4, 0.6]$ interval, taking a conservative approach.

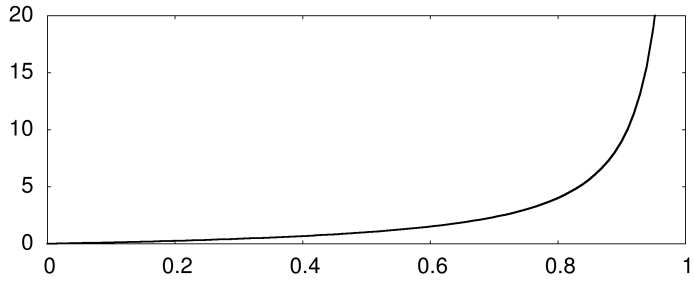


Figure 5.15: $f(x) = x/(1 - x)$

Now that those bounds are set, we determine the document length range that produces a quasi-linear behaviour of $p(r|D)$ in the $[lo, hi]$ ($[0.4, 0.6]$) range, and asymptotical outside that interval. We relate that range to the standard deviation of the document lengths over a whole collection. More precisely, equation 5.28 determines which fraction of the documents are going to be affected linearly, ruled by parameter k. Chebyshev's inequality states that *in any data sample or probability distribution, nearly all the values are close to the mean value*, and provides a *quantitative description of"nearly all" and "close to" in terms of the typical deviation*. More formally, let X be a random variable with mean $\overline{X}$ and variance S^2, then

$$P\left(|X - \overline{X}| > k \cdot S\right) \leq \frac{1}{k^2} \tag{5.28}$$

Figure 5.14 is the sigmoid function, as presented above: it grows linearly between a certain range, and it is possible to determine bounds for both axes. In the figure, hi and lo stand respectively for the maximum and minimum values of the prior estimation $p(r|D)$: 0.6 and 0.4, decided after the illustration of its effect on the pruning score, showed in figure 5.15. The document length range affected is set by an interval around the average document length $\overline{X_d}$, with the help of equation 5.28. The size of the interval is $2k$ times the typical deviation S_d over the document lengths: $[\overline{X_d} - kS_d, \overline{X_d} + kS_d]$. Thus, $x_1 = \overline{X_d} - kS_d$, $x_2 = \overline{X_d} + kS_d$ and k is set empirically to 2.

Substituting those values in equation 5.27, the final equation governing the S-shaped document prior depends on the document length dl in the following way:

$$p(r|D) \approx S(dl) = \frac{1}{2} + tanh\left(\frac{dl - \overline{X_d}}{S_d}\right) \cdot \frac{1}{10} \tag{5.29}$$

It is worth pointing out that this prior can be tweaked in many different ways, and some parameters can be tuned for specific collections, and thus it is possible to increase its overall performance. However, in all the experimental results presented in section 5.4.3 the prior has been calculated with equation 5.29, which is only dependent on collection statistics, for every collection tested.

5.4.2.3 Probability of a term given non-relevance

The last component needed to estimate in equation 5.22 is the probability of seeing a term given *non-relevance* $p(q_i|\overline{r})$. This is the only document-independent probability of our technique. Even though the pruning algorithm described sets the same pruning threshold for every term, this probability biases the amount of pruning towards some terms more than others. Recall equation 5.23: document-dependent probabilities combined should score less than $\epsilon \cdot p(q_i|\overline{r})$ to be pruned. In this particular case, this probability determines how likely the term is to be pruned, when compared to document-dependent factors (query likelihood and document prior).

Traditional estimations of this probability Croft and Harper [1979] consider the whole document set as non-relevant to a query term, and so the probability of a term being non-relevant would be:

$$p(q_i|\overline{r}) = \frac{df(q_i)}{N}\ , \tag{5.30}$$

where $df(q_i)$ is the number of documents q_i appears in, and N the total number of documents in the collection. This is the well-known inverse document frequency (idf) Robertson and Sparck-Jones [1976b] formulation which is an integral part of many retrieval models. Equation 5.30 turned out to be an overestimation for $p(q_i|\overline{r})$, and of little use in the formulation derived in this work. The estimation is too aggressive: it is higher than the two document-dependent probabilities combined, if they are estimated as presented in previous sections. The consequence is that even with a threshold ϵ of 1, the majority of the inverted file would be pruned.

Instead of deriving an alternative formulation for idf (like smoothing it, for instance), we follow a different path and consider the collection as a model of *non-relevance*. In this case, the probability of a term being non-relevant is modelled according to the probability of the collection language model of generating the term. The collection language model is the maximum likelihood estimator (MLE) of the probability of seeing a term in a collection. It is worth to point out that the role of this

probability for LMs is considered equivalent to the role of idf for classical retrieval models Zhai and Lafferty [2004]; if it is calculated using equation 5.25, where the event space is the total number of occurrences of a term in the collection, the probability of non-relevance of a term would be to consider every occurrence in the corpus as non-relevant (just like idf considers every document as non-relevant). We refine this probability next.

$$p(q_i|\overline{r}) = p(q_i|\mathcal{C}) \tag{5.31}$$

If idf is seen as a probability, it refers to the random event of selecting a single document that contains the term q_i (the total event space is determined by the documents). On the other hand, for $p(q_i|\mathcal{C})$, the event is to select a random term occurrence that happens to be q_i (the total event space is determined by the total term occurrences).

The above estimation turns out to be useful in the framework presented here, and gives good practical results. This good behaviour is likely due to the query likelihood component already incorporating the collection MLE $p(q_i|\mathcal{C})$ into its model.

However, it is still appealing to relate the non-relevance probability with document frequency. In particular, the probability should increase monotonically with respect to document frequency, like in the case of idf. How we incorporated this property into the estimation is presented next.

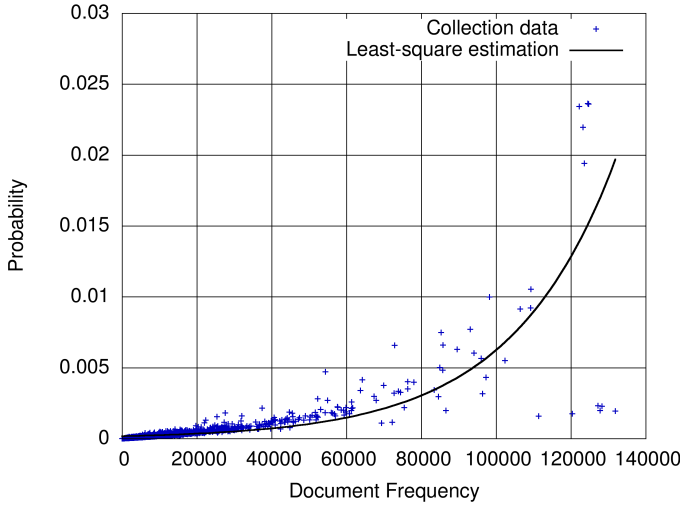


Figure 5.16: $p(q_i|\mathcal{C})$ and estimated exponential fit (least-square), LATimes collection

The points in figure 5.16 represent the individual $p(q_i|\mathcal{C})$ values for every term in the LATimes collection. The probabilities (y-axis) are plotted against the term document frequency (x-axis). Data reveals that the probability grows exponentially after a certain document frequency level.

In order to correlate $p(q_i|\mathcal{C})$ with idf, we take the straight approximation of equation 5.31 and fit the data to an exponential function with the form:

$$p(q_i|\overline{r}) = a \cdot e^{b \cdot x} \tag{5.32}$$

In this case x corresponds to the document frequency of term q_i. We employ the well-known least squares technique in order to fit the data. It is the simplest form of regression, which provides a solution to the problem of how to find the best curve matching a given set of points, in our case $p(q_i|\mathcal{C})$. The procedure tries to minimise the sum of the squares of the residuals of the points of the curve, adjusting a and b iteratively.

For providing an estimate for the first iteration, we opted to calculate the initial values for the a and b parameters in equation 5.32 by forcing the curve to pass through two points:

- the maximum value in the y-axis

Table 5.4: Collections and topics employed in the PRP-based pruning experiments

Collection	Size	Terms	Documents	Topics	Task
LATimes	450M	189,790	131,896	401-450	ad-hoc
WT2G	2G	1,002,586	247,491	401-450	ad-hoc
Disks 4&5	1.9G	521,469	528,155	301-450 + 601-700	ad-hoc
WT10G	10G	3,140,837	1,692,096	451-550	ad-hoc
.GOV	18.1G	2,788,457	1,247,753	1-225	NP & HP + TD

- (x_{av}, y_{av}), where x_{av} is the average value on the x-axis, and y_{av} is the average y-value on the $[x_{av} - 10000, x_{av} + 10000]$ interval. The choice of the width of this interval does not affect much the final parameter estimation and consequently, the shape of the exponential curve.

The least squares algorithm finds the best-fitting curve to a given set of points by minimising the sum of the squares of the residuals (SSR) of the point from the curve. At each step (iteration) the procedure calculates the SSR, and then the parameter values for the next iteration. The process continues until a stopping criterion is met, either the relative change in SSR is less than a convergence threshold (10^{-5} in this case), or it reaches a maximum number of iterations (100).

This fit is accurate and adequate for terms with low document frequency, but it is not completely adequate for some of the high document frequency terms, which appear at the rightmost extreme of figure 5.16. The points below the solid line representing the fit are a few high-frequency terms which only occur a few times in each document (tag words like *column* or *page* in a journalistic-domain collection or *http* on a web collection), and the fit behaves well boosting its probability of non-relevance. On the other hand, the fit is not adequate for terms which appear several times in many documents. In IR such terms are called 'stopwords' and are usually excluded from most computations, because they are of little significance to the final ranking of an IR system. Blanco and Barreiro [2007b] present an aggressive variation of Carmel's method that prunes every term with document frequency $df > N/2$. This modification is simple and worked well in some test collections (retrieval-wise). In this work we follow a conservative approach and maintain the *fit* for terms with $df < N/2$ and rule out from the index the rest of them, which is equivalent to assigning them automatically a high probability of non relevance.

In this section we proposed one of several possible estimations for $p(q_i|\overline{r})$. It is possible to estimate this probability in other ways that use term specific information which is also document independent. In section 5.4.4 we assess the effect of removing stopwords for different pruning techniques.

5.4.3 Experiments and results

This section describes the experimental evaluation of the proposed probabilistic pruning method. The objective is to assess the trade-off between pruning level and retrieval precision. For detailed effects on the retrieval efficiency of pruning algorithms see Büttcher and Clarke [2006], Carmel et al. [2001b].

This section is structured as follows: first we describe the experimental settings. Then, we define a baseline starting from Carmel's method and following some design considerations pointed out by Blanco and Barreiro [2007b], and detailed in section 5.2. Next, we compare the baseline and the proposed pruning model with three different retrieval models: TF-IDF (the model originally employed by Carmel et al. [2001b]), BM25 with standard and optimised settings (to avoid bias depending on parameter tuning) for every collection, and finally a parameter-free model based on divergence from randomness: DLHH (Amati [2006])

5.4.3.1 Experimental settings

Retrieval performance has been assessed on five TREC different collections, which exhibit different characteristics. LATimes and TREC Disks 4&5 contain documents of single sources of news-press media, assumed to be fairly homogeneous. On the contrary, WT2G, WT10G and .GOV contain

crawls of Web pages, which come from an heterogeneous source (the World Wide Web). Overall, the collections vary in content, size and statistics, so they form a wide-ranged testbed for the pruning techniques.
Evaluation has been done in two different ways:

- Standard Ad-Hoc retrieval with short and long queries.
- Mixed Web track 2004 topics: topic distillation and homepage and namepage finding tasks.

Both types of experiments imply well-known TREC-style evaluation. This means that every collection has an associated set of queries and a human-assessed set of relevance judgements, relating which documents are relevant to those queries; this set of judgements is not complete across the whole set of documents. Standard TREC queries are formed of three types of fields, each describing the topic in more details, namely *title*, *description* and *narrative*. In this work we experiment with two types of queries for the ad-hoc task: *short* (title only) and *long* (title and description). The Web track 2004 is formed of 225 short queries with different purposes (75 each) (Craswell and Hawking [2004]): homepage finding (the query is the name of a homepage website the user wants to reach), named page finding (the query is the name of a non-homepage website the user wants to reach) and topic distillation (queries describe general topics).

Retrieval performance is measured with Mean Average Precision (MAP) and Precision at 10 (P@10) (see section 2.6). The experimentation is designed in order to assess the effect of the pruning in retrieval. Results are reported for MAP and P@10 with respect to the percentage of postings kept in the inverted file (pruning level). Measuring the percentage of posting entries removed from the index turns out to be a good indicator of both final disk space savings in a compressed index and query performance gains (Blanco and Barreiro [2007a], section 5.2).

The evaluation is as follows. Given a collection, a judged query set and a retrieval model, first we evaluate the original unpruned index using the queries and relevance judgements. Then, we produce two sets of pruned indexes, one using Carmel's method (by varying the ϵ parameter) and the other with PRP-based pruning (varying the threshold). Every index in those sets is evaluated using the same conditions as the original unpruned index. we employ three different retrieval models for the query-document ranking, two probabilistic and one vector-space based, which are presented next. These models have been described previously in section 2.5.

Matching functions usually *normalise* the document length contribution to the final score by a factor that can be controlled by a parameter. Other models are built over some assumption that permit to surpass the parameter dependency, and so they are parameter-free, for instance some models found in the DFR framework (Amati and van Rijsbergen [2002], see also section 2.5.1.3).

The first matching function considered, equation 2.5, is a normalised TF-IDF variant as implemented in the SMART system (Buckley et al. [1995]), as described in section 2.5.1.2, equation 2.5.

This matching function is based on the vector-space retrieval model by Salton et al. [1975a]. The document length normalisation factor $\sqrt{(1 - slope) \cdot avgdl + slope \cdot dl}$ normalises document length so that there is no bias for longer documents. This document normalisation factor is knows as *pivoted normalisation* (Singhal et al. [1996]), and the default value for $slope$ is 0.2 (Buckley et al. [1995]). The first batch of experiments were carried out with this matching function because it was the one employed in the first evaluation by Carmel et al. [2001b] as a core part of the JURU search engine (Carmel et al. [2001a]).

The second matching function is the probabilistic Okapi's Best Match25 (BM25) Robertson et al. [1995], equation 2.10. BM25 includes three parameters, namely k_1, k_3, and b. Some studies (Chowdhury et al. [2002], He and Ounis [2003]), have shown that both k_1 and k_3 have little impact on retrieval performance compared to the b parameter. We set k_1 and k_3 to the constant values recommended by Robertson et al. [1995] ($k_1 = 1.2$, $k_3 = 1000$). The b parameter controls the document length normalisation factor. The experiments have been carried out using two different settings:

- Using the recommended value for $b = 0.75$ (Robertson et al. [1995]).

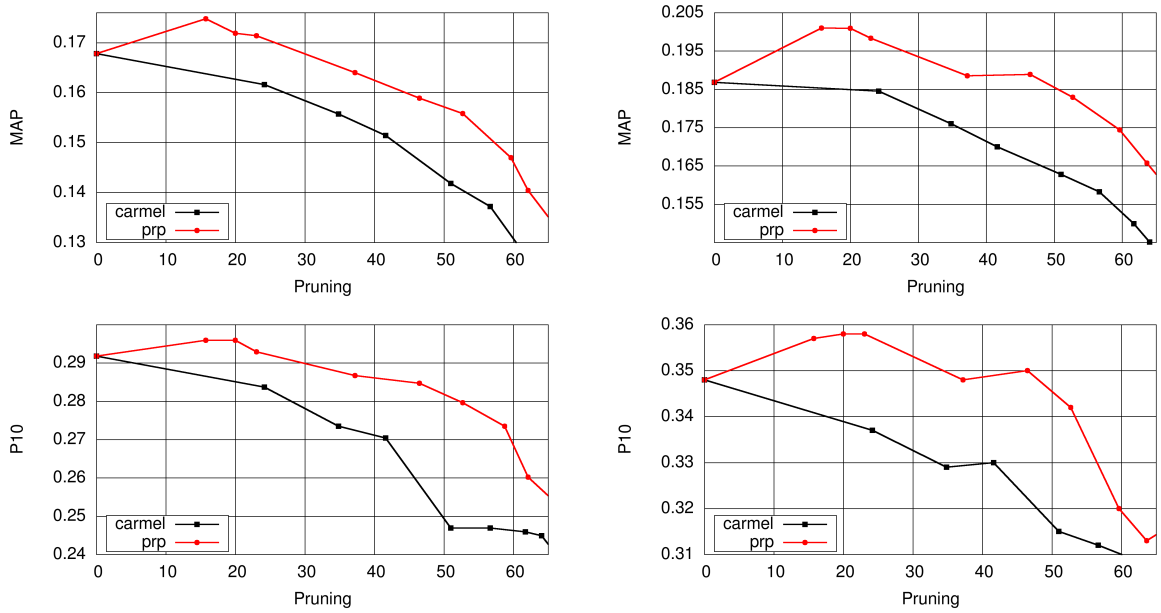


Figure 5.17: PRP vs Carmel pruning, short (left) and long (right) queries, WT10G, TF-IDF

- Optimising the value for b with 1D exploration ($b \in [0,1]$, $b = 0.1, 0.2...$). For every pruning level, we optimise b for MAP and P@10, choosing the values of b that maximise both measures.

DLHH (Amati [2006]) is a recent probabilistic-based retrieval model based on measuring the divergence between a random draw of occurrences of terms in documents and the actual distribution, equation 2.12. It is based upon a hyper-geometrical model, which embodies two probabilities: the relative within-document term frequency and the entire collection term frequency. This model is useful because it is *parameter-free*, and therefore tuning is not necessary for obtaining high retrieval performance. The goal is to assess whether retrieval performance can benefit from pruning or not, i.e., if discarding information from an inverted file can result in increased retrieval performance. Parameter-free models reduce their number of variables and factors to take into account for such a validation, and thus their convenience for this particular purpose.

The parameter settings for the estimations of the proposed pruning algorithm described in section 5.4.1 are set to the following values:

- to calculate the query-likelihood component $p(q_i|D,r)$, section 5.4.2.1, λ in equation 5.24 was set to 0.6. Preliminary experiments on some test collections demonstrated that setting this value performs well, although better performance can be obtained if it is tuned for a given dataset. However, we skip any further tuning of λ in order to prove that the technique is robust.

- the document prior component $p(r|D)$, section 5.4.2.2, is estimated by document length using equation 5.29.

- the probability of a term given non-relevance $p(q_i|\overline{r})$ is calculated by equation 5.31 and interpolated with the algorithm described in section 5.4.2.3.

It is possible to obtain better results by tuning those parameters dependently for each collection, but we omit those experiments to assess the behaviour of the method using a default setup. The cost-associated threshold for the probabilistic-based pruning was initially set to 1. In the following figures the first point corresponds to the pruning level obtained with $\epsilon = 1$. The subsequent pruning levels were obtained by increasing that value.

Regarding the settings for Carmel's method, k was set to 10, as it is beneficial for P@10 (Blanco and Barreiro [2007b], section 5.2), and the subsequent pruning levels were obtained by modifying ϵ.

Next, we present a set of six different experiments. The first (section 5.2.1.1) defines the baseline from a minor variant of Carmel's method, that considers the update of the statistics in the pruned index. In section 5.4.3.2, we repeat the experiments presented in Carmel et al. [2001b] by using pivoted TF-IDF. In sections 5.4.3.3 and 5.4.3.4, we focus on a high-performing parameterised model (BM25) without and with parameter tuning. In section 5.4.3.5 we repeat the experiments using a parameter-free model (DLHH). Section 5.4.3.6 summarises the results obtained by setting the threshold ϵ to a default value (1). Finally, section 5.4.3.7 tests our proposed pruning method on a bigger collection (.GOV), with a competitive model (BM25) on optimised settings, and using non ad-hoc queries. The aim is to see the effect of our technique on a different environment that ad-hoc retrieval.

In the following sections, we present the results for the WT10G collection using MAP, in several figures. In this chapter, we provide figures for P@10 for pivoted TF-IDF results only, to keep the number of graphs low and because when a set of relevance judgements has already been created MAP is a more reliable effectiveness measure than P@10 Sanderson and Zobel [2005]. In any case, those results with P@10 are reported to assess the good behaviour of the pruning techniques with respect to that metric.

Due to the high number of experiments and results, we present in an appendix the results for every collection listed in table 5.4 to avoid further graph overload. Those results confirm that the robustness of PRP-based pruning technique independently of the dataset employed.

5.4.3.2 PRP-based pruning and pivoted tf-idf retrieval

The first batch of experiments mimicked the settings of the experiments reported in the first static pruning paper by Carmel et al. [2001b]. This implies using pivoted TF-IDF (equation 2.5) for retrieval, with $slope$ set to 0.2. The pruning scores for Carmel's method were derived from the pivoted TF-IDF as well (as implied from the algorithm). As a consequence of the results presented in section 5.2.1.1 the implementation considered in this section updates the document lengths and the rest of the collection statistics, therefore implying a higher baseline for the comparison.

These results are presented in figure 5.17. The PRP-pruning with this particular retrieval model outperforms the baseline for MAP and P@10 at all levels and for both query lengths, in the WT10G collection. Overall, the original precision values can be retained up to a 40-50% pruning level, and at earlier levels PRP-based pruning is able to improve them.

5.4.3.3 PRP-based pruning and BM25 retrieval (recommended settings)

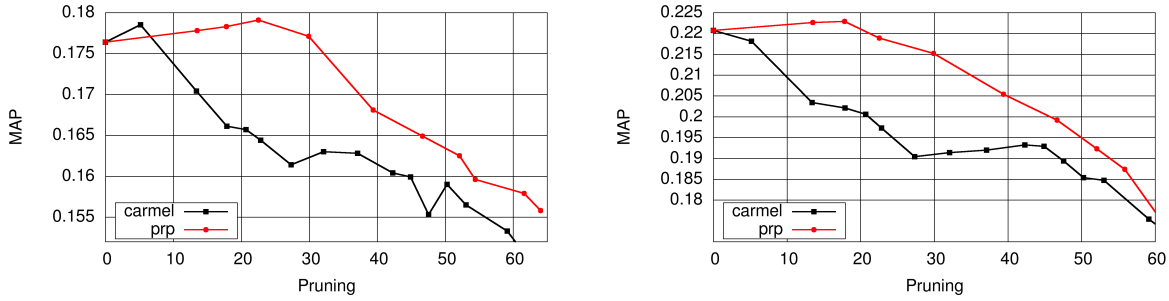


Figure 5.18: PRP vs Carmel pruning. BM25 retrieval with $b = 0.75$, short (left) and long (right) queries, WT10G.

Figure 5.18 presents the result in the WT10G collection. Results are very consistent across collections. The PRP-based pruning performs as well as, or outperforms (in most cases the latter) the baseline. Overall, original precision can be increased for every collection, and the method behaves well for both short and long queries.

5.4.3.4 PRP-based pruning and BM25 retrieval (optimised settings)

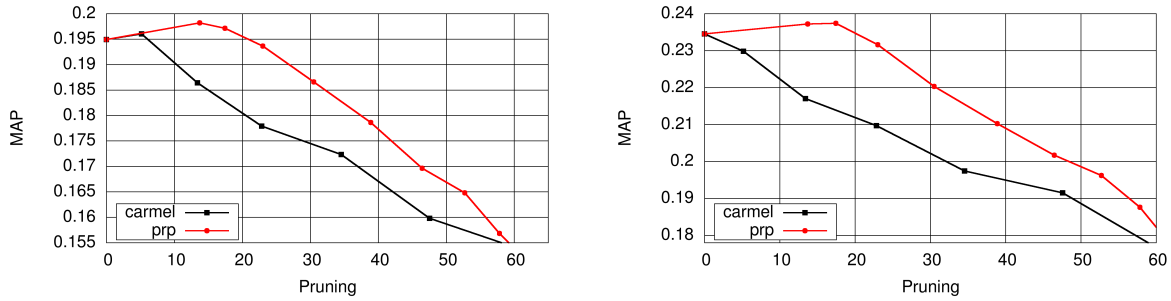


Figure 5.19: PRP vs Carmel pruning. BM25 retrieval with the best b, short (left) and long (right) queries, WT10G.

In order to rule out any possible bias introduced by the document length normalisation effect, this batch of experiments optimises the value of the b parameter in the BM25 formula, for every pruned index. This means that the MAP and P@10 results obtained are the best that the BM25 scoring function can achieve in every pruning level (for both the baseline and PRP-based pruning method). The b value used for retrieval is optimised for both MAP and P@10. The b value Carmel's method uses for pruning was set to 0.75.

Results are shown in figure 5.19. Again, the probabilistic pruning outperforms the baseline in terms of MAP, and this effect is most likely to be independent of the normalisation effect. Results are consistent for MAP (in most cases PRP-based pruning works better) across collections. With respect to P@10 the new technique works clearly better for the WT10G collection; in other collections the behaviour is not so clear: in disks 4&5 (Figure 7.11) the PRP-based method outperforms the baseline for short queries and long queries at early pruning levels($< 40\%$). In the WT2G collection (Figure 7.10) the new technique works better than the baseline for short queries and worse for long queries. In the LATimes collection (figure 7.9) it works better with short queries and high pruning levels.

5.4.3.5 Pruning using a parameter-free retrieval model

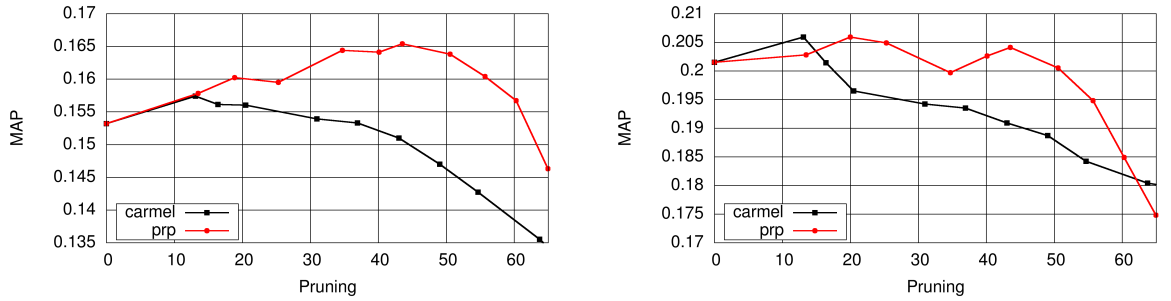


Figure 5.20: PRP vs Carmel pruning. DLHH retrieval, short (left) and long (right) queries, WT10G.

In the previous experiments we showed that the new pruning algorithm performs well with both TF-IDF and BM25, that these results are robust across different collections and that the performance of the new pruning technique is independent of a particular choice of parameters. In order to furthermore assess this last property, we consider a retrieval model that is completely parameter-free. BM25 and pivoted TF-IDF have parameters controlling the amount of normalisation assigned to long documents. Even when we tuned those parameters for BM25 in section 5.4.3.4, it is not totally clear whether pruning is discarding irrelevant (or noisy) data from the inverted file, or it is altering the collection statistics in a way that they might be more suitable for a particular matching function. As

Table 5.5: MAP and P@10 values obtained by setting the threshold ϵ to 1, compared to those of an unpruned index. The pruning level is $\approx 14\%$, WT10G collection

	WT10G							
	Short queries				Long queries			
	Original		Pruned		Original		Pruned	
	MAP	P@10	MAP	P@10	MAP	P@10	MAP	P@10
TF-IDF	0.1678	0.2918	0.1744	0.2939	0.1868	0.3480	0.1992	0.3560
BM25 $b = 0.75$	0.1764	0.2847	0.1779	0.2888	0.2207	0.3560	0.2231	0.3650
BM25 best b	0.1949	0.3112	0.1982	0.3122	0.2345	0.3650	0.2372	0.3720
DLHH	0.1532	0.2561	0.1581	0.2663	0.2015	0.3460	0.2032	0.3530

well, it would be interesting to see how the pruning behaves when using a third matching function, as the new pruning algorithm is score-independent (unlike the baseline). We employed DLHH (Amati [2006]), which is a parameter-free matching function derived from the DFR framework (Amati and van Rijsbergen [2002]).

Figure 5.20 presents the last batch of experiments. Results report that the parameter-free model benefits from pruning in every collection and with long and short queries. Hence, this experiment gives empirical support for the claim that pruning can bring beneficial parameter independent effects in retrieval. It is particularly important to remark that the probabilistic pruning algorithm is able to improve the retrieval performance, even at high pruning levels, fact that also occurred in many of the previous experiments.

5.4.3.6 Pruning with a default threshold

Table 5.5 summarises the values obtained with the threshold set to $\epsilon = 1$, as recommended in section 5.4.1, in the WT10G collection. In this case, the pruning level resulted in $\approx 14\%$, and the indexes produced produce gains in retrieval performance, despite of the retrieval method employed, for both MAP and P@10. This supports the claim that controlled pruning can be beneficial for both efficiency and effectiveness of IR systems.

5.4.3.7 Experiments with the .GOV collection

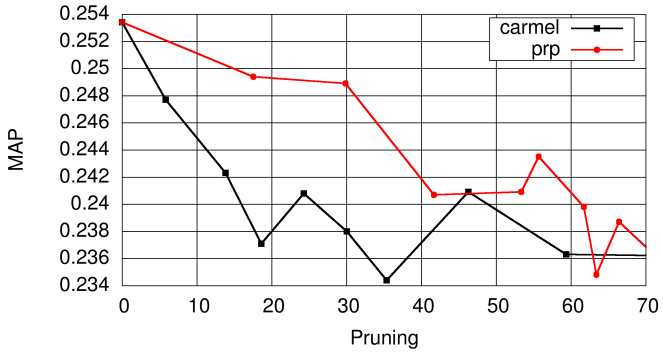


Figure 5.21: .GOV collection, namepage-homepage finding + topic distillation. BM25 retrieval with the best b.

In this batch of experiments, we ran both pruning algorithms on the .GOV, a larger web collection (actually WT10G has more documents, although they are shorter). The reason for these experiments is that we are performing ad-hoc retrieval with queries of a different nature, namely page finding and topic distillation. We did not employ any kind of query detection mechanism or link information to obtain the scores, just a raw BM25 (with the b parameter tuned like in section 5.4.3.4). The topic

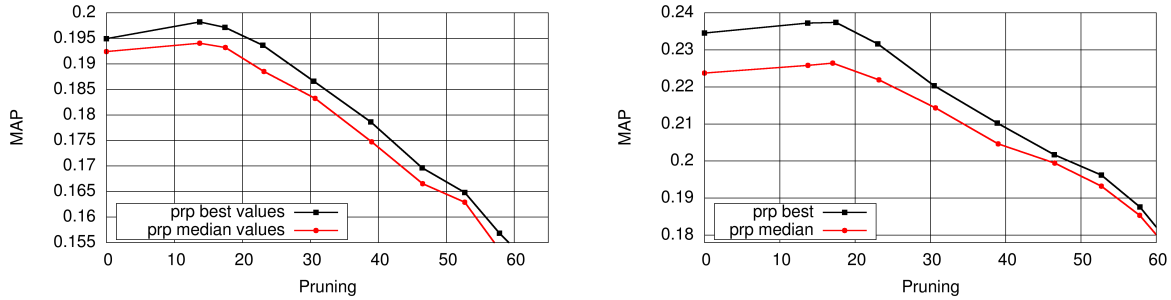


Figure 5.22: BM25 retrieval median and optimal performance for all the b values employed in section 5.4.3.4, short (left) and long (right) queries), WT10G collection

set employed was the one developed for the TREC Web track 2004. It contains a mix between three types of queries: home page finding, named page finding and topic distillation.

Figure 5.21 presents the results for both pruning regimes. The probabilistic pruning outperforms the baseline for most of the pruning levels, although this time the original MAP value could not be improved.

5.4.4 Discussion

The final formulation for the PRP-based pruning presented in this paper (equation 5.23) could also be seen as a term weighting function derived in a probabilistic fashion, much in the same way as language models. Given so, PRP pruning can be stated as a uniform pruning method, using the same cut-off value for pruning every posting list. However, the benefits of this particular formulation are clear: if we run Carmel's method with either BM25 (pure probabilistic) or TF-IDF, in every situation the PRP-pruned indexes behave better retrieval-wise (using the same scoring function for ranking). It is also important to stress that the pruning criterion (to decide which posting entries we keep/rule out) is stated in a completely different way than other pruning techniques.

The probabilities presented in section 5.4.2 can be split into term-dependent and term-independent. The most discriminative factor in the derivation is the query likelihood probability. The effect of the document prior (term-independent probability) is to adjust the effect of the estimation for every document. For a given term and a fixed term frequency, the query-likelihood probability, $p(q_i|D,r)$ is penalised for longer documents (considering smoothing, it is a softened maximum-likelihood). When the $p(r|D)/(1-p(r|D))$ component is combined with the query likelihood, this value is softened for longer documents, so it is easier for the term to score below a certain threshold.

Term-dependent probabilities, $p(q_i|D,r)$ and $p(q_i|\overline{r})$, are the most influential element into deciding whether the pruning algorithm is going to rule out or not a term-document entry from the index.

We are assuming that the estimations for the query likelihood and document prior are suitable for pruning purposes. This is supported by the fact that the smoothing method (linear interpolation) chosen for $p(q_i|D,r)$ is a well-known technique for LM-based retrieval, and the document prior is beneficial for retrieval.

The PRP-based pruning algorithm, as detailed in section 5.4.2 involves a number of parameters: λ for the query likehood component (equation 2.20), average $\overline{X}_d$ and typical deviation S_d over the document lengths for the document prior (equation 5.29), and the result of the fit, equation 5.32, (which uses term frequencies) and the number of documents N for the probability of a term given non-relevance. Something remarkable is that, except for the parameter λ, all these parameters are derived from collection statistics. After some preliminary tests, we decided to set empirically λ to 0.6, for every experiment and every collection. Employing these settings implies that for all the runs presented there were no collection-dependent tuned parameters.

We performed an additional experiment in order to assess the robustness of the method with respect to the tunable parameters of scoring functions. We selected the WT10G collection and ran the PRP-based pruning algorithm with BM25, and for every pruned index we measured the performance

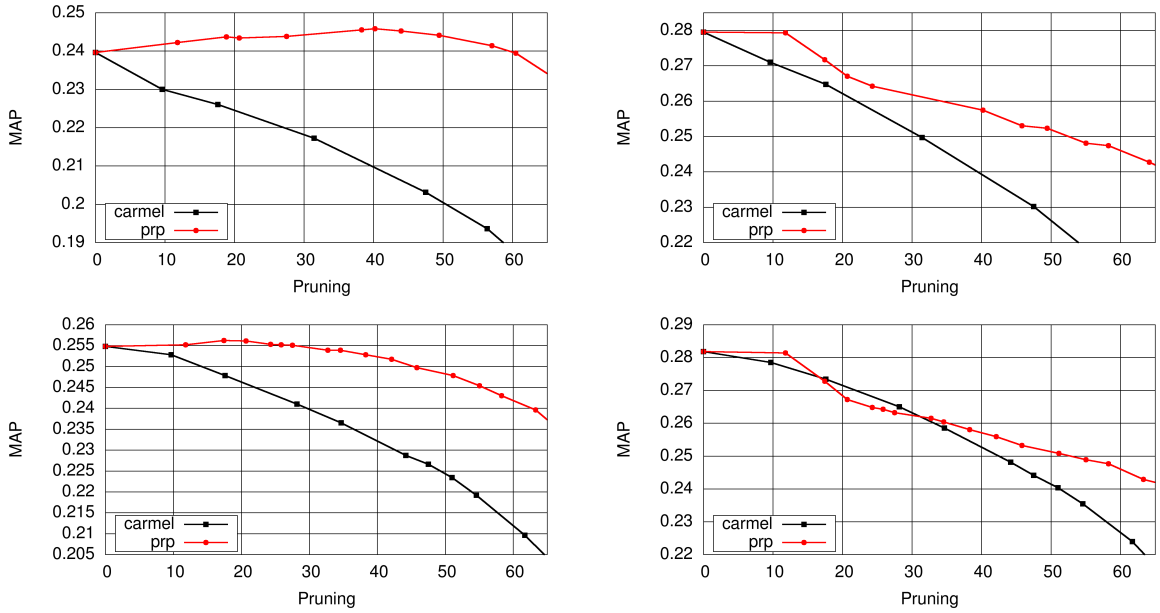


Figure 5.23: PRP vs Carmel pruning, short (left) and long (right) queries, Disks 4&5 BM25 with b=0.75 (top) and the best b (bottom)

using MAP for 10 different values of the b parameter. Then, we selected the median value for every pruning level and plotted both the median and best values in figure 5.22. From the results obtained, it is clear that the median performance is close to the optimum, and therefore, we can conclude that the resulting produced indexes are robust for retrieval and not obtained just by performing collection-dependent parameter tuning. This statement is true for both the parameters involved in the pruning method and scoring function.

In section 5.4.3 we compared our pruning method against a baseline of Carmel's pruning and also with respect to a full index (0% pruning). Our pruning method removed terms with $df > N/2$, however those stopwords were not removed from the full index nor from Carmel's pruned index. Because PRP-based pruning is effectively removing automatically stopwords, it can be argued that its retrieval superiority over Carmel's method could be (at least partially) due to this fact. To investigate this point further, we performed an additional experiment with Disks 4&5 (figure 5.23) and WT10G (figure 5.24) collections. We comment next the results for the TREC Disks 4&5 collection because it has the largest number of topics (250) and also the retrieval performance gains at early pruning levels (<40%) are very noticeable.

We compared both pruning methods by removing exactly the same terms (those with $df > N/2$) at an initial pruning level and ran both algorithms under those conditions. The first point on Carmel's method line in figure 5.23 ($\approx 9\%$) is due just to the removal of those terms; the effect of pruning with $\epsilon = 1$ in PRP-based pruning (see section 5.4.3.6) includes other posting entries removed in the index.

Removing those high frequency terms improves Carmel's method performance and makes differences among both pruning techniques less noticeable. In any case PRP pruning still behaves better, specially with short queries.

We focus now on the number of removed *query* terms. We checked the number of pruned stopwords at every pruning level from the total number of query terms and total number of *different* query terms. Pruning and retrieving short queries using BM25 has a noticeable improvement in retrieval precision at low pruning levels (see figure 5.23). However, the proportion of the query terms' inverted lists pruned under the different methods is a very low percentage of all the query terms and it is constant through all the pruning levels. For pruning levels up to 80%, PRP-based pruning rules out 7 unique query terms out of 538 unique terms appearing in the 250 queries (less than 1.5%). If we consider all non-unique query terms the number of removed terms is 24 out of 691 (less than

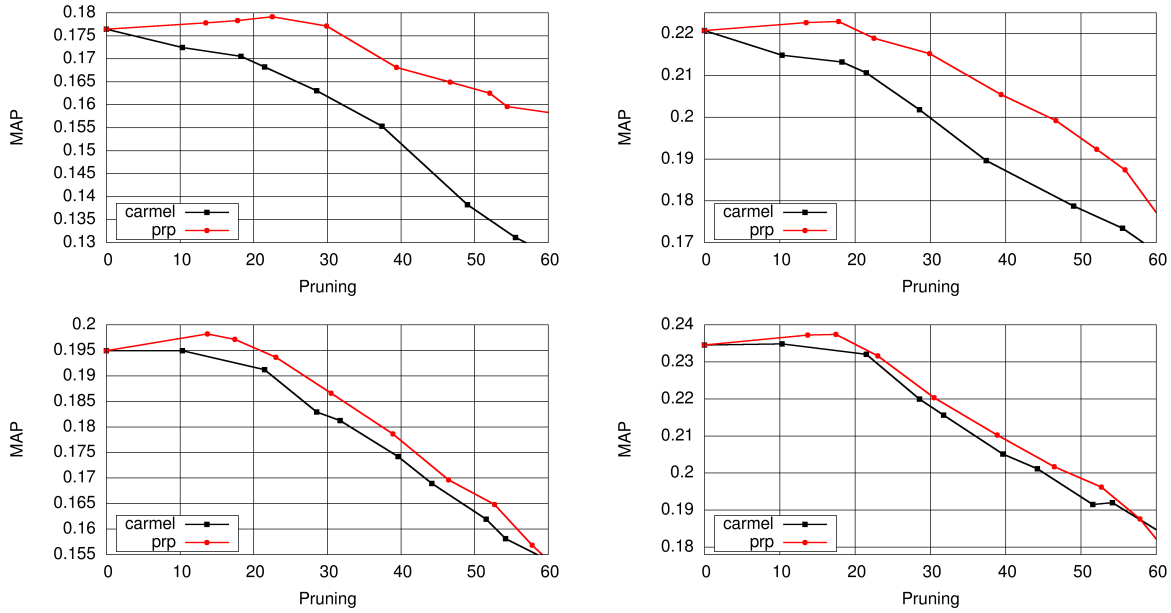


Figure 5.24: PRP vs Carmel pruning, short (left) and long (right) queries, WT10G BM25 with b=0.75 (top) and the best b (bottom)

1.5%).

Any method that discards whole terms entries from the index has the risk of not being able to find any documents for a given query (the query "the who" for instance). If this is an issue, an elegant solution for keeping information on pruned terms could be using a two-tiered index Ntoulas and Cho [2007]: a first layer normally pruned index and a secondary layer index that contains the pruned terms' information. The first index can be used for normal access and answering most of the queries whereas the secondary index would be accessed only if necessary.

5.4.5 Conclusions and future work

The Probability Ranking Principle is the foundation of the probabilistic model of IR, which retrieves documents relevant to a query (see section 5.4.1.1). The fundamental idea behind this section is to introduce this principle for static pruning and not for the estimation of relevance. Specifically, in this section we presented a novel index pruning algorithm based on the same assumptions as probabilistic retrieval models. Contrarily to other pruning approaches, the technique presented here does not aim at maintaining the top results returned by an IR system. Instead, it defines a pruning criterion that relies on the goodness of the estimation of three probabilities.

In order to assess the performance of PRP-based pruning algorithm, we compared it to an enhanced version of Carmel et al.'s algorithm, using standard TREC datasets with queries of different lengths, and three different retrieval models with both default and optimised settings. The experiments showed that our technique outperforms the baseline in terms of MAP and performs at least as good in terms of P@10. Results are consistent across five different collections, 450 ad-hoc queries of two different lengths, 225 topic distillation and named page finding queries, and three retrieval models. Furthermore, a set of experiments allowed us to conclude that the outcome of the pruning algorithm does not depend on a particular parameter tuning of the retrieval model.

The fact that static pruning may improve retrieval precision up to certain pruning levels, may indicate a weakness of the scoring function. It is possible to consider that a *perfect* matching function could leave the index as it is and prune it dynamically, but with high on-line query processing costs (this may be more critical for pruning regimes that operate at a document by document fashion). This can be critical in high-demand environments, or in systems deployed on low-memory devices.

Furthermore, this could open a future line of work for retrieval modeling inspired by the results in pruning performance.

One of the weaknesses of the pruning method presented in this section is that it does not guarantee to preserve the top-k results when using the original unpruned index. This is also the case for the original implementation of Carmel et al's pruning method of the method, which shifts the scores of posting entries and that was presented in section 5.1, and also for the improved baseline presented in section 5.2.1.1. However, if the estimation of the probabilities presented in section 5.4.2 were *ideal*, then PRP-based pruning would bring the optimum amount of pruning for a given index. Then, the top-k results would not be the same - they would be better, or, at least as good as the original top-k.

The approach proposed in this section follows a term-by-term mechanism, and it needs collection statistics that can only be obtained after indexing. For that reason, the pruning algorithm would operate in two different phases. This is a drawback with respect to the approach described in Büttcher and Clarke [2006] where this problem is solved by inspecting a sub-collection and extrapolating the statistics found to the whole collection, and hence pruning can be done while indexing. This could be a future line of research. Forthcoming work could try to dig deeper into the nature and estimation of the probabilities or try to incorporate term co-occurrence into the pruning process in a formal way and also assigning different non-constant costs per document in equation 5.18. For instance, pruning reputably home-pages would imply higher costs than other kinds of documents.

An immediate future line of research is to design and perform suitable statistical test for comparing different pruning methods. Note that traditional significance tests employed in retrieval pose a number of problems in this scenario. First of all, pruning algorithms do not produce a specific pruning level: both operate with a threshold, and the exact amount of pruning cannot be quantisised in advance. This makes difficult the comparison at the same pruning level between different methods or collections; it would be necessary to perform a trial-and-error procedure until both methods produce an index with exactly the same amount of pruning. This would allow for comparing single pruning levels. For obtaining a single value over the whole pruning levels, a first step could be to interpolate the pruning level and compare the curves produced by the pruning methods on the basis of any variable. This could open the way for standard significance tests.

Finally, as stated in section 5.4.4 it could be possible to alter the $p(q_i|\overline{r})$ estimation in such a way that it incorporates more refined stopword detection variants that could lead to even higher benefits to pruning.

Chapter 6

Document Priors

This chapter comes motivated by the need of a reliable estimation for the document prior probability (section 5.4.2.2). Although the estimation presented in last chapter worked well for static pruning, it gave some insights on why a document prior probability might benefit retrieval performance. Thus, this chapter addresses the issue of devising a new document prior for the language modeling (LM) approach for Information Retrieval. The prior is based on term statistics, derived in a probabilistic fashion and portrays a novel way of considering document length. Furthermore, we developed a new way of combining document length priors with the query likelihood estimation based on the risk of accepting the latter as a score. This prior has been combined with a document retrieval language model that uses Jelinek-Mercer (JM), a smoothing technique which does not take into account document length. The combination of the prior boosts the retrieval performance, so that it outperforms a LM with a document length dependent smoothing component (Dirichlet prior) and other state of the art high-performing scoring function (BM25). Improvements are significant, robust across different collections and query sizes.

The main bulk of this chapter has been published in Blanco and Barreiro [2008] and it is organised as follows: section 6.1 introduces the chapter and realted work on document prior usage in IR; section 6.2 describes in detail the formulation on document priors in the context of language modelling for IR (see also section 2.5.1.4); section 6.3 presents a simple well-known linear document prior and two novel length-based document priors: one using a sigmoid function (employed for static pruning in section 5.4.2.2) and a second one that approximates document length in a probabilistic fashion; section 6.4 explores new ways for combining the document prior with the query likelihood, and section 6.5 describes our experimental findings.

6.1 Introduction

As stated throughout this thesis, IR systems aim to retrieve relevant documents in response to a user need, which is usually expressed as a query. The retrieved documents are returned to the user in decreasing order of relevance. Most retrieval models use term statistics, such as term frequency, to assign weights to individual terms, which represent the contribution of the term to the document content. These term weights are then used to estimate the score of relevance of a document for a query (van Rijsbergen [1979]).

In addition to term statistics, IR models are often extended with further evidence that can improve retrieval performance, e.g. using the term frequency in specific fields of structured documents (e.g. title, abstract) (Robertson et al. [2004]), or integrating query-independent evidence in the retrieval model in the form of prior probabilities for a document (Craswell et al. [2005], Kraaij et al. [2002]) ('prior' because they are known before the query). In short, when determining the relevance between a query and a document, most IR models use primarily query terms statistics, and sometimes also add query-independent evidence to further enhance retrieval performance. In this chapter, we propose a new form of prior for documents, which we combine with IR models from the language modeling

(LM) approach (Ponte and Croft [1998]).

Language models for IR view documents as models and queries as segments of text generated or sampled from those models. Documents are ranked according to the probability of each query text string being generated from the respective document model. Although traditionally language models abandoned the explicit notion of document and query *relevance*, the work by Lafferty and Zhai [2003] connects the notion of relevance and generative language models.

The LM framework models the relevance of documents to queries by estimating two probabilities (namely, query likelihood and document prior). Considering a multinomial generation of events (Zhai and Lafferty [2004]), documents are ranked against queries according to those estimations. The query likelihood component is a query dependent feature representing the probability of the query being generated by the language model of a document and the document prior is a query-independent feature representing the probability of seeing the document. Typically, this probability is assumed to be the same for any document, hence the document prior is taken to be uniform (Zhai and Lafferty [2004]). Alternatively, the document prior is useful for representing and incorporating other sources of information to the retrieval process; this is currently an active area of research. For instance, document priors can be derived from the link structure of Web pages. In fact, this is a popular source for priors: Westerveld et al. [2002] introduced the number of incoming links (inlinks) count, which was subsequently used in various Web retrieval tasks of the Text REtrieval Conference (TREC Voorhees and Harman [2005]) repeatedly and with success. Another type of evidence from which document priors are derived is URL depth, also introduced by Westerveld et al. [2002]. These two priors were further explored by the work by Kraaij et al. [2002]. Other URL-derived information and also the Pagerank (Brin and Page [1998]) algorithm for ranking Web documents according to their popularity, have been used to derive document priors (Upstill et al. [2003]).

Overall, incorporating prior knowledge on documents into retrieval has been particularly effective on Web retrieval, namely *homepage* and *named page finding*. Homepage and named page finding refer to the retrieval of a single Web page; on the contrary, *ad-hoc* retrieval refers to the more general application of retrieving as much relevant information to the query as possible.

In this chapter, we revisit the idea of deriving high-quality document priors based on document length and term statistics. Most retrieval models include a document length normalisation component, so that longer documents do not have an unfair advantage over shorter documents of being retrieved. This normalisation is fairly critical and some successful models of retrieval are based in part on document length models, like BM25 (Robertson and Walker [2000]) (see section 2.5.1.3). In section 5.4.2.2 we advanced how to model the document prior probability using a sigmoid function, and the experiments and results reported in section 5.4.3 show the benefit it yields for static pruning. In this chapter, we show that it is possible to encode document length information as a prior probability and improve significantly retrieval effectiveness of a simple language model that uses Jelinek-Mercer (JM) smoothing; these results are complementary to the ones presented in section 5.4.3 for a different retrieval task and taking a different perspective of the prior estimation. In particular, we provide ad-hoc retrieval experiments with two length-based priors: one prior estimated proportionally to document length (we call this prior *linear*) (Kraaij et al. [2002],Westerveld et al. [2002]), and a document length based prior which is not computed directly from the number of tokens in the document but estimated in a probabilistic fashion from term statistics, which are typically used by retrieval models. Deriving a document prior from these term statistics is a novel approach. We also give some insights on the document prior previously presented in section 5.4.2.2.

Generally, priors are combined with the retrieval model either using heuristics or handtuned parameters (Craswell et al. [2005]). In this work, we combine the proposed priors with the LM in two different ways: using a standard logarithmic combination, and proposing a novel combination that considers the prior as a measure of the risk of accepting the score given by the query likelihood estimation of the LM. A thorough experimentation on four TREC collections of different size and domain, and 450 short and long queries show that our proposed prior benefits retrieval performance significantly, and in a robust way. Specifically, we find that it is possible to boost the performance of a retrieval model based on JM smoothing up to values comparable to state of the art retrieval models, and further outperform retrieval models traditionally considered to be more effective in previous literature (Westerveld et al. [2002]).

6.2 Document priors in the language modeling approach

The language modeling framework (section 2.5.1.4) allows a mathematically elegant way of incorporating query-independent features, i.e. just related to a document *without seeing a query*. Next, it follows a derivation of the LM retrieval model where the probability of relevance $p(r|Q,D)$, given a query and a document is estimated indirectly by invoking Bayes' rule. For the formal connection between language models and the probabilistic model of retrieval refer to (Lafferty and Zhai [2003]), or equation 5.21.

Let the random variables D and Q denote a document and a query, respectively. Let the binary random variable R stand for relevance r, $p(r) = p(R = 1)$ and non-relevance $\overline{r}$, $p(\overline{r}) = p(R = 0)$.

$$p(r|Q,D) = \frac{p(D,Q|r)\ p(r)}{p(D,Q)} \tag{6.1}$$

$$= p(Q|D,r)\ p(D|r)\ \frac{p(r)}{p(D,Q)} \tag{6.2}$$

$$= p(Q|D,r)\ p(r|D)\frac{p(D)}{p(D,Q)} \tag{6.3}$$

Assuming independence between queries and documents $p(D,Q) = p(D)p(Q)$, and given that $p(Q)$ does not affect the ranking (it is document-independent), equation 6.3 becomes

$$p(r|Q,D) = \frac{p(Q|D,r)\ p(r|D)}{P(Q)} \stackrel{rank}{=} p(Q|D,r)\ p(r|D)\ , \tag{6.4}$$

where $p(Q|D,r)$ is the query likelihood and $p(r|D)$ is the document prior. In equation 6.4, we took a strong independence assumption to get a final formulation with dependence on $p(r|D)$. The derivation presented by Lafferty and Zhai [2003] (equation 5.21) took a more reasonable assumption, Q and D are independent under $\overline{r}$, and starting from the odds-ratio of relevance the final relevance score is dependent on $p(r|D)/(1-p(r|D))$.

It is usual to decompose the query into its query terms $Q = \{q_1, q_2, \ldots, q_n\}$ and assume that, given relevance and the document, they are independent of each other and generated by a multinomial distribution.

$$p(Q|D,r) = \prod_{i=1}^{n} p(q_i|D,r) \tag{6.5}$$

In order to rule out zero probabilities for non-seen terms in a document, this estimate has to be *smoothed*, which eventually leads to different language models-based scoring functions. Most smoothing methods employ two distributions, one for words occurring in the document (p_s) and one for *unseen* words (p_u). Taking logs (refer to equations 2.15- 2.19 for a complete derivation) it can be shown that equation 6.6 suffices to provide a document rank using sums of logarithms, equivalent to the one that equation 6.4 would yield.

$$\log p(Q|D,r) \stackrel{rank}{=} \sum_{i \backslash tf(q_i,D)>0} \log \frac{p_s(q_i|D)}{\alpha_d p(q_i|\mathcal{C})} + n \cdot \log \alpha_d\ , \tag{6.6}$$

where $tf(q_i,D)$ stands for the frequency of term q_i in document D, α_d is a parameter and $p(q_i|\mathcal{C})$ is the collection language model.

The smoothing technique we considered as the baseline in this study is Jelinek-Mercer (JM) (also known as linear interpolation):

$$p_s(q_i|D) = (1-\lambda)p_{mle}(q_i|D) + \lambda\ p(q_i|\mathcal{C}),\ \lambda \in [0,1]\ , \alpha_d = \lambda \tag{6.7}$$

where $|D| = \sum_{w_i \in D} tf(w_i, D)$ (the document length), p_{mle} is the maximum likelihood estimator for a term q_i given a document d, $p_{mle}(q_i|D) = \frac{tf(q_i,d)}{|D|}$ and λ is a parameter controlling the amount of mass distribution assigned to the *document* and *collection*.

Another popular and effective smoothing technique is Dirichlet prior smoothing:

$$p_s(q_i|D) = \frac{tf(q_i, D) + \mu p(q_i|\mathcal{C})}{|D| + \mu},\ \alpha_d = \frac{\mu}{|D| + \mu}\ , \tag{6.8}$$

where μ is a parameter. In most cases $p(r|D)$ is taken to be uniform (Zhai and Lafferty [2004]). However, there have been several studies where the document length and link structure have been encoded as a prior probability, for ad-hoc and some non ad hoc tasks (Kraaij et al. [2002], Westerveld et al. [2002]).

Most weighting models include document length as a part of their core query-dependent retrieval model and that might be one of the reasons for traditionally not being considered a document static feature. For most retrieval models, the amount of normalisation contributed by document length is controlled by a parameter. This is not the case for JM smoothing, but it can be seen that the μ parameter in Dirichlet prior smoothing is playing the length normalisation role. The weight for a matched query term q_i in JM smoothing is $\log(1 + (1-\lambda)p_{mle}(q_i, D)/\lambda p(q_i|\mathcal{C})))$ and for Dirichlet prior smoothing is $\log(1 + |D|p_{mle}(q_i, D)/(\mu p(q_i|\mathcal{C})))$. Clearly, $|D|/\mu$ and $(1-\lambda)/\lambda$ play the same role, with the difference that the former is document-dependent while the latter is document-independent (Zhai and Lafferty [2004]). It is assumed from past studies (Zhai and Lafferty [2004],Westerveld et al. [2002]), that Dirichlet prior smoothing outperforms JM smoothing, especially for short queries. In our opinion, this is due to the fact that Dirichlet prior smoothing includes document length normalisation as a part of the query likelihood estimation.

Although JM smoothing does not comprise document dependent length normalisation notions, it has the advantage of "explaining" the common words of the query. This is the reason JM behaves better with long queries: these kind of queries are usually more verbose. Experiments using short-verbose queries by Zhai and Lafferty [2004] confirmed the query-modeling role of JM smoothing. Otherwise, it is assumed that Dirichlet prior smoothing has an effect of improving the accuracy of the estimated document language model. Incorporating a good document length prior into LM-JM would hopefully result in a model that will embody both roles mentioned before.

6.3 Length-based document priors

6.3.1 Linear Prior

Previous studies (Singhal et al. [1996], Kraaij et al. [2002]), tried to establish a connection between the likelihood of relevance/retrieval and document length. In particular, Singhal et al. [1996] compared the results of a set of queries and tried to obtain a relevance versus document length and retrieval versus document length patterns (of a particular scoring function) to see how they deviate from each other. The relevance pattern happened to follow a linear dependence on document length. The results presented by Kraaij et al. [2002] on a another testbed, further confirmed that hypothesis. Then, our first document length based prior is proportional to document length. The intuition behind this prior is that longer documents span more topics and are more likely to be relevant *if no query has been seen* (denoted as *scope* hypothesis by Robertson and Walker [1994]). It has been reported that this prior increases the retrieval performance (Kraaij et al. [2002]) on the WT10G collection up to 0.03 on an absolute scale.

The linear document prior is given by:

$$p(r|D) \approx \frac{|D|}{\sum_{D_i \in \mathcal{C}} |D_i|} = C \cdot |D|\ , \tag{6.9}$$

where C is a constant that can be dropped out from the scoring function since it does not affect the ranking of documents.

6.3.2 Sigmoid prior

We presented this prior in section 5.4.2.2, and stated that it had an "S-like" shape, derived from the use of a sigmoid function. This prior is also based on document length and similar to the linear prior, but with two main differences. First of all, this prior can be parameterised so that probabilities adjust to collection characteristics. The second difference is that the range of probabilities values are different for both priors. The linear prior assumes that $p(r|D) = 1 \; iff \; |D| = T \Leftrightarrow N = 1$, ($T$ stands for the total number of tokens in the collection) whereas the sigmoid prior assigns $p(r|D) = 1 \; iff \; |D| \geq \overline{X_d} + k \cdot S_d$. Also, recall that $avgdl << N$ and thus the linear prior would assign a very low probability to the majority of the documents in the collection (lower as the collection size grows). The sigmoid prior assigns $p(r|D_{|D|=avgdl}) = 0.5$. Also, the event space between the priors is different. Recall that the following estimates relationship holds for the linear prior:

$$\sum_{D_i \in \mathcal{C}} p(r|D_i) = 1 \; , \tag{6.10}$$

where N is the total number of documents in the collection. The estimation considered for the linear prior implies that there is a probability distribution for the event r conditioned to $|D|$. This is, the event space is formed by r under D_i, where D_i is every different document in the collection. The outcome of the distribution specifies the probability that the random variable takes each of the different possible length values. Instead, the sigmoid prior estimates the conditional probability $p(r|D)$, where r and D are random variables. Typically, the linear prior would assign a document with average document length a probability of $1/N$, which is much lower than the 0.5 probability value the sigmoid prior would estimate. This is fully extended in section 6.3.3, where r and D are considered random variables themselves and thus the estimation of the prior is radically different than the one the linear prior implies.

The different event space consideration is one of the reasons why the range of values is different, and why the linear prior is of little usage for pruning in the framework developed in section 5.4.

This prior has been extensively tested (with its default settings) for static pruning (section 5.4.3) and thus we omit its usage for ad-hoc retrieval: the prior includes a number of parameters that would have to be trained or tuned, which is out of the scope of this chapter, but opens future lines of research.

6.3.3 Probabilistic Prior

We propose another prior indirectly based on document length by extending the idea of estimating the document prior as a function depending on the statistics of the terms it contains.

To estimate the conditional probability $p(r|D)$ we compute the expectation over the universe of terms $\{w_i\}$. Also, in 6.11 we make the additional assumption that r is independent of D once we picked a term w_i.

$$p(r|D) \approx \sum_{w_i} p(r|w_i)p(w_i|D) \tag{6.11}$$

$$= \sum_{w_i \in D} p(r|w_i)p(w_i|D) + \sum_{w_i \notin D} p(r|w_i)p(w_i|D) \tag{6.12}$$

$$\approx \sum_{w_i \in D} p(r|w_i)p(w_i|D) = \sum_{w_i \in D} \left(1 - p(\overline{r}|w_i)\right) p(w_i|D) \tag{6.13}$$

$$= \sum_{w_i \in D} \left(1 - p(w_i|\overline{r}) \frac{p(\overline{r})}{p(w_i)}\right) p(w_i|D) \approx \sum_{w_i \in D} \left(1 - p(w_i|\mathcal{C}) \frac{p(\overline{r})}{p(w_i)}\right) p(w_i|D) \tag{6.14}$$

$$\approx \sum_{w_i \in D} p(w_i|D) \tag{6.15}$$

In the derivation we made the following assumptions, in order to obtain a simple model for the prior. In 6.12, $p(w_i|D) \approx 0$ if $w_i \notin D$. In 6.13 $p(w_i|\overline{r}) \approx p(w_i|\mathcal{C})$; this assumes the collection to be a

model of *non-relevance*, which goes accordingly to the hypothesis taken by Croft and Harper [1979], that every document is non-relevant (and eventually leading to the inverse document frequency formula as we know it). Lastly, in 6.14, it is assumed for convenience that $p(\overline{r})p(w_i|\mathcal{C}) << p(w_i)$.
The final form of this prior comes from the distribution for the terms on a document, by smoothing the maximum likelihood estimator as follows:

$$p(r|D) \approx \sum_{w_i \in D} p(w_i|D) \tag{6.16}$$

$$= \sum_{w_i \in D} \left[(1-\lambda') \frac{tf(w_i,d)}{|D|} + \lambda' p(w_i|\mathcal{C}) \right] \tag{6.17}$$

$$= (1-\lambda') + \lambda' \cdot \sum_{w_i \in D} p(w_i|\mathcal{C}) \tag{6.18}$$

In this work, it is not required that the document model employed in the prior and the document model used to compute the query likelihood be the same. The former, has a parameter, $\lambda' \in [0,1]$, coming out from the JM smoothing formula.
The result of this derivation results in a prior obtained from the sum of the individual contributions of each term occurring in the document. The linear document length-based prior (equation 6.9) has a similar form: it is a sum over the document terms frequencies, floored by a constant:

$$p(r|D) \approx \frac{1}{\sum_{D_i \in \mathcal{C}} |D_i|} \cdot \sum_{w_i \in D} tf(w_i, D) \tag{6.19}$$

An interesting remark, is that if

$$p(w_i|D) \approx \phi \; \forall \; w_i \; , \tag{6.20}$$

with ϕ constant, then, the value of the probabilistic prior $p(r|D)$ is proportional to the different number of terms in the document.

The probabilistic prior is higher for documents with common terms than for documents with many rare terms, which may seem counter-intuitive. Note that the probabilistic prior counts the contribution of a term only once, despite of its document frequency. Hence, documents with many *different common* words will receive a higher prior value. Very common stopwords are likely to appear in every document, and therefore their effect is the same for every document. However, in heterogeneous collections, there may be a number of keywords describing its different topics or clusters. Keywords are likely to be frequent (at least inside the clusters), and documents containing many of those terms will be promoted in the rank list by the prior. This goes accordingly to the *scope* hypothesis (Robertson and Walker [1994]): documents covering many topics are more likely to be relevant.

6.4 Combination of the prior and the query likelihood

In order to evaluate both priors we combine them with the query likelihood $p(Q|D,r)$ component in two different ways: a *standard* logarithmic sum and a novel method presented below. If we follow a log sum derivation from equation 2.14 then, the standard way of combining the document prior with the query likelihood estimation in order to produce a document score would be:

$$score(D,Q) = \log p(Q|D,r) + \log p(r|D) \tag{6.21}$$

We further devised a new prior-query likelihood combination, taking into account the fact that probability estimates for longer documents are more reliable than for shorter ones. We modeled this fact by considering the risk of accepting a certain score s, $\hat{R}(s) \in [0,1]$. It is possible to bias s and calculate a new score for the document and query $score(Q,D)$ as

$$score(Q,D) = s^{1-\hat{R}(s)} \tag{6.22}$$

Collection	size	Topics	# queries
LATimes	450 MB	401-450	50
TREC disks 4&5	2G	301-450+601-700	250
WT2g	2G	401-450	50
WT10g	10G	451-550	100

Table 6.1: Collections and topics employed in the document priors experiments

Taking into account the fact that *longer* documents may provide a better estimate of $p(Q|D,r)$, it is reasonable to associate the document prior $p(r|D)$ with $1-\hat{R}(s)$, resulting in

$$score(Q,D) = scoreLM(Q,D)^{\hat{p}(r|D)} \tag{6.23}$$

or in logarithmic notation

$$score(Q,D) \stackrel{rank}{=} \hat{p}(r|D) \cdot \log(scoreLM(Q,D)) \ , \tag{6.24}$$

where $scoreLM(Q,D)$ stands for the score a language model assigns to document D under a query Q. We combined both priors (linear and probabilistic) with the query-likelihood using this new approach. However, for the risk-based combination and linear prior, we modified the document length with a logarithmic transformation given that the probability of relevance versus logarithm of document length curve seems to be approximately linear in some ranges (Singhal et al. [1996]):

$$score(Q,D) = \log(|D|) \cdot \log(scoreLM(Q,D)) \tag{6.25}$$

6.5 Experiments and Results

The main goal of these experiments is to evaluate the effectiveness of both priors and combinations proposed before, and assess their effect on retrieval. To evaluate the new priors and combinations, we plug them into a LM with Jelinek-Mercer smoothing (equation 6.7). This scoring function without the prior serves as the baseline.

The TREC datasets used are described in table 6.5. The collections differ in size and domain, hence they represent a broad and varied experimental dataset. We experiment with short (title-only) and long (title, description and narrative) queries. We apply the standard Porter stemming algorithm, and skip any stopwords removal, in order to avoid any bias by any choice of stoplist[1]. For all the retrieval experiments we use the Terrier IR platform[2].

The metrics used are Mean Average Precision (MAP), precision at top ten retrieved documents (P@10) and binary preference (BPref) (Buckley and Voorhees [2004]). The value of the λ parameter in JM smoothing (with and without priors) has been optimised for every measure in every collection by using increasing values of 0.05 in the range (0,1]. We performed a preliminary tuning for the λ' parameter in some datasets (values increasing in 0.1 steps), and decided to set it to 0.7 for every collection. We report that it is possible to obtain marginal gains if λ' is tuned specifically for a given collection, but that step is omitted in order to prove the robustness of the technique.

The experiments presented next, compare separately the best scores produced by the two priors and two ways of combining them with the query-likelihood, with the best scores the LM-JM baseline produces. Finally, the best performing prior and combination is compared against two state of the art retrieval models (LM with Dirichlet prior smoothing and BM25).

Table 6.2 presents the results for all the priors and combinations. The first column is the type of prior and combination used. JM is the baseline (without any prior). LP C1 stands for the model that uses the standard log sum combination (equation 6.21) and a linear prior (equation 6.9), and this

[1] we repeated these experiments using a standard stop-word list and the conclusions derived from this experimentation are the same

[2] `http://ir.dcs.gla.ac.uk/terrier/` (accessed on 03-04-08)

LATimes					
Model	MAP	P@10	Bpref	Δ%	p-value
JM	0.2322	0.2711	0.2275	–	–
LP C1	0.2560	0.2889	0.2398	10.24	0.323
PP C1	0.2332	0.2680	0.2256	0.43	0.642
LP C2	0.2591	0.2784	0.2370	11.58	0.149
PP C2	**0.2685**	**0.2889**	**0.2507**	**15.63**	**0.043***
Disks 4&5					
Model	MAP	P@10	Bpref	Δ%	p-value
JM	0.2333	0.3908	0.2395	–	–
LP C1	0.2544	0.4313	0.2583	9.04	**≈ 0***
PP C1	0.2377	0.3996	0.2479	1.72	**≈ 0***
LP C2	0.2535	0.4307	0.2570	8.65	**≈ 0***
PP C2	**0.2639**	**0.4454**	**0.2651**	**13.11**	**≈ 0***
WT2g					
Model	MAP	P@10	Bpref	Δ%	p-value
JM	0.2495	0.3480	0.2407	–	–
LP C1	0.3110	0.4660	0.2946	24.64	**≈ 0***
PP C1	0.2572	0.3760	0.2507	3.09	**0.013**
LP C2	0.3123	0.4640	0.2998	25.17	**≈ 0***
PP C2	**0.3335**	**0.4820**	**0.3182**	**33.66**	**≈ 0***
WT10g					
Model	MAP	P@10	Bpref	Δ%	p-value
JM	0.1479	0.2469	0.1474	–	–
LP C1	0.1926	0.2959	0.1889	30.22	**≈ 0***
PP C1	0.1574	0.2582	0.1597	6.42	**≈ 0***
LP C2	0.1939	0.3153	0.1928	31.10	**≈ 0***
PP C2	**0.1984**	**0.3316**	**0.1956**	**34.14**	**≈ 0***

LATimes					
Model	MAP	P@10	Bpref	Δ%	p-value
JM	**0.3010**	0.3067	**0.2865**	–	–
LP C1	0.2696	0.2978	0.2527	-10.43	0.059
PP C1	0.2937	**0.3200**	0.2848	-2.43	0.259
LP C2	0.2856	0.2978	0.2511	-5.01	0.669
PP C2	0.2996	0.3044	0.2861	-0.46	0.986
Disks 4&5					
Model	MAP	P@10	Bpref	Δ%	p-value
JM	0.2844	0.4791	0.2838	–	–
LP C1	0.2731	0.4514	0.2741	-3.97	**0.019***
PP C1	0.2849	0.4847	0.2876	0.18	0.337
LP C2	0.2822	0.4711	0.2783	-0.77	0.537
PP C2	**0.2967**	**0.4984**	**0.2992**	**4.32**	**≈ 0***
WT2g					
Model	MAP	P@10	Bpref	Δ%	p-value
JM	0.2678	0.4300	0.2748	–	–
LP C1	0.2871	0.4660	0.2925	7.20	0.184
PP C1	0.2750	0.4280	0.2796	2.69	0.120
LP C2	0.3017	0.4580	0.3010	12.65	0.112
PP C2	**0.3145**	**0.4840**	**0.3138**	**17.43**	**≈ 0***
WT10g					
Model	MAP	P@10	Bpref	Δ%	p-value
JM	0.2274	0.3850	0.2202	–	–
LP C1	0.2298	0.3730	0.2338	1.05	0.592
PP C1	0.2312	0.3890	0.2291	1.67	**0.005***
LP C2	0.2366	0.3810	0.2297	4.04	0.291
PP C2	**0.2509**	**0.4020**	**0.2351**	**10.33**	**≈ 0***

Table 6.2: Optimal performance comparison of JM with the different priors and combinations for short (left) and long (right) queries. Best values are bold. Significant MAP differences according to the Wilcoxon test ($p < 0.05$) are bold and starred

is the only prior-combination form out of the four presented that can be found in previous studies (Kraaij et al. [2002]). PP C1 stands for the probabilistic prior (equation 6.15) and the log sum combination. LP C2 stands for the linear prior and the new risk-based query likelihood combination (equation 6.25). Finally, PP C2 denotes the new probabilistic prior and risk-based combination (equation 6.24). The $\Delta\%$ column stands for the MAP difference between the row value and the baseline. The p-value reported in the last column is obtained from the standard Wilcoxon-paired ranks sign test for the MAP results of the prior in that row and the baseline. Significant values ($p < 0.05$) are bold and starred. The best values in each column for the three measures used are bold.

Results show that under the linear combination C1, the linear prior LP performs better for short queries whereas the probabilistic prior P2 is slightly better with long queries (in three out of four collections). Overall, improvements respect to the baseline are significant with short queries and not significant with long queries under combination C1. The risk-based combination C2 is able to improve the performance of both priors in almost every case. The behaviour of the priors changed in this case, and PP performed better than LP with queries of any size. In any case, the probabilistic prior under this combination always yielded the best performance among all combinations and methods tested, with some impressive improvements. Effectiveness gains are higher with shorter queries, which may be due to the fact that JM smoothing performs better for longer queries, and reduces the importance of the length normalisation step in those cases.

One possible explanation for the different behaviour of both prior combinations may be due to the contribution of the prior with respect to the contribution of the query likelihood. The linear combination C1 sums the logarithm of query likelihood and prior; as the query likelihood increases (by adding more query terms) the prior contribution (query independent) diminishes. The probabilistic prior contribution does not affect much the final results when combined this particular way. A high query likelihood score is not so dominant with the risk-based combination C2: the prior is still important for the final score because the combination multiplies the prior by the query likelihood logarithm. Another result is that the effect of the prior is not very sensitive to query length with the C2 combination.

This opens the new issue of further tuning the linear combination. We present in table 6.3 a simple manual tuning of the combination for prior PP (the same for every collection). The form of the prior

LATimes					
Model	MAP	P@10	Bpref	Δ%	p-value
JM	0.2322	0.2711	0.22275	–	–
P2 C1	0.2572	0.2844	0.2402	10.76	**0.011***
Disks 4&5					
Model	MAP	P@10	Bpref	Δ%	p-value
JM	0.2333	0.2908	0.2395	–	–
P2 C1	0.2556	0.4309	0.2581	9.56	**≈ 0***
WT2g					
Model	MAP	P@10	Bpref	Δ%	p-value
JM	0.2495	0.3480	0.2407	–	–
P2 C1	0.3003	0.4460	0.2859	20.36	**≈ 0***
WT10g					
Model	MAP	P@10	Bpref	Δ%	p-value
JM	0.1479	0.2469	0.1474	–	–
P2 C1	0.1929	0.3112	0.1912	30.42	**≈ 0***

LATimes					
Model	MAP	P@10	Bpref	Δ%	p-value
JM	0.3010	0.3067	0.2865	–	–
P2 C1	0.2948	0.3022	0.2833	-2.05	0.429
Disks 4&5					
Model	MAP	P@10	Bpref	Δ%	p-value
JM	0.2844	0.4791	0.2838	–	–
P2 C1	0.2887	0.4831	0.2879	1.51	**≈ 0***
WT2g					
Model	MAP	P@10	Bpref	Δ%	p-value
JM	0.2678	0.4300	0.2748	–	–
P2 C1	0.3071	0.4800	0.3075	14.67	**≈ 0***
WT10g					
Model	MAP	P@10	Bpref	Δ%	p-value
JM	0.2274	0.3850	0.2202	–	–
P2 C1	0.2482	0.4050	0.2348	9.14	**≈ 0***

Table 6.3: Optimal performance comparison of JM with the boosted linear combination (equation 6.26) of the probabilistic prior for short (left) and long (right) queries. Best values are bold. Significant MAP differences according to the Wilcoxon test ($p < 0.05$) are bold and starred

with the new parameter setting is as follow. This particular ad-hoc tuning of the prior contribution brings significant improvements in every situation with respect to the standard C1 combination. Furthermore, the prior under this new combination performs better than the linear prior on its own.

$$15 \cdot \log(1 + p(r|D)) \tag{6.26}$$

A second batch of experiments compared the new prior and combination developed in this work, probabilistic prior with the risk combination, with LM and Dirichlet prior smoothing and also against BM25. The comparison is fair, as this two matching functions already incorporate a document-dependent normalisation factor. Dirichlet prior smoothing is presented in equation 6.8. The μ parameter chosen is the one that optimised the performance for each metric in every collection, picked up from a reasonable set of possible choices[3]. The second weighting function considered was the probabilistic Okapi's Best Match25 (BM25) (Robertson and Walker [2000], section 2.5.1.3) which has proved to be robust, high-performing and stable in many IR studies. The behaviour of the BM25 scores is governed by three parameters, namely k_1, k_3, and b. Some studies (He and Ounis [2003]) have shown that both k_1 and k_3 have little impact on retrieval performance, so for the rest of this chapter they are set as constant to the values recommended by Robertson and Walker [2000] ($k_1 = 1.2$, $k_3 = 1000$, see also section 2.5.1.3). The b parameter controls the document length normalisation factor and it has been optimised in the same way as λ for JM (parameter exploration in the $(0, 1]$ range with 0.05 steps), independently for each metric and collection. The p-values and $\Delta\%$ differences reported in table 6.4 are calculated considering the Dirichlet prior/BM25 run as a baseline and compared to the JM+prior (PP C2) MAP values. A positive Δ% value on a row for Method X, means that JM+prior performed *better* than X.

This second set of results is presented in table 6.4. These results prove that the PP C2 combination is able to outperform significantly high-performing retrieval matching functions in most cases (again, LATimes and long queries being the exception). It is possible to conclude that by including a high-quality length prior, JM smoothing outperforms Dirichlet prior smoothing, which was considered superior, and also well-tuned BM25.

Last, table 6.5 compares the performance of the median values among every parameter tested (b for BM25, λ for PP, and μ for Dirichlet prior). Results show that the probabilistic prior combined with JM smoothing is stable for short queries, and still brings the best median performance in most collection/metrics (except MAP and WT10G collection), whereas underperforms for long queries, where BM25 performs better than language models. These results go accordingly to the ones presented by Zhai and Lafferty [2004], where JM on its own proved to be robust for short queries. The instability of the new prior with respect to long queries opens future work.

[3] $\mu \in \{100, 500, 800, 1000, 2000, 3000, 4000, 5000, 8000, 10000\}$

LATimes					
Model	MAP	P@10	Bpref	Δ%	p-value
JM-Prior	**0.2685**	0.2889	**0.2507**	-	-
BM25	0.2586	**0.2978**	0.2398	3.82	**0.041***
Dirichlet	0.2572	0.2889	0.2355	4.39	**0.017***
Disks 4&5					
Model	MAP	P@10	Bpref	Δ%	p-value
JM-Prior	**0.2639**	**0.4454**	**0.2651**	-	-
BM25	0.2548	0.4402	0.2565	3.57	**0.047***
Dirichlet	0.2559	0.4329	0.2569	3.12	**≈ 0***
WT2g					
Model	MAP	P@10	Bpref	Δ%	p-value
JM-Prior	**0.3335**	**0.4820**	**0.3182**	-	-
BM25	0.3205	0.3560	0.3039	4.05	0.250
Dirichlet	0.3087	0.4500	0.2924	8.03	**0.002***
WT10g					
Model	MAP	P@10	Bpref	Δ%	p-value
JM-Prior	**0.1984**	**0.3316**	**0.1956**	-	-
BM25	0.1954	0.3102	0.1872	1.53	0.45
Dirichlet	0.1932	0.2898	0.1887	2.69	**0.035***

LATimes					
Model	MAP	P@10	Bpref	Δ%	p-value
JM-Prior	0.2996	0.3044	0.2861	-	-
BM25	0.3022	0.3044	0.2870	-0.86	0.450
Dirichlet	**0.3061**	**0.3111**	**0.2970**	-2.12	0.604
Disks 4&5					
Model	MAP	P@10	Bpref	Δ%	p-value
JM-Prior	**0.2967**	**0.4984**	**0.2992**	-	-
BM25	0.2825	0.4896	0.2814	5.03	**0.004***
Dirichlet	0.2743	0.4667	0.2737	8.27	**≈ 0***
WT2g					
Model	MAP	P@10	Bpref	Δ%	p-value
JM-Prior	**0.3145**	**0.4840**	**0.3138**	-	-
BM25	0.2833	0.4600	0.2910	11.01	**0.060***
Dirichlet	0.2906	0.4280	0.2805	8.22	**0.012***
WT10g					
Model	MAP	P@10	Bpref	Δ%	p-value
JM-Prior	**0.2509**	**0.4020**	**0.2351**	-	-
BM25	0.2319	0.3940	0.2295	8.19	**0.012***
Dirichlet	0.2435	0.3910	0.2223	3.03	0.2708

Table 6.4: Optimal performance comparison between JM+probabilistic prior, Dirichlet prior smoothing and BM25 on different collections for short (left) and long (right) queries. Best values are bold. Significant MAP differences according to the Wilcoxon test ($p < 0.05$) are bold and starred

LATimes				
Model	MAP	P@10	Bpref	parameter
JM-Prior	0.**2589**	**0.2889**	**0.2438**	λ =0.7
BM25	0.2513	0.2867	0.2350	b =0.25
Dirichlet	0.2448	0.2733	0.2266	μ =800
Disks 4&5				
Model	MAP	P@10	Bpref	parameter
JM-Prior	**0.2609**	**0.4438**	**0.2653**	λ =0.65
BM25	0.2484	0.4325	0.2555	b =0.4
Dirichlet	0.2429	0.4137	0.2489	μ =800
WT2g				
Model	MAP	P@10	Bpref	parameter
JM-Prior	**0.3230**	**0.4760**	**0.3096**	λ =0.3
BM25	0.2921	0.4620	0.2851	b =0.15
Dirichlet	0.2922	0.4400	0.2783	μ =2000
WT10g				
Model	MAP	P@10	Bpref	parameter
JM-Prior	0.1814	**0.3041**	**0.1838**	λ =0.85
BM25	**0.1869**	0.2929	0.1806	b =0.35
Dirichlet	0.1833	0.2714	0.1826	μ =2000

LATimes				
Model	MAP	P@10	Bpref	p-value
JM-Prior	0.2114	0.2556	0.1975	λ =0.9
BM25	0.2610	**0.2930**	0.2314	b =0.95
Dirichlet	**0.2685**	0.2867	**0.2316**	μ =500
Disks 4&5				
Model	MAP	P@10	Bpref	parameter
JM-Prior	0.2250	0.4040	0.2357	λ =0.85
BM25	**0.2741**	**0.4755**	**0.2750**	b =0.85
Dirichlet	0.2670	0.4506	0.2684	μ =3000
WT2g				
Model	MAP	P@10	Bpref	parameter
JM-Prior	0.2588	0.410	0.2744	λ =0.8
BM25	**0.2711**	**0.4400**	**0.2824**	b =0.2
Dirichlet	0.2567	0.4241	0.2550	μ =10000
WT10g				
Model	MAP	P@10	Bpref	parameter
JM-Prior	0.1872	0.3210	0.2016	λ =0.85
BM25	0.2235	**0.3770**	**0.2274**	b =0.65
Dirichlet	**0.2281**	0.362	0.2192	μ =2000

Table 6.5: Median value performance (MAP, P@10 and BPref) for every different parameter value tried; methods JM+probabilistic prior, Dirichlet prior smoothing and BM25 on different collections for short (left) and long (right) queries. Best values are bold.

6.6 Conclusions

In this chapter we developed two different document priors: one sigmoid prior (previously used for static pruning in section 5.4) and probabilistic document prior that takes into account term statistics and give a probabilistic derivation for it. The effect of the priors in retrieval is also dependent on the way they are combined with the query likelihood. Hence, we also demonstrated the effectiveness of a new way of combining document-length based priors with the query likelihood, that leverages the effect of the prior and likelihood components. The prior boosts the performance of a LM based on JM smoothing significantly, with robust and stable results across collections of different nature and topics of different sizes. The retrieval effectiveness of JM with the new prior is also able to outperform LM using Dirichlet prior smoothing and BM25, when the optimal parameters are used for all of them, and on the basis of three different effectiveness measures. The excellent outcome in terms of retrieval effectiveness of the prior and risk-based combination opens the way for future research directions, for instance we could will try to address the problem of using this new developed way of considering document length into other retrieval matching functions, and other retrieval tasks.

Chapter 7

Conclusions and Future Research

This chapter summarises the conclusions of this thesis and suggests future research directions.

7.1 Conclusions

This thesis investigated two novel inverted files compression methods developed for the Information Retrieval field, namely *document identifier reassignment* and *static pruning*. In particular, the main claim behind this dissertation has been how techniques developed to enhance the effectiveness of IR systems can be successfully adapted also to improve their efficiency, and how both main topics are intertwined.

- Section 4.1 investigated the *clustering* or *clumpiness* occurrence of terms in documents in a text collection, and compared the occurrence distribution with a randomised one. This gives a clear insight on how and why a document identifier reassignment would work for compression.
- Section 4.3 characterised formally the document identifier assignment problem which is a form of rearranging the index in order in order to its compressibility to be maximised. We stated its NP-completeness by reformulating the problem as a *pattern sequencing problem* and also to a *graph-layout problem*. This result linked the well-known world of graph-layout problems with the assignment problem.
- We presented several approaches to tackle the document identifier reassignment problem. Section 4.6 presented an efficient solution of the document identifier reassignment problem using an heuristic for the travelling salesman problem after a dimensionality reduction. This is a novel application of SVD in retrieval, typically used for latent semantic indexing. Section 4.7 experimented with a efficient blocking strategy to reduce the computational overhead time. Finally, we presented a combination of clustering and TSP-based solutions (section 4.8). All these techniques outperformed state-of-the-art solutions for document identifier reassignment in terms of space occupancy. Finally, section 4.9 showed a comparison of the different techniques and gave insights into both theoretical aspects and scalability issues of the proposed work, as well as future research lines.
- Section 5.2 outlined and experimented with several new variants of the most well-known inverted file pruning algorithm by Carmel et al. [2001b], and stated how it is possible to tweak the technique so that the precision is improved when some information is removed selectively. As a main conclusion, discarding pointers from an inverted file off-line and independently of the queries, is able to devise satisfactory results, efficiency and effectiveness-wise, for ad-hoc retrieval. Also, carefully addressing the algorithmic issues involved brings some new intuitions regarding weighting schemes or term frequency normalisation.
- In section 5.3 we implemented several pruning techniques based on the *informativeness* and *discriminative value* of terms. Next, we evaluated the behaviour of precision with respect to

pruning, and the final effect in index file reduction and query processing times. Those methods have been compared with the well-known pruning method introduced by Carmel et al. [2001b]. In general, pruning whole terms is better for maintaining or improving MAP, and it keeps precision values at high pruning levels with long queries, whereas pruning pointers is better with respect to P@10. Therefore, methods that prune terms could be useful in applications such as indexing collections for PDAs and mobile devices, and desktop search.

- Section 5.4 presented a new application of the Probability Ranking Principle. The fundamental idea is to introduce this principle for static pruning and not for the estimation of relevance. Specifically, in this section we presented a novel index pruning algorithm based on the same assumptions as probabilistic retrieval models. Contrarily to other pruning approaches, the technique presented here does not aim at maintaining the top results returned by an IR system. Instead, it defines a pruning criterion that relies on the goodness of the estimation of three probabilities. The experimentation carried out comprised a thorough range of tests. In order to assess the performance of PRP-based pruning algorithm, we compared it to an enhanced version of Carmel et al.'s algorithm. The experiments conducted showed that the technique provides excellent outcomes: it outperforms the baseline in terms of MAP and performs at least as good in terms of P@10. Results are consistent across four different collections and they are assessed by 450 ad-hoc queries of two different lengths and 225 page finding queries. The conclusions drawn are valid for three different retrieval models. Furthermore, a set of experiments allowed us to conclude that the outcome of the pruning algorithm does not depend on a particular parameter tuning of the retrieval model.

- As a side result from the probabilistic static pruning, the need of the estimation of probability values led to the development of a new document prior based on a sigmoid function (section 5.4.2.2) and a way to estimate the non-relevance of a term without seeing a query (section 5.4.2.3). Hence, we proposed a novel application of the probability ranking principle, and new ways to estimate and combine the probabilities it implies.

- In chapter 6 we further investigated the issue of developing high-quality document priors and their effect in retrieval. We proposed a probabilistic document prior that takes into account term statistics and give a probabilistic derivation for it. The effect of the priors in retrieval is also dependent on the way they are combined with the query likelihood. Hence, we also demonstrated the effectiveness of a new way of combining document-length based priors with the query likelihood, that leverages the effect of the prior and likelihood components. The prior boosts the performance of a LM based on JM smoothing significantly, with robust and stable results across collections of different nature and topics of different sizes. The retrieval effectiveness of JM with the new prior is also able to outperform LM using Dirichlet prior smoothing and BM25, when the optimal parameters are used for all of them, and on the basis of three different effectiveness measures.

7.2 Future Research

This section suggests how parts of this thesis can be extended in the future.

7.2.1 Document Identifiers Assignment

Chapter 4 presented assignment of document identifiers as a mechanism of compressing an index without any loss of information. Different variants of this mechanism were presented, namely a graph-oriented solution based on the Travelling Salesman Problem (TSP), two alternative approaches based on Singular Value Decomposition (SVD), further clustering solutions, and finally a technique that combines TSP and clustering. Even though experimental evaluation showed these strategies can outperform other approaches and are in fact good reordering strategies, this does not mean that

these strategies necessarily give the optimal solution to the problem of reassignment of document identifiers. Better solutions can be developed by discovering heuristics that take into account the exact cost function to be minimised, just as in section 4.3. For a precise characterisation of this cost, see equation 4.12, on page 49. Such new heuristics will be based on the optimal cost function and/or the characterisation of the distance that a particular solution achieves with respect to the optimum. The same idea can also hold for TSP solutions: as long as this algorithm in the reduced dimension space performs well, we may pursue a formal characterisation to the distance of the optimal solution reached, with this sort of heuristic solutions.

In addition, it would be interesting to further combine TSP, SVD and clustering and measure the gains of applying the TSP in each cluster, instead to the whole collection. Moreover, this is especially indicated for collection partitioning techniques that may produce unbalanced clusters, such as the *k-means* algorithm (Jain et al. [1999]). The negative effect of large clusters would probably be softened with the SVD, so the TSP technique may run with acceptable times.

In any case it is necessary to experiment with efficient algorithms for computing SVD or other faster dimensionality reduction techniques more suitable for very large datasets.

A promising future line of research on the reassignment problem could follow very simple and fast ad-hoc solutions like the one by Silvestri [2007] that include more sophisticated clustering/document arranging mechanisms, and to study the query time and memory cache improvements of these sort of techniques with respect to traditional compression algorithms.

7.2.2 Static Pruning

Chapter 5 presented several index pruning techniques based on the informativeness and discriminative value of terms, and also based on the Probability Ranking Principle (PRP). Experimental evaluation showed that the proposed techniques are beneficial to retrieval mainly if high values of precision are desired, although they can be quite aggressive. A conclusion has been that pruning whole terms is better for maintaining or improving the mean average precision, whereas pruning pointers is better with respect to early precision (namely P@10). One future research line would be to design a pointer-based pruning method that operates selectively over posting lists, driven by a global term rank, instead of removing the whole posting lists at once.

Another topic of future research would be to address the problem of pruning while allowing for phrasal queries. None of the methods presented in this work is appropriate for processing phrasal queries. To tackle these problems it is necessary to develop an explicit pruning method for this purpose (de Moura et al. [2005]), combine a pruned inverted file with a next-word index (Bahle et al. [2002]), or develop two-tiered index schemes (Ntoulas and Cho [2007]).

In addition, experimental evaluation showed that the proposed index pruning techniques may improve retrieval precision up to certain pruning levels. This may indicate a weakness of the scoring function used for retrieval. It is possible to consider that a *perfect* matching function could leave the index as it is and prune it dynamically, but with high on-line query processing costs which may be more noticeable for pruning regimes that operate at a document by document fashion. This can be critical in high-demand environments, or in systems deployed on low-memory devices. Furthermore, this could open a future line of work for retrieval modeling inspired by the results in pruning performance.

Related to the above is also the issue of employing an index pruning technique that preserves the top-k results returned by the original unpruned index. This is also the case for the original implementation of Carmel et al's pruning method of the method, which shifts the scores of posting entries and that was presented in section 5.1, and also for the improved baseline presented in section 5.2.1.1. One can reason that if the estimation of the probabilities presented in section 5.4.2 were *ideal*, then the index pruning techniques would bring the optimum amount of pruning for a given index. Then, the top-k results would not be the same - they would be better, or, at least they would be as good as the original top-k.

With respect to the PRP-based index pruning technique presented in chapter 5, that approach follows a term-by-term mechanism, which needs collection statistics that can only be obtained after indexing. For that reason, the pruning algorithm would operate in two different phases. This is a

drawback with respect to the approach described in Büttcher and Clarke [2006] where this problem is solved by inspecting a sub-collection and extrapolating the statistics found to the whole collection, and hence pruning can be done while indexing. This could be a future line of research. Forthcoming work could try to dig deeper into the nature and estimation of the probabilities or try to incorporate term co-occurrence into the pruning process in a formal way and also assigning different non-constant costs per document in equation 5.18. For instance, pruning reputably home-pages would imply higher costs than other kinds of documents.

Additionally, as stated in section 5.4.4, it could be possible to alter the $p(q_i|\overline{r})$ estimation in such a way that it incorporates more refined stopword detection variants that could lead to even higher benefits to pruning.

Furthermore, another interesting future line of research is to design and perform suitable statistical test for comparing different pruning methods. Note that traditional significance tests employed in retrieval pose a number of problems in this scenario. First of all, pruning algorithms do not produce a specific pruning level: both operate with a threshold, and the exact amount of pruning cannot be quantisised in advance. This makes difficult the comparison at the same pruning level between different methods or collections; it would be necessary to perform a trial-and-error procedure until both methods produce an index with exactly the same amount of pruning. This would allow for comparing single pruning levels. For obtaining a single value over the whole pruning levels, a first step could be to interpolate the pruning level and compare the curves produced by the pruning methods on the basis of any variable. This could open the way for standard significance tests.

7.2.3 Document Priors

Chapter 6 presented two different document priors which take into account document length, and demonstrated the effectiveness of integrating them into retrieval. Experimental evaluation showed that integrating these priors into retrieval benefited retrieval performance. This is an encouraging observation, which could be further extended, for instance one could will try to address the problem of using this new developed way of considering document length into other retrieval matching functions, and other retrieval tasks. We also discussed briefly the *sigmoid* document prior used in section 5.4 for pruning. A future line of research would involve experimentation with this prior in ad-hoc retrieval. The sigmoid prior involves parameters that could be tuned specifically for different collections and therefore it would be necessary to assess their impact in retrieval performance.

Future work will try to examine how those priors affect the probability of being retrieved vs document length and probability of relevance pattern vs. document length patterns, which could help to understand the role of document length-based query independent evidence for language models.

APPENDIX

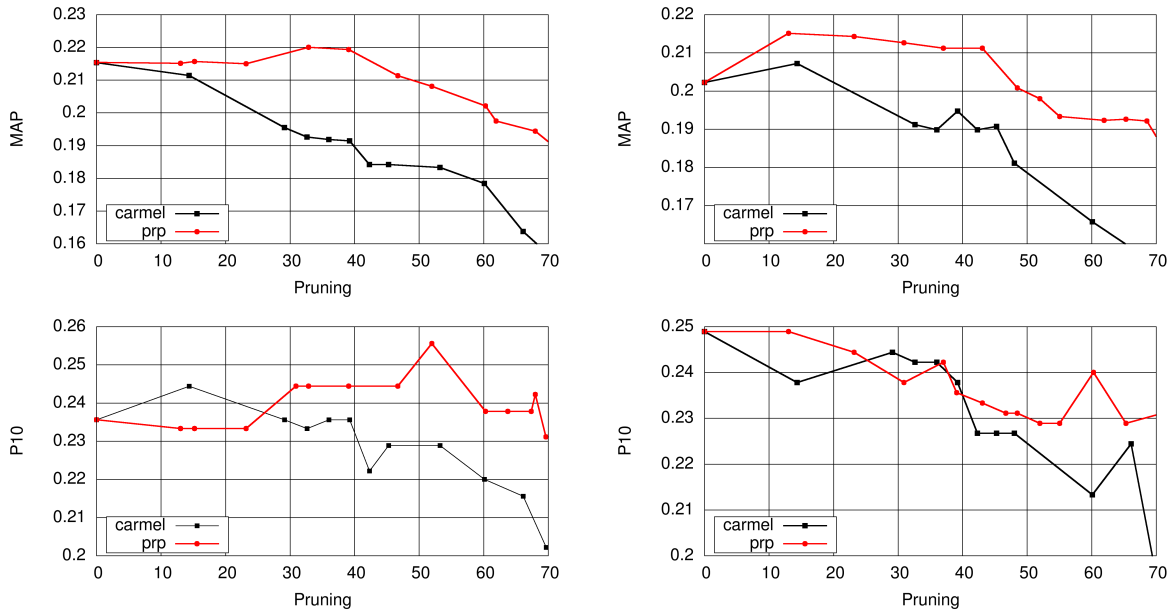


Figure 7.1: PRP vs Carmel pruning, short (left) and long (right) queries, LATimes, TF-IDF

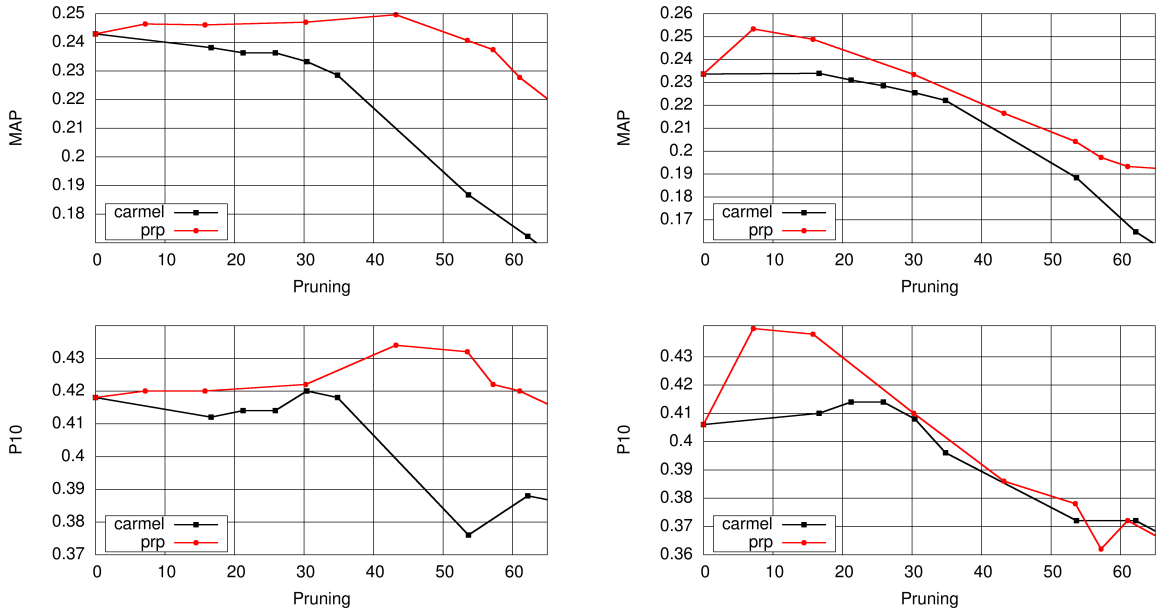


Figure 7.2: PRP vs Carmel pruning, short (left) and long (right) queries, WT2G, TF-IDF

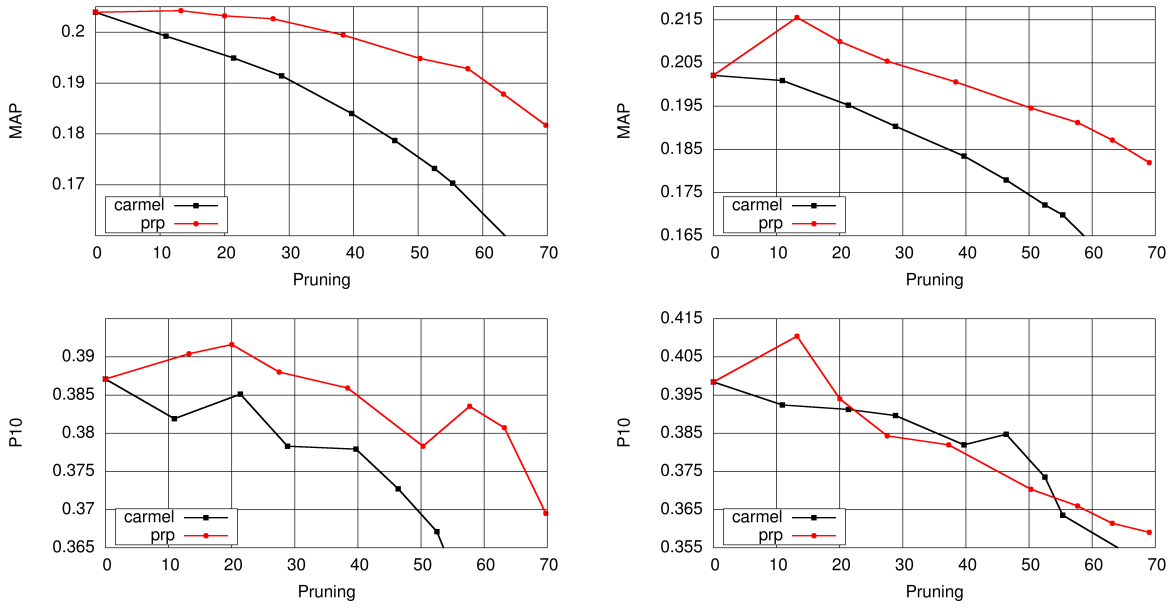


Figure 7.3: PRP vs Carmel pruning, short (left) and long (right) queries, Disks 4&5 , TF-IDF

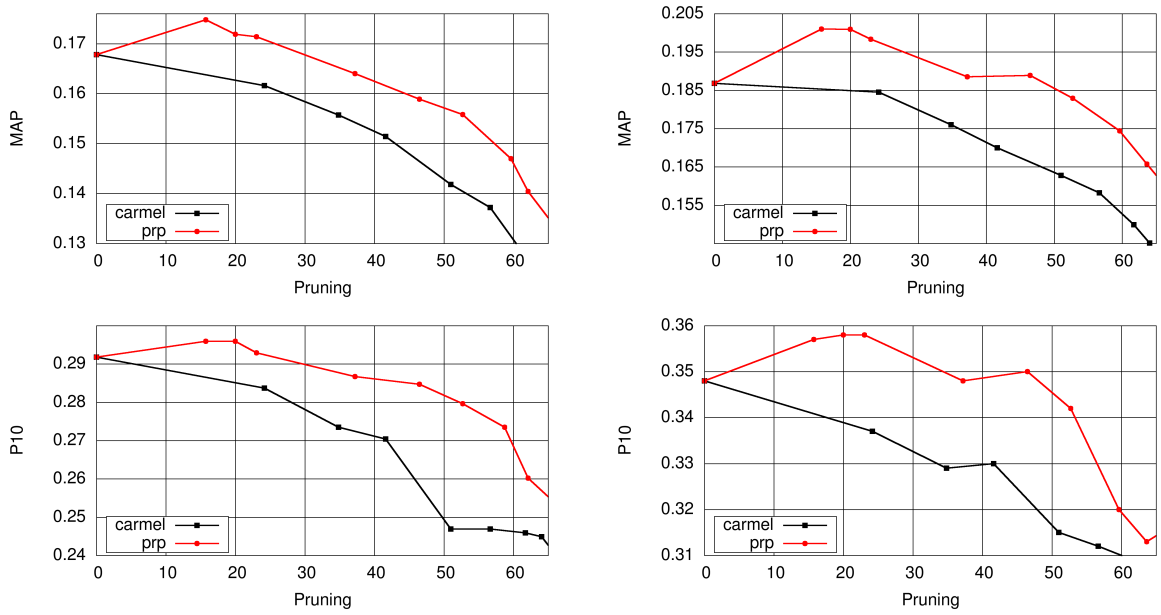


Figure 7.4: PRP vs Carmel pruning, short (left) and long (right) queries, WT10G, TF-IDF

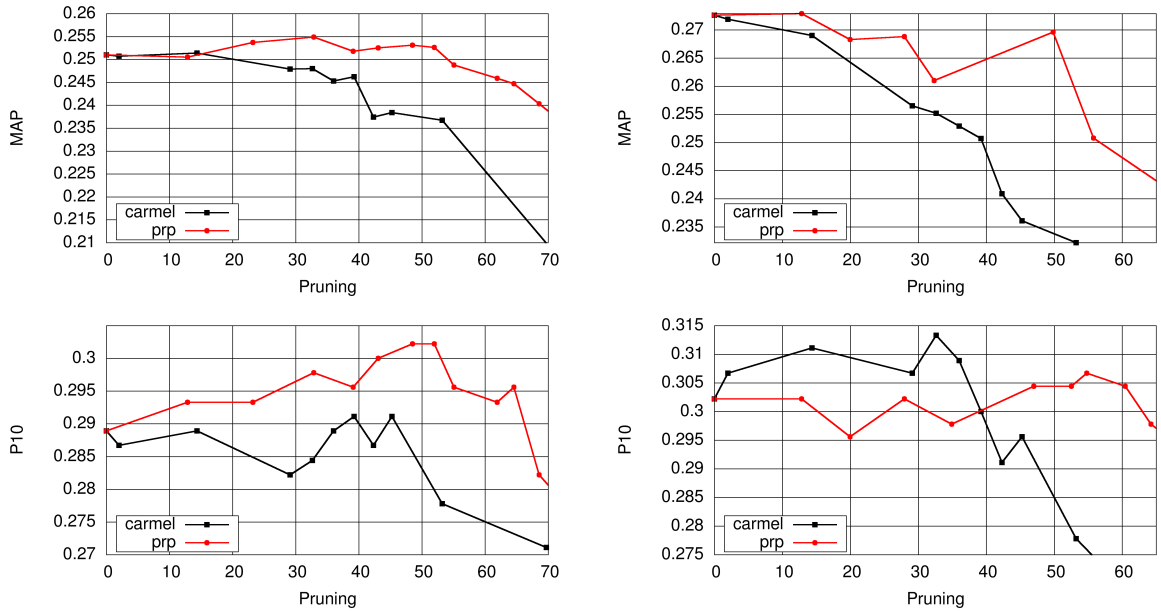


Figure 7.5: PRP vs Carmel pruning, BM25 retrieval with $b = 0.75$, short (left) and long (right) queries, LATimes.

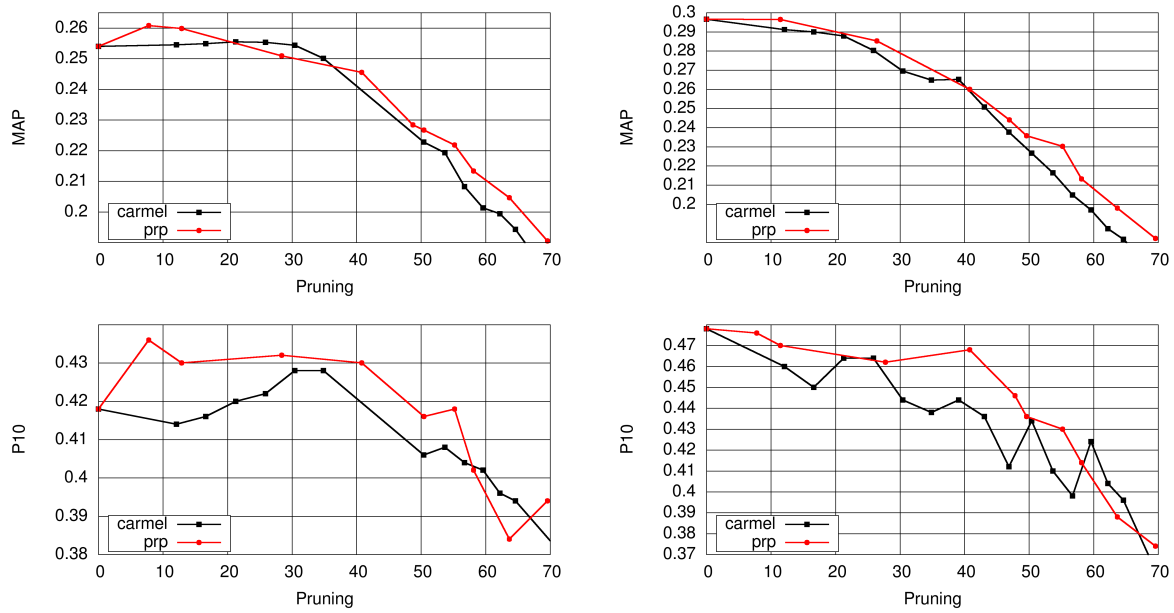


Figure 7.6: PRP vs Carmel pruning, BM25 retrieval with $b = 0.75$, short (left) and long (right) queries, WT2G.

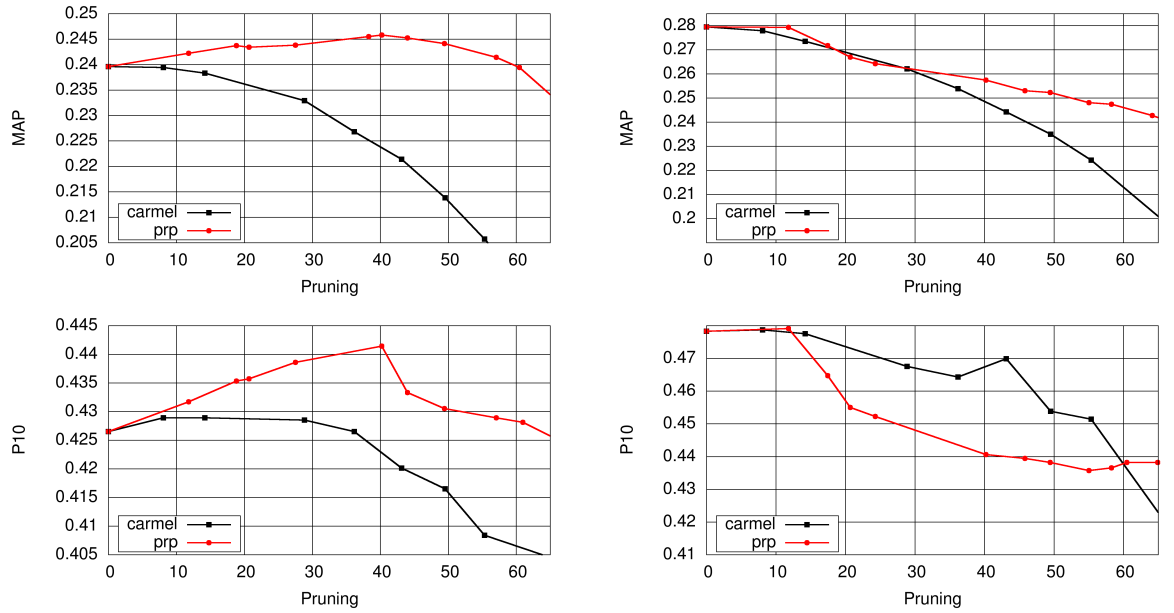


Figure 7.7: PRP vs Carmel pruning, BM25 retrieval with $b = 0.75$, short (left) and long (right) queries, Disks45.

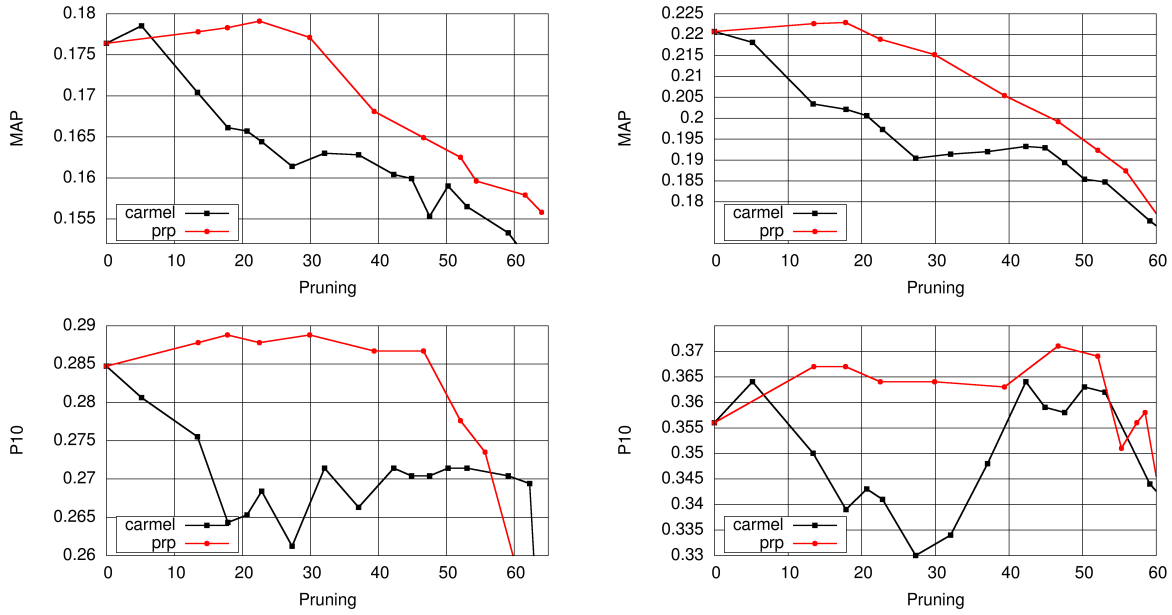


Figure 7.8: PRP vs Carmel pruning, BM25 retrieval with $b = 0.75$, short (left) and long (right) queries, WT10G.

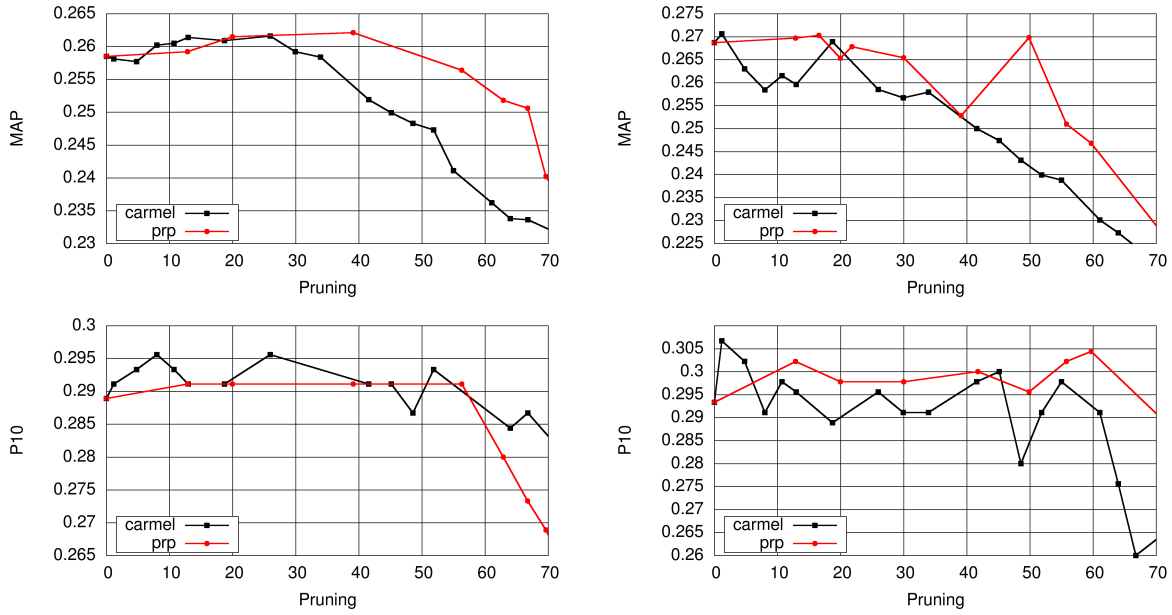


Figure 7.9: PRP vs Carmel pruning, BM25 retrieval with the best b, short (left) and long (right) queries, LATimes.

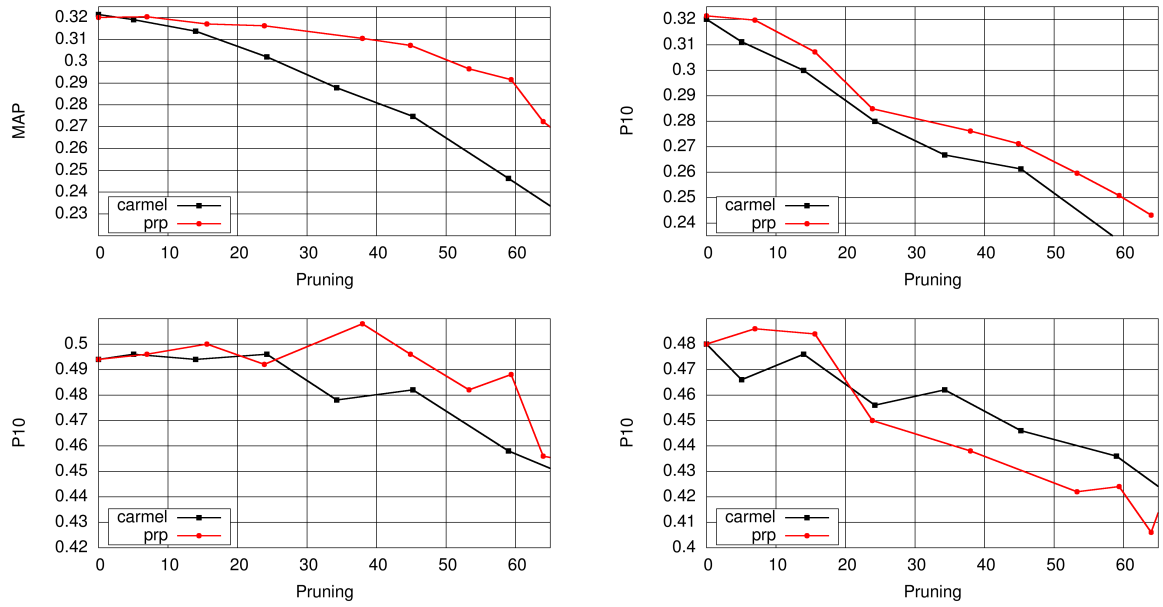


Figure 7.10: PRP vs Carmel pruning, BM25 retrieval with the best b, short (left) and long (right) queries, WT2G.

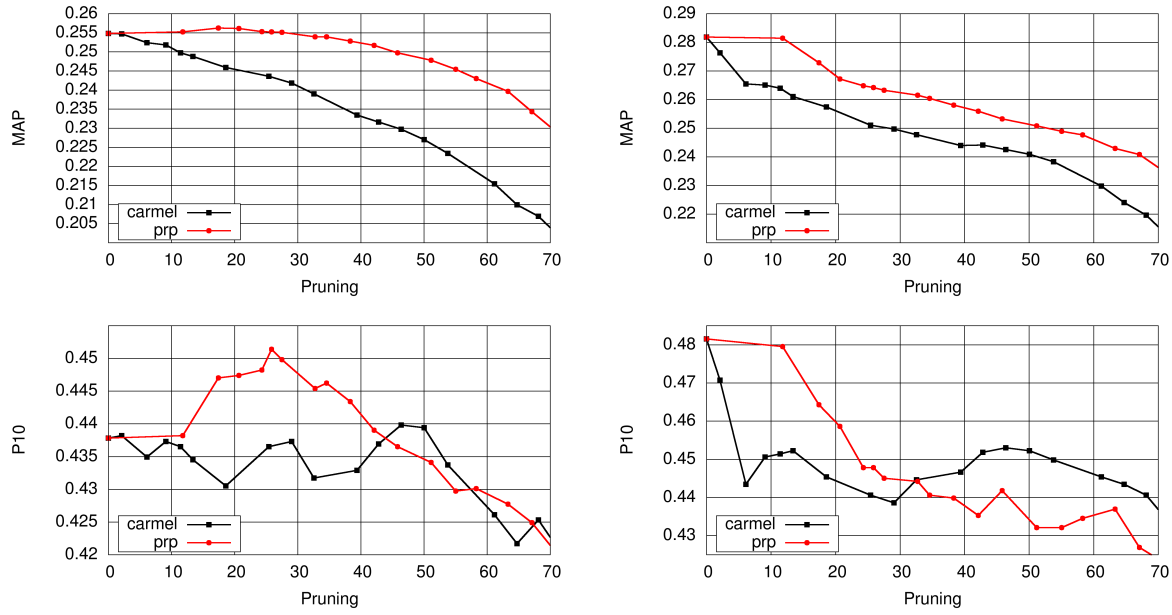


Figure 7.11: PRP vs Carmel pruning, BM25 retrieval with the best b, short (left) and long (right) queries, Disks45.

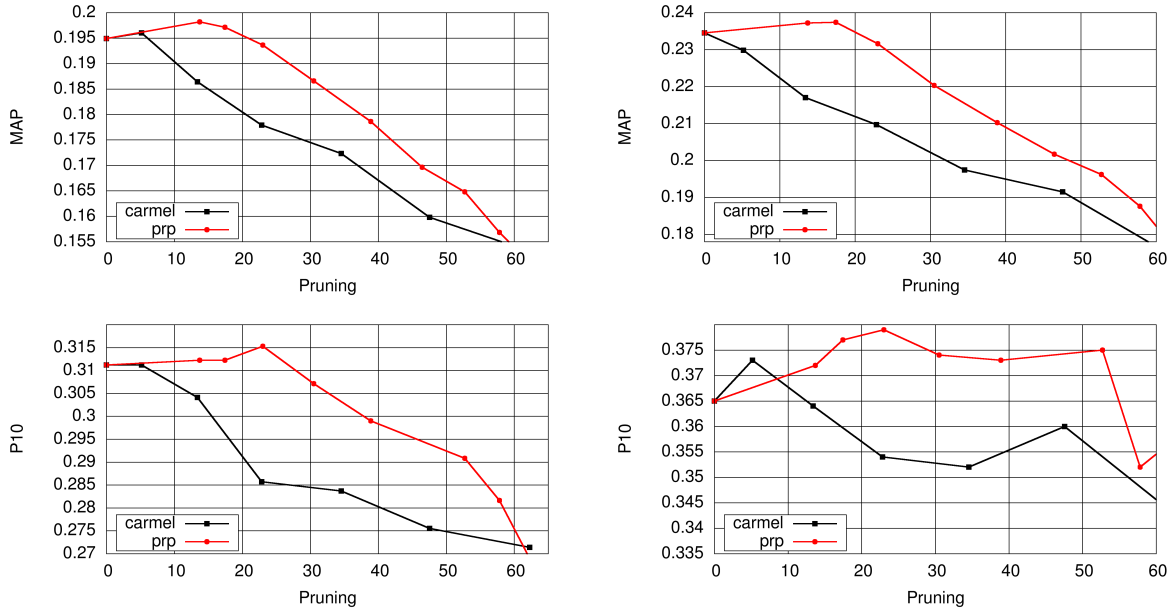


Figure 7.12: PRP vs Carmel pruning, BM25 retrieval with the best b, short (left) and long (right) queries, WT10G.

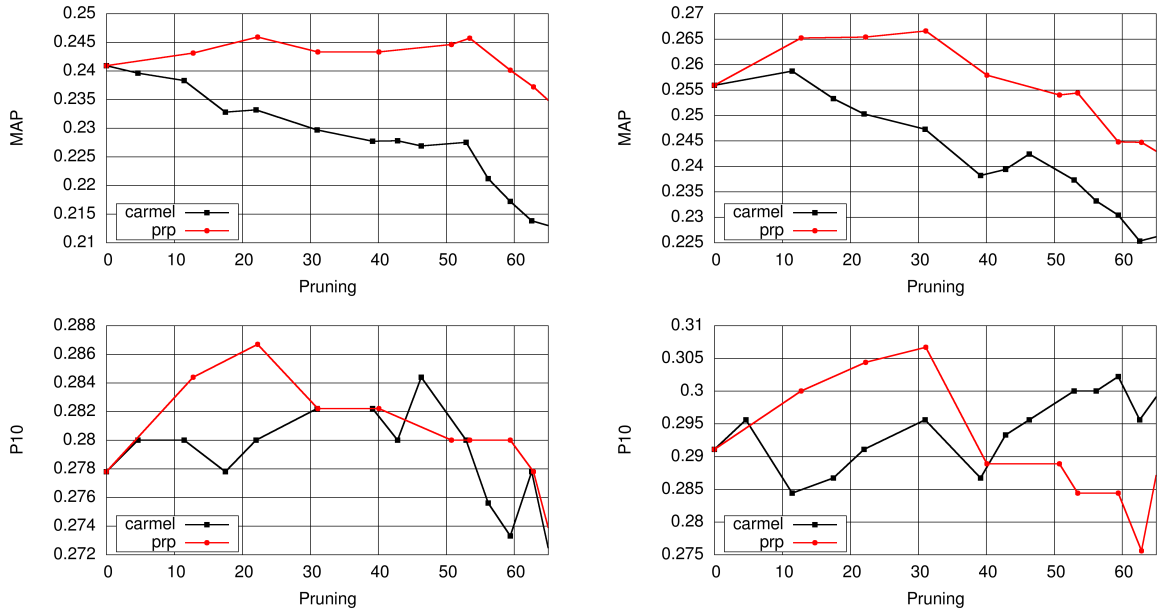


Figure 7.13: PRP vs Carmel pruning, parameter-free DLHH retrieval, short (left) and long (right) queries, LATimes.

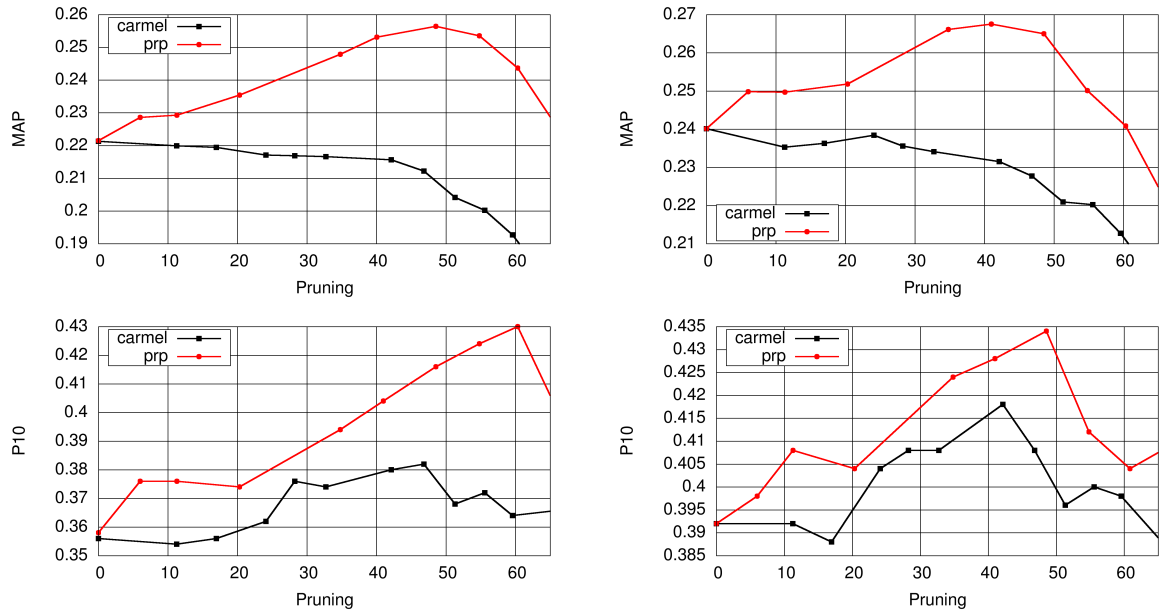


Figure 7.14: PRP vs Carmel pruning, parameter-free DLHH retrieval, short (left) and long (right) queries, WT2G.

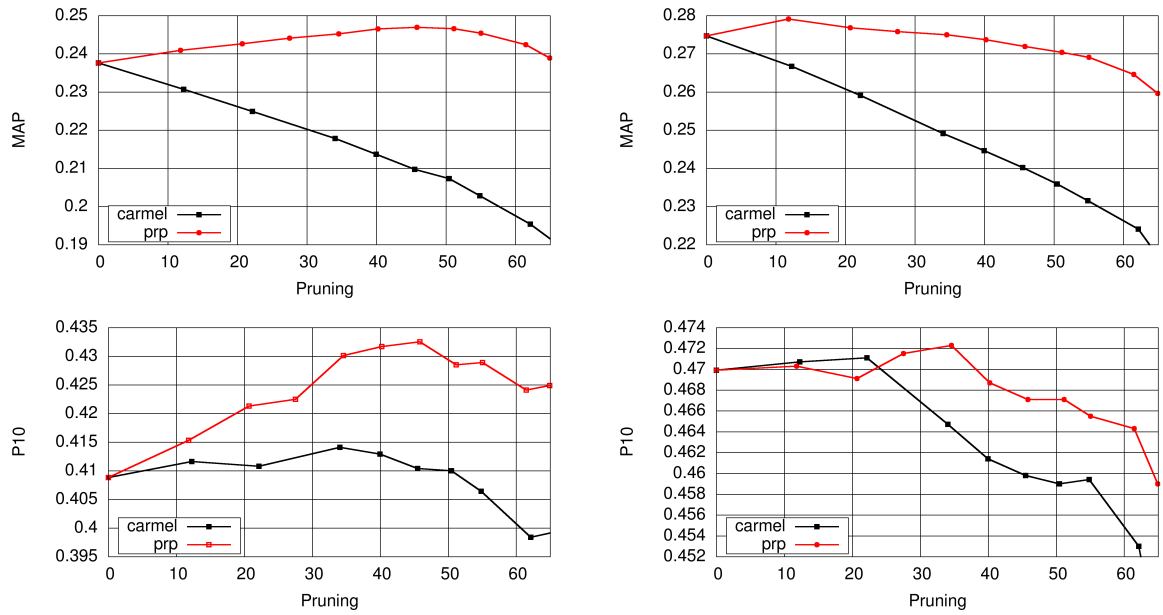


Figure 7.15: PRP vs Carmel pruning, parameter-free DLHH retrieval, short (left) and long (right) queries, Disks45.

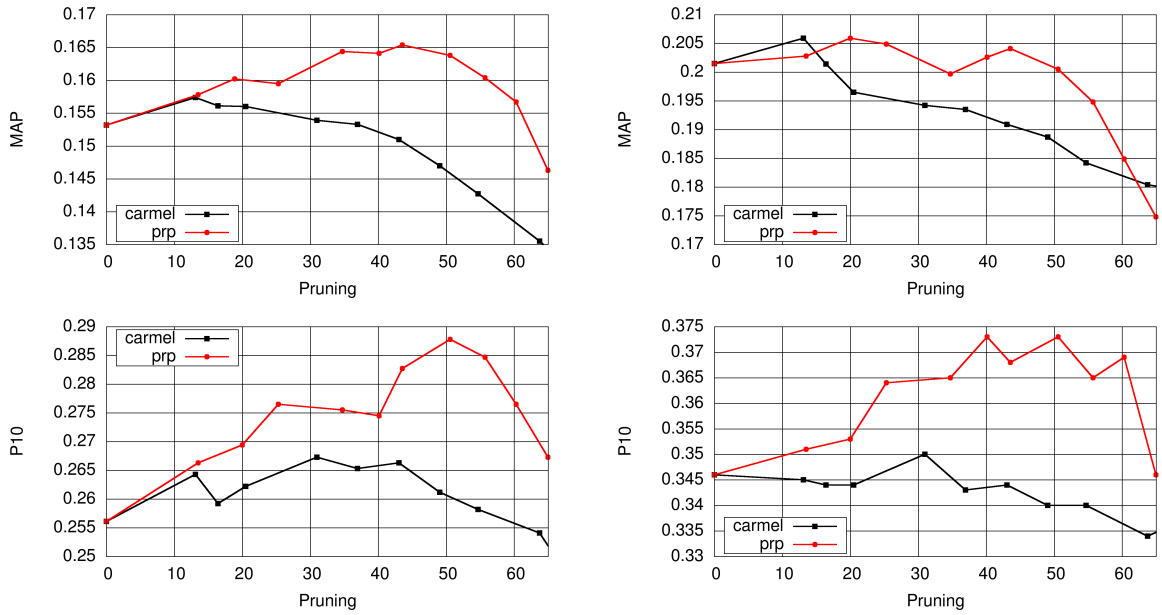


Figure 7.16: PRP vs Carmel pruning, parameter-free DLHH retrieval, short (left) and long (right) queries, WT10G.

Bibliography

G. Amati. Frequentist and bayesian approach to information retrieval. In *ECIR 2006: Proceedings of the 28th European Conference on IR Research*, volume 3936 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2006.

G. Amati and C. J. van Rijsbergen. Probabilistic models of information retrieval based on measuring divergence from randomness. *ACM Transactions on Information Systems*, 20(4):357–389, 2002.

V. Ngoc Anh and A. Moffat. Impact transformation: effective and efficient web retrieval. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–10, New York, NY, USA, 2002. ACM.

V. Ngoc Anh and A. Moffat. Pruned query evaluation using pre-computed impacts. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 372–379, New York, NY, USA, 2006. ACM.

R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. The impact of caching on search engines. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 183–190, New York, NY, USA, 2007. ACM.

R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM / Addison-Wesley, 1999. ISBN 0-201-39829-X.

D. Bahle, H. Williams, and J. Zobel. Efficient phrase querying with an auxiliary index. In *Proceedings of the 25th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 215–221, New York, NY, USA, August 2002.

B. T. Bartell, G. W. Cottrell, and R. K. Belew. Latent semantic indexing is an optimal special case of multidimensional scaling. In *SIGIR '92: Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 161–167, New York, NY, USA, 1992. ACM.

R. Blanco and A. Barreiro. Tsp and cluster-based solutions to the reassignment of document identifiers. *Information Retrieval*, 9(4):499–517, 2006.

R. Blanco and A. Barreiro. Document identifier reassignment through dimensionality reduction. In *ECIR 2005: Proceedings of the 27th European Conference on IR Research*, volume 3408 of *Lecture Notes in Computer Science*, pages 375–387. Springer, 2005a.

R. Blanco and A. Barreiro. Static pruning of terms in inverted files. In *ECIR 2007: Proceedings of the 29th European Conference on IR Research*, volume 4425 of *Lecture Notes in Computer Science*, pages 64–75. Springer, 2007a.

R. Blanco and A. Barreiro. Probabilistic document length priors for language models. In *ECIR 2008: Proceedings of the 30th European Conference on IR Research*, Lecture Notes in Computer Science, pages 394–405. Springer, 2008.

R. Blanco and A. Barreiro. Characterization of a simple case of the reassignment of document identifiers as a pattern sequencing problem. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 587–588, New York, NY, USA, 2005b. ACM.

R. Blanco and A. Barreiro. Boosting static pruning of inverted files. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 777–778, New York, NY, USA, 2007b. ACM.

D. Blandford and G. Blelloch. Index compression through document reordering. In *DCC '02: Proceedings of the Data Compression Conference*, pages 342–351, Washington, DC, USA, 2002. IEEE Computer Society.

P. Boldi and S. Vigna. Compressed perfect embedded skip lists for quick inverted-index lookups. In *SPIRE 2005 String Processing and Information Retrieval*, Lecture Notes in Computer Science, pages 25–28, 2005a.

P. Boldi and S. Vigna. Codes for the worldâĹŠwide web. *Internet Mathematics*, 2:405–427, 2005b.

A. Bookstein and D.R. Swanson. Probabilistic models for automatic indexing. *Journal of the American Society for Information Science*, 25:312–316, 1974.

A. Bookstein, S. T. Klein, and T. Raita. Clumping properties of content-bearing words. *Journal of the American Society of Information Science*, 49(2):102–114, 1998.

R. Brandow, K. Mitze, and L. Rau. Automatic condensation of electronic publications by sentence selection. *Information Processing and Management*, 31(5):675–685, 1995.

S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *WWW7: Proceedings of the seventh international conference on World Wide Web 7*, pages 107–117, Amsterdam, The Netherlands, 1998. Elsevier Science Publishers B. V.

A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*, pages 426–434, New York, NY, USA, 2003. ACM.

E. W. Brown. Fast evaluation of structured queries for information retrieval. In *SIGIR '95: Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 30–38, New York, NY, USA, 1995. ACM.

C. Buckley and A. F. Lewit. Optimization of inverted vector searches. In *SIGIR '85: Proceedings of the 8th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 97–110, New York, NY, USA, 1985. ACM.

C. Buckley and E. M. Voorhees. Retrieval evaluation with incomplete information. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 25–32, New York, NY, USA, 2004. ACM.

C. Buckley, A. Singhal, M. Mitra, and G. Salton. New retrieval approaches using smart: Trec. In *Proceedings of the Fourth Text REtrieval Conference (TREC-4)*, pages 25–48, November 1995.

S. Büttcher and C. L. A. Clarke. A document-centric approach to static index pruning in text retrieval systems. In *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 182–189, New York, NY, USA, 2006. ACM.

S. Büttcher, C. L. A. Clarke, and B. Lushman. Hybrid index maintenance for growing text collections. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 356–363, New York, NY, USA, 2006. ACM.

G. Candela and D. Harman. Retrieving records from a gigabyte of text on a mini-computer using statistical ranking. *Journal of the American Society for Information Science*, 41(8):581–589, 1990.

D. Carmel, E. Amitay, M. Herscovici, Y. S. Maarek, Y. Petruschka, and A. Soffer. Juru at trec 10 - experiments with index pruning. In *Proceedings of the Tenth Text REtrieval Conference (TREC 2001)*, pages 228–236, 2001a.

D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y. Maarek, and A. Soffer. Static index pruning for information retrieval systems. In *Proceedings of the 24th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–50, New York, NY, USA, 2001b. ACM.

B. Carterette and F. Can. Comparing inverted files and signature files for searching a large lexicon. *Information Processing and Management*, 41(3):613–633, 2005. ISSN 0306-4573.

C. Castillo. *Effective Web Crawling*. Ph.D. Thesis, University of Chile, 2004.

A. Chowdhury, M. C. MacCabe, D. Grossman, and O. Frieder. Document normalization revisited. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 381–382, New York, NY, USA, 2002. ACM.

K. Church and W. Gale. Poisson mixtures. *Natural Language Engineering*, 2(1):163–190, 1995.

N. Craswell and D. Hawking. Overview of the TREC-2004 Web Track. In *Proceedings of the Thirteenth Text REtrieval Conference (TREC 2004)*, November 2004.

N. Craswell, S. Robertson, H. Zaragoza, and M. Taylor. Relevance weighting for query independent evidence. In *Proceedings of ACM SIGIR 2005*, pages 416–423, New York, NY, USA, 2005. ACM.

W. B. Croft. Knowledge-based and statistical approaches to text retrieval. *IEEE Expert: Intelligent Systems and Their Applications*, 8(2):8–12, 1993. ISSN 0885-9000.

W.B. Croft and D.J. Harper. Using probabilistic models of document retrieval without relevance information. *Journal Of Documentation*, 35(4):285–295, 1979.

D. Cutting and J. Pedersen. Optimizations for dynamic inverted index maintenance. In *Proceedings of the 13th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 405–411, New York, NY, USA, 1990.

Z. J. Czech, G. Havas, and B. S. Majewski. An optimal algorithm for generating minimal perfect hash functions. *Information Processing Letters*, 43(5):257–264, 1992.

E. S. de Moura, C. F. dos Santos, D. R. Fernandes, A. S. Silva, P. Calado, and M. A. Nascimento. Improving web search efficiency via a locality based static pruning method. In *Proceedings of the 14th International Conference on World Wide Web*, pages 235–244, 2005.

J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

E. D. Demaine, A. López-Ortiz, and J. I. Munro. Experiments on adaptive set intersections for text retrieval systems. In *ALENEX '01: Revised Papers from the Third International Workshop on Algorithm Engineering and Experimentation*, pages 91–104, London, UK, 2001. Springer-Verlag.

J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34(3): 313–356, 2002.

S. T. Dumais. Latent semantic indexing (lsi): Trec-3 report, 1994.

P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2):194–203, 1975.

C. Faloutsos and S. Christodoulakis. Description and performance analysis of signature file methods for office filing. *ACM Transactions on Information Systems*, 5(3):237–257, 1987. ISSN 1046-8188.

C. Faloutsos and S. Christodoulakis. Signature files: an access method for documents and its analytical performance evaluation. *ACM Transactions on Information Systems*, 2(4):267–288, 1984. ISSN 1046-8188.

P. Ferragina and R. Venturini. Compressed permuterm index. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 535–542, New York, NY, USA, 2007. ACM.

A. Fink and S. Voss. Applications of modern heuristic search methods to pattern sequencing problems, 1998.

C. Fox. A stop list for general text. *SIGIR Forum*, 24(1-2):19–21, 1990.

E. Fox, S. Betrabet, M. Koushik, and W. Lee. Extended boolean models. In *Information retrieval: data structures and algorithms*, pages 393–418, Upper Saddle River, NJ, USA, 1992. Prentice-Hall, Inc.

E. A. Fox and W. C. Lee. Fast-inv: A fast algorithm for building large inverted files. Technical report, Virginia Polytechnic Institute & State University, Blacksburg, VA, USA, 1991.

R. Gallager and D. van Voorhis. Optimal source codes for geometrically distributed integer alphabets. *IEEE Transactions on Information Theory*, 21(2):228–230, 1975.

J. Goldstein, R. Ramakrishnan, and U. Shaft. Compressing relations and indexes. In *ICDE*, pages 370–379, 1998.

S. Golomb. Run-length encodings. *IEEE Transactions on Information Theory*, 12(1):399–401, 1966.

R. Grossi, A. Gupta, and J. Vitter. High-order entropy-compressed text indexes. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03)*, pages 841–850, 1993.

S. P. Harter. A probabilistic approach to automatic keyword indexing. *Journal of the American Society for Information Science*, 26:197–206, 1975.

B. He and I. Ounis. A study of parameter tuning for term frequency normalization. In *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*, pages 10–16, New York, NY, USA, 2003. ACM.

S. Heinz and J. Zobel. Efficient single-pass index construction for text databases, 2003.

D. Hiemstra. *Using language models for information retrieval*. Ph.D. Thesis, Centre for Telematics and Information Technoloy, 2001.

D. Huffman. A method for the construction of minimum redundancy codes. *Proceedings of the IRE*, 40:1098–1101, 1952.

A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3): 264–323, 1999.

G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report TR 95-035, Department of Computer Science, University of Minnesota, 1995.

E. Kokiopoulou and Y. Saad. Polynomial filtering in latent semantic indexing for information retrieval. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 104–111, New York, NY, USA, 2004. ACM.

W. Kraaij, T. Westerveld, and D. Hiemstra. The importance of prior probabilities for entry page search. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 27–34, New York, NY, USA, 2002. ACM.

J. Lafferty and C. Zhai. Probabilistic relevance models based on document and query generation. In W.B. Croft and J. Lafferty, editors, *Language Modeling for Information Retrieval*, pages 11–56. Kluwer, 2003.

V. Lavrenko and W. B. Croft. Relevance based language models. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 120–127, New York, NY, USA, 2001. ACM.

N. Lester, J. Zobel, and H. E. Williams. In-place versus re-build versus re-merge: index maintenance strategies for text retrieval systems. In *ACSC '04: Proceedings of the 27th Australasian conference on Computer science*, pages 15–23, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

N. Lester, A. Moffat, W. Webber, and J. Zobel. Space-limited ranked query evaluation using adaptive pruning. In *Proceedings of the WISE Workshop on Information Systems Engineering*, pages 470–482, New York, NY, USA, nov 2005a.

N. Lester, A. Moffat, and J. Zobel. Fast on-line index construction by geometric partitioning. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 776–783, New York, NY, USA, 2005b. ACM.

R.T.W. Lo, B. He, and I. Ounis. Automatically building a stopword list for an information retrieval system. In *Proceedings of the 5th Dutch-Belgian Information Retrieval Workshop (DIR'05)*, Utrecht, Netherlands, 2005.

U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. In *SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 319–327. Society for Industrial and Applied Mathematics, 1990.

R. Mandala, T. Tokunaga, and H. Tanaka. Combining multiple evidence from different types of thesaurus for query expansion. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 191–197, New York, NY, USA, 1999. ACM.

R. Manmatha, T. Rath, and F. Feng. Modeling score distributions for combining the outputs of search engines. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 267–275, New York, NY, USA, 2001. ACM.

C. D. Manning, P. Raghavan, and H. Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

E. L. Margulis. N-poisson document modelling. In *Proceedings of the 15th International ACM SIGIR Conference on Research and development in information retrieval*, pages 177–189, New York, NY, USA, 1992. ACM.

S. Melnik, S. Raghavan, B. Yang, and H. Garcia-Molina. Building a distributed full-text index for the web. *ACM Transactions on Information Systems*, 19(3):217–241, 2001. ISSN 1046-8188.

D. Metzler and W. B. Croft. Latent concept expansion using markov random fields. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 311–318, New York, NY, USA, 2007. ACM.

A. Moffat. Word-based text compression. *Software Practice Experertise*, 19(2):185–198, 1989. ISSN 0038-0644.

A. Moffat and V. N. Anh. Inverted index compression using word-aligned binary codes. *Information Retrieval*, 8:151–166, 2005.

A. Moffat and T. A. H. Bell. In situ generation of compressed inverted files. *Journal for the American Society Information Science*, 46(7):537–550, 1995. ISSN 0002-8231.

A. Moffat and L. Stuiver. Binary interpolative coding for effective index compression. *Information Retrieval*, 3(1):25–47, 2000. ISSN 1386-4564.

A. Moffat and J. Zobel. Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems*, 14(4):349–379, 1996.

A. Ntoulas and J. Cho. Pruning policies for two-tiered inverted index with correctness guarantee. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 191–198, New York, NY, USA, 2007. ACM.

R. Pagh and F. F. Rodler. Cuckoo hashing. *Lecture Notes in Computer Science*, 2161:121–47, 2001.

V. Plachouras and I. Ounis. Multinomial randomness models for retrieval with document fields. In *ECIR 2007: Proceedings of the 29th European Conference on Information Retrieval*, pages 28–39. Springer, 2007.

J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–281, New York, NY, USA, 1998. ACM.

M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, July 1980.

J. D. M. Rennie and T. Jaakkola. Using term informativeness for named entity detection. In *Proceedings of the 28th International ACM SIGIR Conference on Research and development in information retrieval*, pages 353–360, New York, NY, USA, 2005. ACM.

B. Ribeiro-Neto, E. S. Moura, M. S. Neubert, and N. Ziviani. Efficient distributed algorithms to build inverted files. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 105–112, New York, NY, USA, 1999. ACM.

B. A. Ribeiro-Neto and R. A. Barbosa. Query performance for tightly coupled distributed digital libraries. In *DL '98: Proceedings of the third ACM conference on Digital libraries*, pages 182–190, New York, NY, USA, 1998. ACM.

R. F. Rice. Some practical universal noiseless coding techniques. Technical Report 79–22, Jet Propulsion Laboratory, Pasadena, California, JPL Publication, 1979.

J. Langdon G. G. Rissanen. Arithmetic coding. *IBM Journal of Research and Development*, 23:149–162, 1979.

R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321 (Informational), April 1992. URL `http://www.ietf.org/rfc/rfc1321.txt`.

S. Robertson. The probability ranking principle in ir. *Journal of Documentation*, 33:294–304, 1977.

S. Robertson and K. Sparck-Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27:129–146, 1976a.

S. Robertson and K. Sparck-Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27:129–146, 1976b.

S. Robertson and S. Walker. Okapi/Keenbow at TREC-8. In *Proceedings of the Eigth Text REtrieval Conference (TREC-8)*, pages 151–162, 2000.

S. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 232–241, New York, NY, USA, 1994. ACM/Springer-Verlag.

S. Robertson, C. J. van Rijsbergen, and M. F. Porter. Probabilistic models of indexing and searching. In *SIGIR '80: Proceedings of the 3rd annual ACM conference on Research and development in information retrieval*, pages 35–56, Kent, UK, 1981. Butterworth & Co.

S. Robertson, H. Zaragoza, and M. Taylor. Simple bm25 extension to multiple weighted fields. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 42–49, New York, NY, USA, 2004. ACM.

S. E. Robertson, S. Walker, M. Hancock-Beaulieu, A. Gull, and M. Lau. Okapi at TREC-4. In *Proceedings of the Fourth Text REtrieval Conference (TREC-4)*, pages 73–86, 1995.

J. J. Rocchio. Relevance feedback in information retrieval. In *The SMART Retrieval System: Experiments in Automatic Document Processing*, pages 313–323. G. Salton, editor. Prentice-Hall, Englewood Cliffs, NJ, USA, 1971.

T. Sakai and K. Sparck-Jones. Generic summaries for indexing in information retrieval. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 190–198, New York, NY, USA, 2001. ACM.

G. Salton and C. Buckley. Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science*, 41(4):288–297, 1980.

G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975a. ISSN 0001-0782.

G. Salton, C. S. Yang, and C. T. Yu. A theory of term importance in automatic text analysis. *JASIS*, 26 (1):33–44, 1975b.

M. Sanderson and J. Zobel. Information retrieval system evaluation: effort, sensitivity, and reliability. In *SIGIR '05: Proceedings of the 28th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 162–169, New York, NY, USA, 2005. ACM.

F. Scholer, H. Williams, J. Yiannis, and J. Zobel. Compression of inverted indexes for fast query evaluation, 2002.

C. E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–423, 1948.

W.-Y. Shieh and C.-P. Chung. A statistics-based approach to incrementally update inverted files. *Information Processing and Management*, 41(2):275–288, 2005. ISSN 0306-4573.

W.-Y. Shieh, T.-F. Chen, J. J.-J. Shann, and C.-P. Chung. Inverted file compression through document identifier reassignment. *Information Processing and Management*, 39(1):117–131, 2003. ISSN 0306-4573.

F. Silvestri. Sorting out the document identifier assignment problem. In *ECIR 2007: Proceedings of the 29th European Conference on IR Research*, volume 4425 of *Lecture Notes in Computer Science*, pages 101–112. Springer, 2007.

F. Silvestri, S. Orlando, and R. Perego. Assigning identifiers to documents to enhance the clustering property of fulltext indexes. In *Proceedings of the 27th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 305–312, New York, NY, USA, 2004.

A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In *SIGIR '96: Proceedings of the 19st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 21–29, New York, NY, USA, 1996. ACM.

K. Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972.

K. Sparck-Jones and C. J. van Rijsbergen. Information retrieval test collections. *Journal of Documentation*, 32:59–75, 1976.

T. Strohman and W. B. Croft. Efficient document retrieval in main memory. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 175–182, New York, NY, USA, 2007. ACM.

T. Strohman, H. Turtle, and W. B. Croft. Optimization strategies for complex queries. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 219–225, New York, NY, USA, 2005. ACM.

M. Theobald, G. Weikum, and R. Schenkel. Top-k query evaluation with probabilistic guarantees. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 648–659. VLDB Endowment, 2004.

A. Tomasic, H. García-Molina, and K. Shoens. Incremental updates of inverted lists for text document retrieval. In *SIGMOD '94: Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 289–300, New York, NY, USA, 1994. ACM.

H. Turtle and J. Flood. Query evaluation: strategies and optimizations. *Information Processing and Management*, 31(6):831–850, 1995.

T. Upstill, N. Craswell, and D. Hawking. Query-independent evidence in home page finding. *ACM Transactions on Information Systems*, 21(3):286–313, 2003.

C. J. van Rijsbergen. A theoretical basis for the use of co-occurrence data in information retrieval. *Journal of Documentation*, 33:106–119, 1977.

C. J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 1979.

E. M. Voorhees and D. K. Harman. *TREC: Experiment and Evaluation in Information Retrieval*. The MIT Press, 2005.

T. Westerveld, W. Kraaij, and D. Hiemstra. Retrieving web pages using content, links, urls and anchors. In *Proceedings of the tenth Text Retrieval Conference (TREC-10)*, pages 663–672, 2002.

I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987. ISSN 0001-0782.

I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, San Francisco, CA, 1999. ISBN 1-55860-570-3.

J. Xu and W. B. Croft. Query expansion using local and global document analysis. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 4–11, New York, NY, USA, 1996. ACM.

Jinxi Xu and W. Bruce Croft. Improving the effectiveness of information retrieval with local context analysis. *ACM Transactions on Information Systems*, 18(1):79–112, 2000. ISSN 1046-8188.

C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems*, 22(2):179–214, 2004. ISSN 1046-8188.

J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38:1–56, 2006.

J. Zobel, S. Heinz, and H. E. Williams. In-memory hash tables for accumulating text vocabularies. *Information Processing Letters*, 80(6):271–277, 2001.

M. Zukowski, S. Heman, N. Nes, and P. Boncz. Super-scalar ram-cpu cache compression. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, pages 59–71. IEEE Computer Society, 2006.