A Comparison of Open Source Search Engines

Christian Middleton, Ricardo Baeza-Yates

Contents

Chapter 1

Introduction

As the amount of information available on the websites increases, it becomes necessary to give the user the possibility to perform searches over this information. When deciding to install a search engine in a website, there exists the possibility to use a commercial search engine or an open source one. For most of the websites, using a commercial search engine is not a feasible alternative because of the fees that are required and because they focus on large scale sites. On the other hand, open source search engines may give the same functionalities (some are capable of managing large amount of data) as a commercial one, with the benefits of the open source philosophy: no cost, software maintained actively, possibility to customize the code in order to satisfy personal needs, etc.

Nowadays, there are many open source alternatives that can be used, and each of them have different characteristics that must be taken into consideration in order to determine which one to install in the website. These search engines can be classified according to the programming language in which it is implemented, how it stores the index (inverted file, database, other file structure), its searching capabilities (boolean operators, fuzzy search, use of stemming, etc), way of ranking, type of files capable of indexing (HTML, PDF, plain text, etc), possibility of on-line indexing and/or making incre-

mental indexes. Other important factors to consider are the last date of update of the software, the current version and the activity of the project. These factors are important since a search engine that has not been updated recently, may present problems at the moment of customizing it to the necessities of the current website. These characteristics are useful to make a broad classification of the search engines and be capable of narrowing the available spectrum of alternatives. Afterward, it is important to consider the performance of these search engines with different loads of data and also analyze how it degrades when the amount of information increases. In this stage, it is possible to analyze the indexing time versus the amount of data, as well as the amount of resources used during the indexing, and also analyze the performance during the retrieval stage.

The present work is the first study, to the best of our knowledge, to cover a comparison of the main features of 17 search engines, as well as a comparison of the performance during the indexing and retrieval tasks with different document collections and several types of queries. The objective of this work is to be used as a reference for deciding which open source search engine fits best with the particular constraints of the search problem to be solved.

On chapter 2 we prefer a background of the general concepts of Information Retrieval. On chapter 3 it is presented a description of the search engines used in this work. Then, on chapter 4 the methodology used during the experiments is described. On chapters 5.1 and 5.2 we present the results of the different experiments conducted, and on chapter 5.3 the analysis of these results. Finally, on chapter 6 the conclusions are presented.

Chapter 2

Background

Information Retrieval (IR) is a very broad field of study and it can be characterized as a field that:

> "... deals with the representation, storage, organization of, and access to information items."[1]

As a general field, it must be able to manipulate the information in order to allow the user to access it efficiently, focusing on the user information need. Another definition, without loss of generality, can be stated as:

> "Information retrieval is finding material (usually documents) of an unstructured nature (usually text) that satisfy an information need from within large collections (usually on local computer servers or on Internet)"[17]

The main idea is to satisfy the user information need by searching over the available material for information that seems relevant. In order to accomplish this, the IR system consists on several modules that interact among them (see Figure 2.1). It can be described, in a general form, as three main areas: *Indexing*, *Searching*, and *Ranking*:

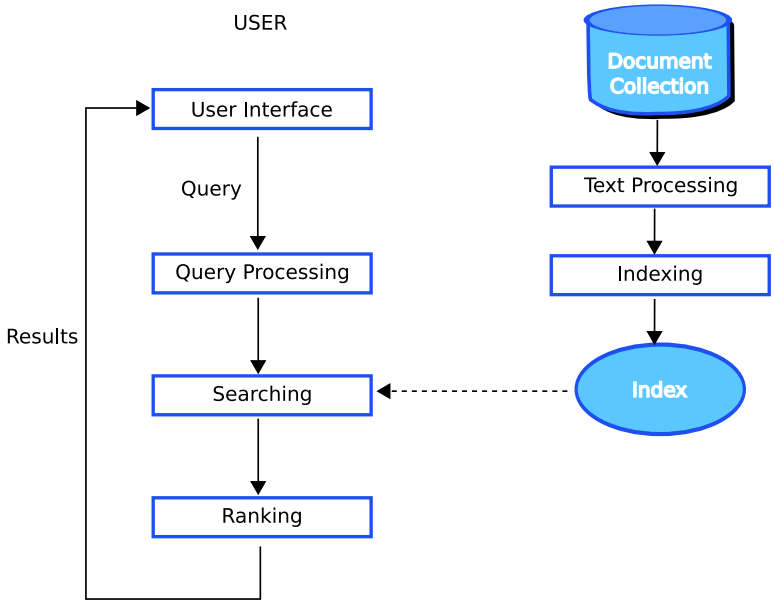


Figure 2.1: Information Retrieval process

Indexing In charge of the representation and organization of the material, allowing rapid access to the information.

Searching In charge of extracting information from the index that satisfies the user information need.

Ranking Although this is an optional task, it is also very important for the retrieval task. It is in charge of sorting the results, based on heuristics that try to determine which results satisfy better the user need.

2.1 Document Collection

In order to have information where to search for, it is necessary to collect it and use it as input for the Indexing stage. A document collection can be any type of source of data, that can be used to extract information. There can be several scenarios, depending on the application of the IR system.

2.1.1 Web Crawling

In the scenario of Web search, it is necessary to use a *crawler* that, basically, navigates through the Web and downloads the pages it access. There are several crawlers available, some with commercial licenses, and others available with open-source licenses. Since the Web has become immense, the crawlers may differ on the algorithm used to select the pages to crawl in order to leverage adding new pages to the collection and updating existent ones. Also, they must consider the bandwidth usage, in order not to saturate the crawled sites.

2.1.2 TREC

Other document collections have been generated, some of them for academic analysis. For example, in the *Text REtrieval Conference*[13] (TREC), they have created several document collections with different sizes and different types of documents, specially designed for particular tasks. The tasks are divided into several *tracks* that characterize the objective of the study of that collection. For example, some of the seven 2007 TREC Tracks are:

- Blog Track: Their objective is to explore information seeking behavior in the blogosphere.
- Enterprise Track: Analyze enterprise search, i.e., fulfill the information need of a user searching data of an organization to complete some task.
- Genomics Track: Study retrieval tasks in a specific domain (genomics data).
- Spam Track: Analyze current and proposed spam filtering approaches.

Besides the document collection, TREC is used as an instance for discussing and comparing different retrieval approaches by analyzing the results of the different groups and the approaches used. For this purpose, they provide a set of retrieval tasks and the corresponding query relevance judgment,

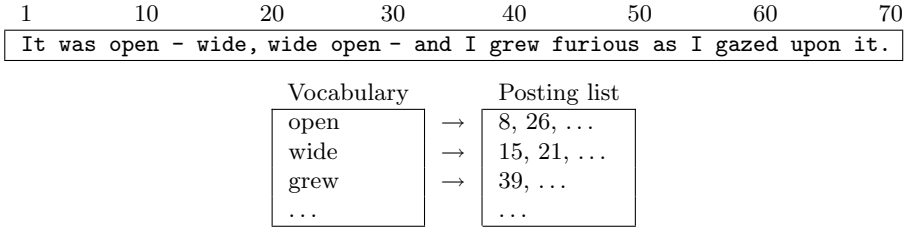


Table 2.1: Example of inverted index based on a sample text. For every word, the list of occurrences is stored.

so it is possible to analyze the precision and recall of the different IR systems, in different scenarios.

2.2 Indexing

In order to be able of making efficient searches over the document collection, it is necessary to have the data stored in specially designed data structures. These data structures are the *indices*, and permits to make fast searches over the collection, basically, by decreasing the number of comparisons needed. One of the most used data structure used on text retrieval is the *inverted index* (see Table 2.1). It consists on a *vocabulary*, that contains all the words in the collection, and a *posting list* that, for each term in the vocabulary, gives the list of all the positions where that word appears in the collection. Depending on the application, and type of matching that is required, some implementations store a list of documents instead, but the concept of index remains.

The space required to store the index is proportional to the size of the document collection, and there exists some techniques for reducing or optimizing the amount of space required. In general, the space used by the vocabulary is small, but the space used by the posting list is much more significant. Also, the space required depends on the functionalities offered by the search engine, so there is a trade-off between the space required and the

functionalities offered. For example, some indexers store the full text of the collection, in order to present the user with a sample of the text ("snippet") surrounding the search, while others use less space, but are not able to give a snippet. Other indexers use techniques for reducing the size of the posting list (e.g., using *block addressing*, where the text is divided into blocks so the posting list points to the blocks, grouping several instances into fewer blocks), but their trade-off is that for obtaining the exact position of a word the engine might need to do extra work (in our case, it is necessary to do a sequential scan over the desired block).

There are several pre-processing steps that can be performed over the text during the indexing stage. Some of the most commonly used are *stopword elimination* and *stemming*.

There are some terms that appear very frequently on the collection, and are not relevant for the retrieval task (for example, in English, the words "a", "an", "are", "be", "for", ...), and they are referred as *stopwords*. Depending on the application and language of the collection, the list of words can vary. A common practice, called stopword elimination, is to remove these words from the text and do not index them, making the inverted index much smaller.

Another technique is stemming, since it is common that, besides the exact term queried by the user, there are some variations of the word that also appear on the text. For example, the plural form and past tense of the word might also be used as a match. To address this problem, some indexers use an algorithm that obtain the *stem* of a word, and querying this word instead. The stem of a word is the portion of the word that is left after the removal of the affixes [1]. An example is the word "connect" that is the stem of the words "connected", "connecting", "connection", and "connections".

All the pre-processing and the way of storing the inverted index affect the space required as well as the time used for indexing a collection. As mentioned before, it depends on the application, it might be convenient to

trade-off time needed to build the index in order to obtain a more space-efficient index. Also, the characteristics of the index will affect the searching tasks, that will be explained on the following section.

2.3 Searching and Ranking

Based on an inverted index, it is possible to perform queries very efficiently. Basically, the main steps in the retrieval task are:

1. Vocabulary Search: The query is splitted into words (terms), and searched over the vocabulary of the index. This step can be achieved very efficient, by having the vocabulary sorted.

2. Retrieval of Occurrences: All the posting lists, of the terms found on the vocabulary, are retrieved.

3. Manipulation of Occurrences: The lists must be manipulated in order to obtain the results of the query.

Depending on the type of query (boolean, proximity, use of wildcards, etc), the manipulation differ and might imply some additional processing of the results. For example, *boolean* queries are the most commonly used, and consist on a set of terms (*atoms*) that are combined using a set of *operators* (as "and", "or", "not") in order to retrieve documents that match these conditions. These kind of queries are very simple to solve using an inverted file, since the only manipulation required is to merge the posting lists and selecting only the ones that satisfy the conditions. On the other hand, *phrase* and *proximity* queries are more difficult to solve, since they require a complex manipulation of the occurrences. Phrase queries refers to queries that search for a set of words that appear in a particular pattern, while proximity queries is a more relaxed version where the words might be at a certain distance, but still satisfying the order of the words. For these type of queries, it is necessary to have the list of occurrences ordered by

the word position, and perform a pattern matching over the resulting list, making the retrieval more complicated than simple boolean queries.

After performing the search over the index, it might be necessary to *rank* the results obtained in order to satisfy the user need. This stage of ranking might be optional, depending on the application, but for the Web search scenario it has become very important. The process of ranking must take into consideration several additional factors, besides whether the list of documents satisfy the query or not. For example, in some applications, the size of the retrieved document might indicate a level of importance of the document; on the Web scenario, another factor might be the "popularity" of the retrieved page (e.g. a combination of the number of in- and out-links, age of the page, etc); the location of the queried terms (e.g. if they appear on the body or in the title of the document); etc.

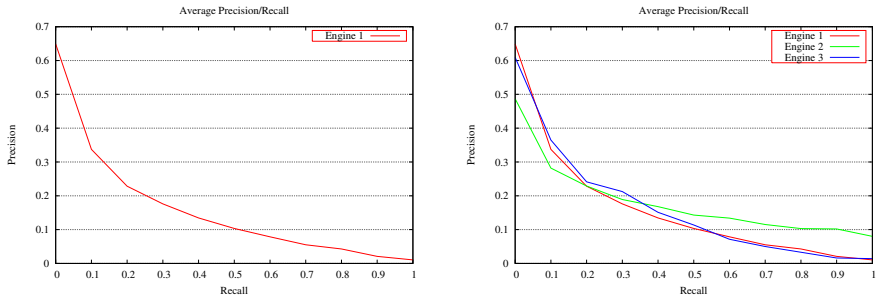
2.4 Retrieval Evaluation

For analyzing the "quality" of a retrieval system, it is possible to study if the results returned by a certain query are related to it or not. This can be done by determining, given a query and a set of documents, the ones that are related (i.e., are *relevant*) and the ones that are not, and then comparing the number of relevant results returned by the retrieval system.

To formalize this notion of quality, there has been defined several measures of quality. We are focusing on *precision* and *recall* and the relationship between them. Let R be the set of documents that are relevant, given a query q in a reference collection I. Also, let A be the set of documents retrieved by the system, when submitting the query q, and Ra the set of documents retrieved that were relevant (i.e., were in the set R). We can define:

- Recall: Ratio between the relevant retrieved documents, and the set of relevant documents.

$$Recall = \frac{|Ra|}{|R|}$$



(a) Data from 1 engine.

(b) Data from 3 engines.

Figure 2.2: Average Precision/Recall

- Precision: Ratio between the relevant retrieved documents and the set of retrieved documents.

$$Precision = \frac{|Ra|}{|A|}$$

Since each of these values by itself may not be sufficient (e.g., a system might get full recall by retrieving all the documents in the collection), it is possible to analyze them by combining the measures. For example, to analyze graphically the quality of a retrieval system, we can plot the *average precision-recall* (see Figure 2.2) and observe the behavior of the precision and recall of a system. These type of plots is useful also for comparing the retrieval of different engines. For example, on Figure 2.1(b) we observe the curves for 3 different engines. We can observe that Engine_2 has lower precision than the others at low recall, but as the recall increases, its precision doesn't degrade as fast as the other engines.

Another common measure is to calculate precision at certain document cut-offs, for example, analyze the precision at the first 5 documents. This is usually called *precision at* n (P@n) and represents the quality of the answer, since the user is frequently presented with only the first n documents retrieved, and not with the whole list of results.

As mentioned before, to calculate precision and recall, it is necessary to analyze the entire document collection, and for each query determine the documents that are relevant. This judgment of whether a document is relevant or not, must be done by an expert on the field that can understand the need represented by the query. In some cases, this analysis is not feasible since the document collection is too large (for example, the whole Web) or maybe the user intention behind the query is not clear. To address this problem, as mentioned on section 2.1.2, besides the document collection, the TREC conference also provides a set of queries (*topics*) and the corresponding set of relevant documents (*relevance judgment*). Using the document collection provided for each track, they have defined a set of topics, with a description of the intention behind it, that can be used to query the engine in study and then compare the results obtained with the list of relevant documents.

Chapter 3

Search Engines

There are several open source search engines available to download and use. In this study it is presented a list of the available search engines and an initial evaluation of them that permits to have a general overview of the alternatives. The criteria used in this initial evaluation was the development status, the current activity and the date of the last update made to the search engine. We compared 29 search engines: ASPSeek, BBDBot, Datapark, ebhath, Eureka, ht://Dig, Indri, ISearch, IXE, Lucene, Managing Gigabytes (MG), MG4J, mnoGoSearch, MPS Information Server, Namazu, Nutch, Omega, OmniFind IBM Yahoo! Ed., OpenFTS, PLWeb, SWISH-E, SWISH++, Terrier, WAIS/ freeWAIS, WebGlimpse, XML Query Engine, XMLSearch, Zebra, and Zettair.

Based on the information collected, it is possible to discard some projects because they are considered outdated (e.g. last update is prior to the year 2000), the project is not maintained or paralyzed, or it was not possible to obtain information of them. For these reasons we discarded ASPSeek, BBDBot, ebhath, Eureka, ISearch, MPS Information Server, PLWeb, and WAIS/freeWAIS.

In some cases, a project was rejected because of additional factors. For example, although the MG project (presented on the book "Managing Gi-

gabytes" [18]) is one of the most important work on the area, it was not included in this work, due to the fact that it has not been updated since 1999. Another special case is the Nutch project. The Nutch search engine is based on the Lucene search engine, and is just an implementation that uses the API provided by Lucene. For this reason, only the Lucene project will be analyzed. And finally, XML Query Engine and Zebra were discarded since they focus on structured data (XML) rather than on semi-structured data as HTML.

Therefore, the initial list of search engines that we wanted to cover in the present work were: Datapark, ht://Dig, Indri, IXE, Lucene, MG4J, mnoGoSearch, Namazu, OmniFind, OpenFTS, Omega, SWISH-E, SWISH++, Terrier, WebGlimpse (Glimpse), XMLSearch, and Zettair. However, with the preliminary tests, we observed that the indexing time for Datapark, mnoGoSearch, Namazu, OpenFTS, and Glimpse where 3 to 6 times longer than the rest of the search engines, for the smallest database, and hence we also did not considered them on the final performance comparison.

3.1 Features

As mentioned before, each of the search engines can be characterized by the features they implement as well as the performance they have in different scenarios. We defined 13 common features that can be used to describe each search engine, based only on the functionalities and intrinsic characteristics they possess:

Storage Indicates the way the indexer stores the index, either using a database engine or simple file structure (e.g. an inverted index).

Incremental Index Indicates if the indexer is capable of adding files to an existent index without the need of regenerating the whole index.

Results Excerpt If the engine gives an excerpt ("snippet") with the results.

Results Template Some engines give the possibility to use a template for parsing the results of a query.

Stop words Indicates if the indexer can use a list of words used as stop words in order to discard too frequent terms.

Filetype The types of files the indexer is capable of parsing. The common filetype of the engines analyzed was HTML.

Stemming If the indexer/searcher is capable of doing stemming operations over the words.

Fuzzy Search Ability of solving queries in a fuzzy way, i.e. not necessarily matching the query exactly.

Sort Ability to sort the results by several criteria.

Ranking Indicates if the engine gives the results based on a ranking function.

Search Type The type of searches it is capable of doing, and whether it accepts query operators.

Indexer Language The programming language used to implement the indexer. This information is useful in order to extend the functionalities or integrate it into an existent platform.

License Determines the conditions for using and modifying the indexer and/or search engine.

On Table 3.2 it is presented a summary of the features each of the search engines have. In order to make a decision it is necessary to analyze the features as a whole, and complement this information with the results of the performance evaluation.

3.2 Description

Each of the search engines that will be analyzed can be described shortly, based on who and where developed it and its main characteristic that identifies it.

ht://Dig [16] is a set of tools that permit to index and search a website. It provides with a command line tool to perform the search as well as a CGI

interface. Although there are newer versions than the one used, according to their website, the version 3.1.6 is the fastest one.

IXE Toolkit is a set of modular C++ classes and utilities for indexing and querying documents. There exists a commercial version from Tiscali (Italy), as well as a non-commercial version for academic purposes.

Indri [3] is a search engine built on top of the *Lemur* [4] project, which is a toolkit designed for research in language modeling and information retrieval. This project was developed by a cooperative work between the University of Massachusetts and Carnegie Mellon University, in the USA.

Lucene [6] is a text search engine library part of the Apache Software Foundation. Since it is a library, there are some applications that make use of it, e.g. the Nutch project [8]. In the present work, the simple applications bundled with the library were used to index the collection.

MG4J [7] (Managing Gigabytes for Java) is full text indexer for large collection of documents, developed at the University of Milano, Italy. As by-products, they offer general-purpose optimized classes for processing strings, bit-level I/O, etc.

Omega is an application built on top of *Xapian* [14] which is an Open Source Probabilistic Information Retrieval library. Xapian is written in C++ but can be binded to different languages (Perl, Python, PHP, Java, TCL, C#).

IBM Omnifind Yahoo! Edition [2] is a search software that enables rapid deployment of intranet search. It combines internal search, based on Lucene search engine, with the possibility to search on Internet using Yahoo! search engine.

SWISH-E [11] (Simple Web Indexing System for Humans - Enhanced) is an open source engine for indexing and searching. It is an enhanced version of SWISH, written by Kevin Hughes.

SWISH++ [10] is an indexing and searching tool based on Swish-E, although completely rewritten in C++. It has most of the features of Swish-E, but not all of them.

Terrier [12] (TERabyte RetrIEveR) is a modular platform that allows rapid development of Web, intranet and desktop search engines, developed at the University of Glasgow, Scotland. It comes with the ability to index, query and evaluate standard TREC collections.

XMLSearch is a set of classes developed in C++ that permits indexing and searching over document collections, by extending the search with text operators (equality, prefix, suffix, phrase, etc). There is a commercial version available from Barcino (Chile), and a non-commercial version for academic use.

Zettair [15] (formerly known as Lucy) is a text search engine developed by the Search Engine Group at RMIT University. Its primary feature is the ability to handle large amounts of text.

3.3 Evaluation

As seen before, each search engine has multiple characteristics that differentiates it from the other engines. To make a comparison of the engines, we would like to have a well-defined qualification process that can give the user an objective grade indicating the quality of each search engine. The problem is that it depends on the particular needs of each user and the main objective of the engine, how to choose the "best" search engine. For example, the evaluation can be tackled from the usability point of view, i.e. how simple is to use the engine out-of-the-box, and how simple it is to customize it in order to have it running. This depends on the main characteristic of the search engine. For example, Lucene is intended to be an index and search API, but if you need the features of Lucene as a front-end you must focus on the subproject Nutch. Another possibility is to analyze the common characteristics, as indexing and searching performance, and these features are much more analytical, but they must be analyzed with care since they are not the only feature. For this reason, we present a comparison based on these quantifiable parameters (indexing time, index size, resource con-

sumption, searching time, precision/recall, etc) and, at the end, we present several use cases and possible alternatives for each case.

Search Engine	Update	Version	Observation
ASPSeek	2002	N/A	The project is paralyzed.
BBDBot	2002	N/A	Last update was on 2002, but since then it has not have any activity.
Datapark	13/03/2006	4.38	
ebhath	N/A	N/A	No existing website.
Eureka	N/A	N/A	Website is not working.
ht://Dig	16/06/2004	3.2.0b6	
Indri	01/2007	2.4	
ISearch	02/11/2000	1.75	According to the website, "the software is not actively maintained, although it is available for download".
IXE	2007	1.5	
Lucene	02/03/2006	1.9.1	
Managing Gigabytes	01/08/1999	1.2.1	
MG4J	03/10/2005	1.0.1	
mnoGoSearch	15/03/2006	3.2.38	
MPS Inform. Server	01/09/2000	6.0	
Namazu	12/03/2006	2.0.16	
Nutch	31/03/2006	0.7.2	Subproject of the Lucene project.
Omega	08/04/2006	0.9.5	Omega is an application that uses the Xapian library.
OmniFind IBM Yahoo!	2006/12/07	8.4.0	
OpenFTS	05/04/2005	0.39	
PLWeb	16/03/1999	3.0.4	On 2000, AOL Search published a letter stating that the code will no longer be available.
SWISH-E	17/12/2004	2.4.3	
SWISH++	14/03/2006	6.1.4	
Terrier	17/03/2005	1.0.2	
WAIS & freeWAIS	N/A	N/A	The software is outdated.
WebGlimpse	01/04/2006	4.18.5	Uses Glimpse as the indexer.
XML Query Engine	02/04/2005	0.69	It is an XML search engine.
Zebra	23/02/2006	1.3.34	It is an XML search engine.
Zettair	09/2006	0.93	

Table 3.1: Initial characterization of the available open source search engines.

Search Engine	Storage[f]	Increm. Index	Results Excerpt	Results Template	Stop words	Filetype[e]	Stemming	Fuzzy Search	Sort[d]	Ranking	Search Type[c]	Indexer Lang.[b]	License[a]
Datapark	2	■	■	■	■	1,2,3	■	■	1,2	■	2	1	4
ht://Dig	1	■	■	■	■	1,2	■	■	1	■	2	1,2	4
Indri	1	■	■	■	■	1,2,3,4	■	■	1,2	■	1,2,3	2	3
IXE	1	■	■	■	■	1,2,3	□	■	1,2	■	1,2,3	2	8
Lucene	1	■	□	□	■	1,2,4	■	■	1	■	1,2,3	3	1
MG4J	1	■	■	■	■	1,2	■	□	1	■	1,2,3	3	6
mnoGoSearch	2	■	■	■	■	1,2	■	■	1	■	2	1	4
Namazu	1	■	■	■	□	1,2	□	□	1,2	■	1,2,3	1	4
Omega	1	■	□	■	■	1,2,4,5	■	□	1	■	1,2,3	2	4
OmniFind	1	■	■	■	■	1,2,3,4,5	■	■	1	■	1,2,3	3	5
OpenFTS	2	■	□	□	■	1,2	■	■	1	■	1,2	4	4
SWISH-E	1	■	□	□	■	1,2,3	■	■	1,2	■	1,2,3	1	4
SWISH++	1	■	□	□	■	1,2	■	□	1	■	1,2,3	2	4
Terrier	1	□	□	□	■	1,2,3,4,5	■	■	1	■	1,2,3	3	7
WebGlimpse	1	■	■[g]	■[g]	□	1,2	□	■	1[e]	■	1,2,3	1	8,9
XMLSearch	1	■	□	□	■	3	□	■	3	□	1,2,3	2	8
Zettair	1	■	■	□	■	1,2	■	□	1	■	1,2,3	1	2

[a] 1:Apache,2:BSD,3:CMU,4:GPL,5:IBM,6:LGPL,7:MPL,8:Comm,9:Free
[b] 1:C, 2:C++, 3:Java, 4:Perl, 5:PHP, 6:Tcl
[c] 1:phrase, 2:boolean, 3:wild card.
[d] 1:ranking, 2:date, 3:none.
[e] 1:HTML, 2:plain text, 3:XML, 4:PDF, 5:PS.
[f] 1:file, 2:database.
[g] Commercial version only.

■ Available
□ Not Available

Table 3.2: Main characteristics of the open source search engines analyzed.

Chapter 4

Methodology

One of the objectives of this study is to present a comparison of the performance of open source search engines in different scenarios (i.e. using document collections of different sizes), and evaluate them using a common criteria. In order to perform this benchmark, we divided the study in the following steps:

1. Obtain a document collection in HTML

2. Determine the tool that will be used to monitor the performance of the search engines

3. Install and configure each of the search engines

4. Index each document collection

5. Process and analyze index results

6. Perform searching tasks

7. Process and analyze search results.

4.1 Document collections

To execute the performance comparison between the different search engines, it was necessary to have several document collections of different sizes, ranging from a collection of less than 1 gigabyte of text, to 2.5 or 3 gigabytes of text. Another requirement was the file-type that will be used, and the common file-type supported by the search engines that were analyzed was HTML. In order to have a collection of nearly 3 GB of HTML documents, one possible solution was to use an on-line site and perform a crawl over the documents and obtain the collection, but this work is focused on the indexing capabilities of the search engines, so it was decided to use a local collection.

To create this document collection, a TREC-4 collection was obtained. This collection consists on several files containing the documents of The Wall Street Journal, Associated Press, Los Angeles Times, etc. Each of these files is in SGML format, so it was necessary to parse these documents and generate a collection of HTML documents of approximately 500kB each. Afterward, the collection was separated into 3 groups of different sizes: one of 750MB (1,549 documents), another of 1.6GB (3,193 documents), and one of 2.7GB (5,572 documents).

Afterward, the comparison was extended to using the WT10g TREC Web corpus (WebTREC). This collection consists on 1,692,096 documents, divided into 5117 files, and the total size is 10.2 GB. The collection was divided into subcollections of different sizes (2.4GB, 4.8GB, 7.2GB, and 10.2 GB) in order to compare the corresponding indexing time. The searching tests were performed over the whole WT10g collection.

4.2 Performance Comparison Tests

We executed 5 different tests over the document collections. The first three experiments were conducted over the parsed document collection (TREC-4),

and the last two experiments were conducted over the WT10g WebTREC document collection. The first test consisted on indexing the document collection with each of the search engines and record the elapsed time as well as the resource consumption. The second test consisted on comparing the search time of the search engines that performed better during the indexing tests, and analyze their performance with each of the collections. The third test consisted on comparing the indexing time required for making incremental indices. The indexing process of all the search engines were performed sequentially, using the same computer. The fourth experiment consisted on comparing the indexing time for subcollections of different sizes from the WT10g, with the search engines that were capable of indexing the whole collection of the previous experiments. Finally, the fifth experiment consisted on analyzing the searching time, precision and recall using a set of query topics, over the full WT10g collection.

4.3 Setup

The main characteristics of the computer used: Pentium 4HT 3.2 GHz processor, 2.0 GB RAM, SATA HDD, running under Debian Linux (Kernel 2.6.15). In order to analyze the resource consumption of every search engine during the process of indexing, it was necessary to have a monitoring tool. There are some open source monitors available, for example, “Load Monitor” [5] and “QOS” [9], but for this work a simple monitor was sufficient. For this reason, we implemented a simple daemon that logged the CPU and memory consumption of a given process, at certain time intervals. Afterward, the information collected can be easily parsed in order to generate data that can be plotted with Gnuplot.

Chapter 5

Tests

5.1 Indexing

5.1.1 Indexing Test over TREC-4 collection

The indexing tests consisted on indexing the document collections with each of the search engines and record the elapsed time as well as the resource consumption (CPU, RAM memory, and index size on disk). After each phase, the resulting time was analyzed and only the search engines that had "reasonable" indexing times continued to be tested on the following phase with the bigger collection. We arbitrarily defined the concept of "indexers with *reasonable* indexing time", based on the preliminary observations, as the indexers with indexing time no more than 20 times the fastest indexer.

Indexing Time

On Figure 5.1 we present a graphical comparison of the search engines that were capable of indexing all the document collections (in reasonable time). As mentioned on chapter 3, we discarded Datapark, Glimpse, mnoGoSearch, Namazu, and OpenFTS search engines, since their indexing time, for the 750MB collection, ranged from 77 to more than 320 minutes. Compared to the other search engines, their performance was very poor. An important

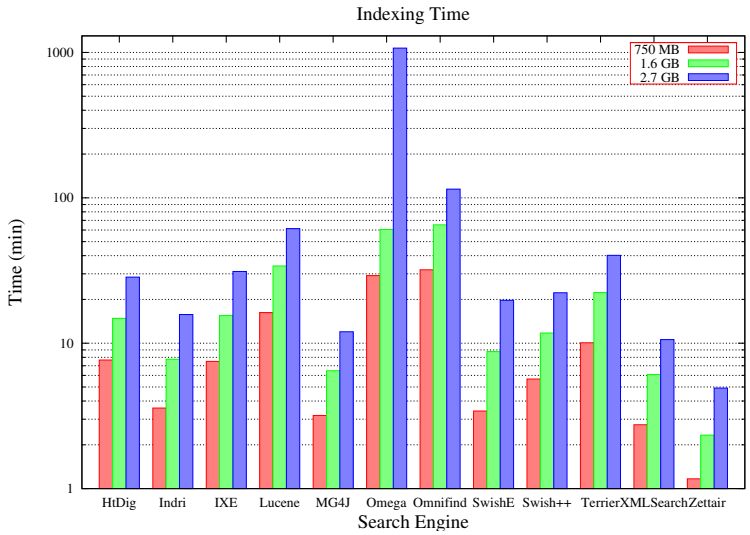


Figure 5.1: Indexing time for document collections of different sizes (750MB, 1.6GB, and 2.7GB) of the search engines that were capable of indexing all the document collections.

observation is that all of the search engines that used a database for storing the index had indexing time much larger than the rest of the search engines.

For the 750MB collection, the search engines had indexing time between 1 and 32 minutes. Then, with the 1.6GB collection, their indexing time ranged from 2 minutes to 1 hour. Finally, with the 2.7GB collection, the indexing time of the search engines, with the exception of Omega, was between 5 minutes and 1 hour. Omega showed a different behavior than the other, since the indexing time for the larger collection was of 17 hours and 50 minutes.

RAM Memory and CPU Consumption

Using the monitoring tool described on chapter 4, we were able to analyze the behavior of the search engines, during the indexing stage. The RAM consumption corresponds to the percentage of the total physical memory of the

server, that was used during the test. We observed that their CPU consumption remained constant during the indexing stage, using almost the 100% of the CPU. On the other hand, we observed 6 different behaviors on the RAM usage: *constant* (C), *linear* (L), *step* (S), and a combination of them: *linear-step* (L-S), *linear-constant* (L-C), and *step-constant* (S-C). ht://Dig, Lucene, and XMLSearch had a steady usage of RAM during the whole process. MG4J, Omega, Swish-E, and Zettair presented a linear growth in their RAM usage, and Swish++ presented a step-like behavior, i.e. it started using some memory, and then it maintained the usage for a period of time, and then continued using more RAM. Indri had a linear growth on the RAM usage, then it decreased abruptly the amount used, and then started using more RAM in a linear way. Terrier's behavior was a combination of step-like growth, and then it descended abruptly, and kept constant their RAM usage until the end of the indexing. Finally, Omega's behavior was a linear growth, but when it reached the 1.6GB of RAM usage, it maintained a constant usage until the end of the indexing.

Index Size

In Table 5.2 it is presented the size of the indices created by each of the search engines that were able of indexing the three collections in reasonable time. We can observe 3 groups: indices whose size range between 25%-35%, a group using 50%-55%, and the last group that used more than 100% the size of the collection.

We also compared the time needed for making incremental indices using three sets of different sizes: 1%, 5%, and 10% of the initial collection. We based on the indices created for the 1.6GB collection and each of the new collections had documents that were not included before. We compared ht://Dig, Indri, IXE, Swish-E, and Swish++. On Figure 5.2 we present the graph comparing their incremental indexing time.

Search Engine	750MB			1.6GB			2.7GB		
	Max. CPU	Max. RAM	RAM Behav.	Max. CPU	Max. RAM	RAM Behav.	Max. CPU	Max. RAM	RAM Behav.
ht://Dig	100.0%	6.4 %	C	100.0%	6.4 %	C	88.9%	6.4 %	C
Indri	100.0%	7.3 %	L-S	97.5%	8.0 %	L-S	88.6%	9.7 %	L-S
IXE	96.7%	39.1 %	S	98.7%	48.5 %	S	92.6%	51.5 %	S
Lucene	99.4%	20.0 %	L	100.0%	38.3 %	L	99.2%	59.4 %	L
MG4J	100.0%	23.4 %	C	100.0%	48.0 %	C	100.0%	70.4 %	C
Omega	100.0%	26.8 %	L	99.2%	52.1 %	L	94.0%	83.5 %	L-C
OmniFind	78.4%	17.6 %	S	**83.3**%	18.3 %	S	83.8%	19.5 %	S
Swish-E	100.0%	16.2 %	L	98.9%	31.9 %	L	98.8%	56.7 %	L
Swish++	99.6%	24.8 %	S	98.5%	34.3 %	S	98.6%	54.3 %	S
Terrier	99.5%	58.1 %	S-C	99.4%	78.1 %	S-C	98.7%	86.5 %	S-C
XMLSearch	93.6%	**0.6** %	C	86.2%	**0.6** %	C	90.1%	**0.6** %	C
Zettair	**77.2**%	20.2 %	L	98.1%	22.3 %	L	**82.7**%	23.1 %	L
RAM behavior: C – constant, L – linear, S – step.									

Table 5.1: Maximum CPU and RAM usage, RAM behavior, and index size of each search engine, when indexing collections of 750MB, 1.6GB, and 2.7GB.

5.1.2 Indexing WT10g subcollections

Another test performed consisted on comparing the time needed to index different subcollections of the WebTREC (WT10g) collection. We observed two groups of search engines: one group was able to index the collection with the original format (i.e. each file consisted on a set of records that contained the actual HTML pages); and another group that did not understand the format, so it was necessary to parse the collection and extract each HTML file separately. Indri, MG4J, Terrier, and Zettair were able to index the WT10g files without any modification, but ht://Dig, IXE[1], Lucene, Swish-E, and Swish++ needed the data to be splitted. XMLSearch was not included in the tests with the WT10g collection, since it does not make ranking over the results.

First, we tested each of the search engines that passed the previous

[1]It included the script to parse the TREC documents, so the splitting into small HTML files was "transparent" for the user.

Search Engine	Index Size		
	750MB	1.6GB	2.7GB
ht://Dig	108%	92%	104%
Indri	61%	58%	63%
IXE	30%	28%	30%
Lucene	**25**%	23%	26%
MG4J	30%	27%	30%
Omega	104%	95%	103%
OmniFind	175%	159%	171%
Swish-E	31%	28%	31%
Swish++	30%	26%	29%
Terrier	51%	47%	52%
XMLSearch	**25**%	**22**%	**22**%
Zettair	34%	31%	33%

Table 5.2: Index size of each search engine, when indexing collections of 750MB, 1.6GB, and 2.7GB.

test, with the whole WT10g collection (10.2 GB). Only Indri, IXE, MG4J, Terrier, and Zettair could index the whole collection with a linear growth in time (compared to their corresponding indexing times on the previous tests). The other search engines did not scale appropriately or crashed due to lack of memory. ht://Dig and Lucene took more than 7 times their expected indexing time and more than 20 times the fastest search engine (Zettair); while Swish-E and Swish++ crashed due to an "out of memory" error. Based on these results, we analyzed the indexing time with subcollections of the original collection, of different sizes (2.4GB, 4.8GB, 7.2GB, and 10.2 GB). On Figure 5.3 we present a comparison of the indexing time for each of the search engines that were capable of indexing the entire collection. We can observe that these search engines scaled linearly as the collection grew.

5.2 Searching

The searching tests are based on a set of queries that must be answered, and then compare the level of "correct" results that each engine retrieved. Depending on the collection and the set of queries, this idea of "correct"

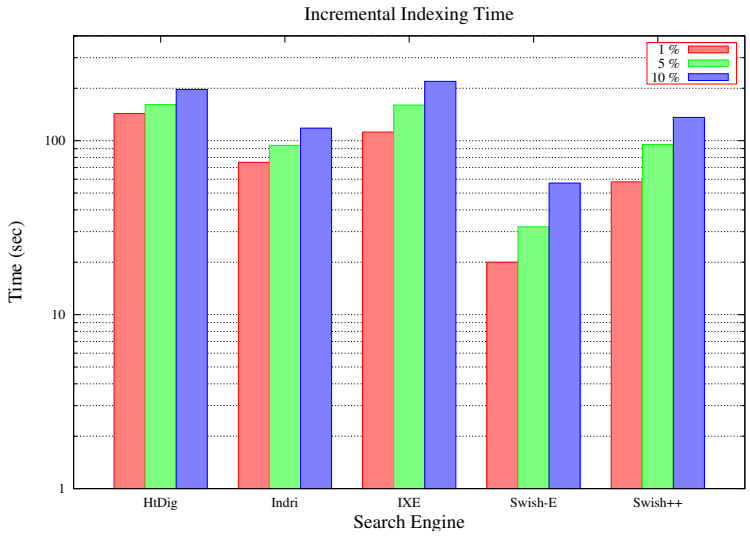


Figure 5.2: Indexing time for Incremental Indices

results will be defined. In order to obtain the set of queries to use, we can identify three approaches:

- Use a query log to find "real" queries
- Generate queries based on the content of the collection
- Use predefined set of queries, strongly related to the content of the collection

The first approach, using a query log, seems attractive since it will test the engines in a "real-world" situation. The problem with this approach is that, in order to be really relevant, it must be tested with a set of pages that are related to the query log, i.e. we would need to obtain a set of crawled pages and a set of query logs that were used over these documents. Since on the first tests we are using the TREC-4 collection which is based on a set of news articles, we don't have a query log relevant to these documents. For this reason we used a set of randomly created queries (more detail on section

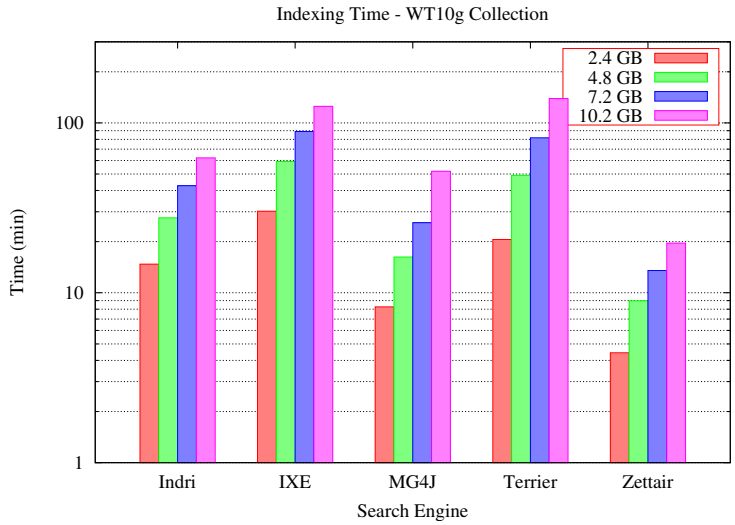


Figure 5.3: Indexing time for the WT10g collection.

5.2.1) based on the words contained on the documents, using different word distributions. Finally, the most complete test environment can be obtained by using a set of predefined set of queries, related to the document collection. These queries can be used on the second set of experiments, that operate over the WT10g collection, created for the TREC evaluation. This approach seems to be the most complete and close to the real-world situation, with a controlled environment.

For the reasons mentioned above, we used a set of randomly generated queries over the TREC-4 collection, and a set of topics and query relevance for the WT10g TREC collection.

5.2.1 Searching Tests over TREC-4 collection

The Searching Tests were conducted using the three document collections, with the search engines that had better performance during the Indexing Tests (i.e., ht://Dig, Indri, IXE, Lucene, MG4J, Swish-E, Swish++, Terrier, XMLSearch, and Zettair). These tests consisted on creating 1-word and 2-

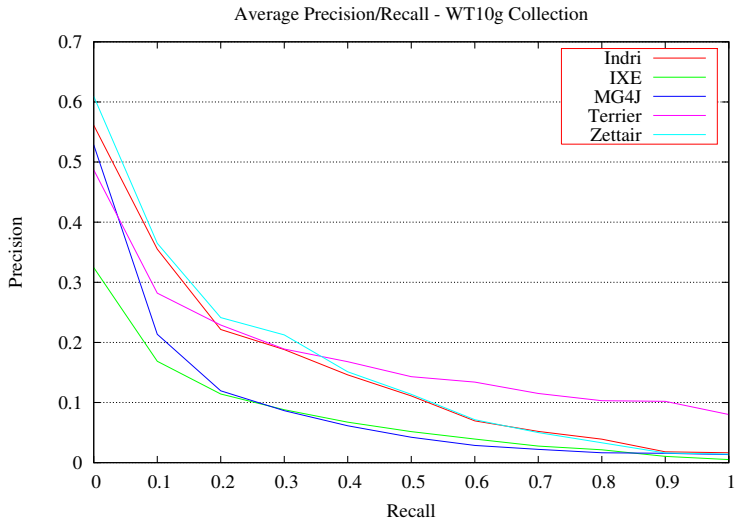


Figure 5.4: Average Precision/Recall for the WT10g collection.

words queries from the dictionary obtained from each of the collections, and then analyzing the search time of each of the search engines, as well as the "retrieval percentage". The "retrieval percentage" is the ratio between the amount of documents retrieved by a search engine and the maximum amount of documents that were retrieved by all of the search engines.

In order to create the queries, we chose 1 or 2 words by random from the dictionary of words that appeared on each of the collections (without stopwords), using several word distributions:

1. Original distribution of the words (power law)

2. Uniform distribution from the 5% of the most frequent words

3. Uniform distribution from the 30% of the least frequent words.

The queries used on each of the collections considered the dictionary and distribution of words particular to that collection. The word frequency of all of the collections followed a Zipf law.

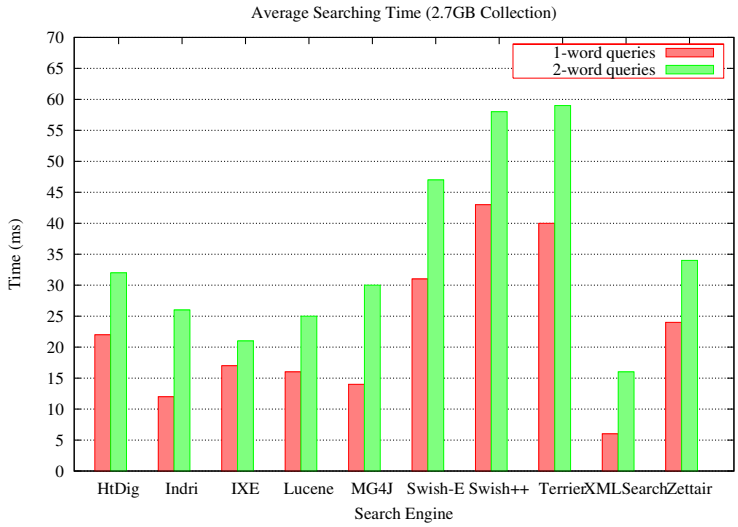


Figure 5.5: Average search time (2.7 GB collection).

Searching time and Retrieval percentage

After submitting the set of 1- and 2-words queries (each set consisted on 100 queries), we could observe the average searching time for each collection and the corresponding retrieval percentage. For the 2-words queries, we considered the matching of any of the words (using the `OR` operator). On figure 5.5 we present a graph comparison of the average search times of each search engine for the 2.7GB collection.

The results obtained show that all of the search engines that qualified for the searching stage had similar searching times in each of the set of queries. In average, the searching time of submitting a 1-word or a 2-words query differed by a factor of 1.5 or 2.0, in a linear way. The fastest search engines were Indri, IXE, Lucene, and XMLSearch. Then it was MG4J, and Zettair. The retrieval percentage was also very similar between them, but it decreased abruptly as the collection became larger and with queries from the lowest 30%.

RAM memory consumption

During the searching stage we observed 4 different behaviors. Indri, IXE, Lucene, MG4J, Terrier, and Zettair used constant memory (1%-2% memory), independent of the size of the collection queried. XMLSearch used few and constant memory, but dependant on the size of the collection (0.6%, 0.8%, and 1.1% for every collection respectively). The memory usage of Swish++ increased linearly upto 2.5%, 3.5% and 4.5% for every collection respectively, and Swish-E and ht://Dig used much more memory with a constant curve. Swish-E used 10.5% RAM and ht://Dig used 14.4% RAM.

5.2.2 Precision and Recall Comparison

Using the indices created for WT10g, it was possible to analyze precision and recall for each of the search engines. We used the 50 topics (using title-only queries) used on the TREC-2001 Web Track for the "Topic Relevance Task", and their corresponding relevance judgments. To have a common scenario for every search engine, we didn't use any stemming, or stop-word removal of the queries, and used the `OR` operator between the terms.

Afterward, the processing of the results was done using the `trec_eval` software, that permits to evaluate the results with the standard NIST evaluation and is freely available. As output of the program, you obtain general information about the queries (e.g. number of relevant documents) as well as precision and recall statistics. We focused on the interpolated average precision/recall and the precision at different levels. The average precision/recall permits to compare the retrieval performance of the engines by observing their behavior throughout the retrieval (see Figure 5.4). On the other hand, we also compared the precision at different cutoff values, allowing to observe how it behaves at different thresholds (see Table 5.3).

Search Engine	P@5	P@10	P@15	P@20	P@30
Indri	0.2851	0.2532	0.2170	0.2011	0.1801
IXE	0.1429	0.1204	0.1129	0.1061	0.0939
MG4J	0.2480	0.2100	0.1800	0.1600	0.1340
Terrier	0.2800	0.2400	0.2130	0.2100	0.1930
Zettair	**0.3240**	**0.2680**	**0.2507**	**0.2310**	**0.1993**

Table 5.3: Answer Quality for the WT10g.

5.3 Global Evaluation

Based on the results obtained, after performing the tests with different collection of documents, the search engines that took less indexing time were: ht://Dig, Indri, IXE, Lucene, MG4J, Swish-E, Swish++, Terrier, XMLSearch, and Zettair. When analyzing the size of the index created, there are 3 different groups: IXE, Lucene, MG4J, Swish-E, Swish++, XMLSearch and Zettair created an index of 25%-35% the size of the collection; Terrier had an index of 50%-55% of the size of the collection; and ht://Dig, Omega, and OmniFind created an index of more than 100% the size of the collection. Finally, another aspect to consider is the behavior that had the RAM usage during the indexing stage. ht://Dig, Lucene, and XMLSearch had a constant usage of RAM. The first two used the same amount of RAM memory, independent of the collection (between 30MB and 120MB). On the other hand, IXE, MG4J, Swish-E, Swish++, and Terrier used much more memory, and growed in a linear way, reaching between 320MB to 600MB for the smallest collection, and around 1GB for the largest collection.

Another fact that can be observed is related to the way the search engines store and manipulate the index. The search engines that used a database (DataparkSearch, mnoGoSearch, and OpenFTS) had a very poor performance during the indexing stage, since their indexing time was 3 to 6 larger than the best search engines.

On the second part of the tests, it was possible to observe that, for a

given collection and type of queries (1- or 2-words), the search engines had similar searching times. For the 1-word queries, the searching time ranged from less than 10 ms to 90 ms, while on the 2-words queries their searching time ranged from less than 10 ms to 110 ms. The search engines that had the smallest searching time were Indri, IXE, Lucene, and XMLSearch. The only difference observed is when searching over the least frequent words, since most of them retrieved 0 or 1 documents, the retrieval percentage is not representative.

From the tests performed with the WT10g collection we can observe that only Indri, IXE, MG4J, Terrier, and Zettair where capable of indexing the whole collection without considerable degradation, compared to the results obtained from the TREC-4 collection. Swish-E, and Swish++ were not able to index it, on the given system characteristics (operating system, RAM, etc.). ht://Dig and Lucene degraded considerably their indexing time, and we excluded them from the final comparison. Zettair was the fastest indexer and its average precision/recall was similar to Indri's, MG4J's, and Terrier's. IXE had low values on the average precision/recall, compared to the other search engines. By comparing the results with the results obtained on other TREC Tracks (e.g. Tera collection) we can observe that IXE, MG4J, and Terrier were on the top list of search engines. This difference with the official TREC evaluation can be explained by the fact that the engines are carefully fine-tuned by the developers, for the particular needs of each track, and most of this fine-tuning is not fully documented on the released version, since they are particularly fitted to the track objective.

Chapter 6

Conclusions

This study presents the methodology used for comparing different open source search engines, and the results obtained after performing tests with document collections of different sizes. At the beginning of the work, 17 search engines were selected (from the 29 search engines found), for being part of the comparison. After executing the tests, only 10 search engines were able to index a 2.7GB document collection in "reasonable" time (less than an hour), and only these search engines were used for the searching tests. It was possible to identify different behaviors, in relation to their memory consumption, during the indexing stage, and also observed that the size of the indexes created varied according to the indexer used. On the searching tests, there was no considerable difference on the performance of the search engines that were able to index the largest collections.

The final tests consisted on comparing their ability to index a larger collection (10GB) and analyze their precision at different levels. Only five search engines were capable of indexing the collection (given the characteristic of the server). By observing the average precision/recall we can observe that Zettair had the best results, but similar to the results obtained by Indri. By comparing these results with the results obtained on the official TREC evaluation, it is possible to observe some differences. This can be explained

Search Engine	Indexing Time (h:m:s)		Index Size (%)		Searching Time (ms)		Answer Quality P@5	
ht://Dig	(7)	0:28:30	(10)	104	(6)	32		-
Indri	(4)	0:15:45	(9)	63	(2)	19	(2)	0.2851
IXE	(8)	0:31:10	(4)	30	(2)	19	(5)	0.1429
Lucene	(10)	1:01:25	(2)	26	(4)	21		-
MG4J	(3)	0:12:00	(8)	60	(5)	22	(4)	0.2480
Swish-E	(5)	0:19:45	(5)	31	(8)	45		-
Swish++	(6)	0:22:15	(3)	29	(10)	51		-
Terrier	(9)	0:40:12	(7)	52	(9)	50	(3)	0.2800
XMLSearch	(2)	0:10:35	(1)	**22**	(1)	**12**		-
Zettair	(1)	**0:04:44**	(6)	33	(6)	32	(1)	**0.3240**

Table 6.1: Ranking of search engines, comparing their indexing time, index size, and the average searching time (for the 2.7GB collection), and the Answer Quality for the engines that parsed the WT10g. The number in parentheses corresponds to the relative position of the search engine.

by the fact that most of the search engines are fine-tuned by the developers for each of the retrieval task of TREC, and some of these tuning are not fully documented.

When comparing the results of the initial tests made with the discarded search engines (Datapark, mnoGoSearch, Namazu, OpenFTS, and Glimpse), it is possible to observe that the discarded search engines were much slower than the final search engines.

With the information presented on this work, it is possible to have a general view of the characteristics and performance of the available open source search engines in the indexing and retrieval tasks. On Table 6.1 we present a ranked comparison of the indexing time and index size when indexing the 2.7GB collection and the average searching time of each of the search engines. The ranked comparison of the searching time was made considering all the queries (1- and 2-words queries with original and uniform distribution) using the 2.7GB collection. Also we present the precision from the first 5 results for the search engines that indexed the WT10g collection.

By analyzing the overall quantitative results, over the small (TREC-

4) and the large (WT10g) collections, we can observe that Zettair is one of the most complete engines, due to its ability to process large amount of information in considerable less time than the other search engines (less than half the time of the second fastest indexer) and obtain the highest average precision and recall over the WT10g collection.

On the other hand, in order to make a decision on what search engine to use, it is necessary to complement the results obtained with any additional requirement of each website. There are some considerations to make, based on the programming language (e.g. to be able to modify the sources) and/or the characteristics of the server (e.g. RAM memory available). For example, if the size of the collection to index is very large and it tends to change (i.e. needs to be indexed frequently), maybe it can be wise to focus the attention on Zettair, MG4J or Swish++, since they are fast in the indexing and searching stages. Swish-E will also be a good alternative. On the other hand, if one of the constraints is the amount of disk space, then Lucene would be a good alternative, since it uses few space and has low retrieval time. The drawback is the time it takes to index the collection. Finally, if the collection does not change frequently, and since all the search engines had similar searching times, you can make a decision based on the programming language used by the other applications in the website, so the customization time is minimized. For Java you can choose MG4J, Terrier or Lucene, and for C/C++ you can choose Swish-E, Swish++, ht://Dig, XMLSearch, or Zettair.

Bibliography

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, Wokingham, UK, 1999.

[2] IBM OmniFind Yahoo! Homepage. http://omnifind.ibm.yahoo.net/.

[3] Indri Homepage. http://www.lemurproject.org/indri/.

[4] Lemur Toolkit Homepage. http://www.lemurproject.org/.

[5] Load Monitor Project Homepage. http://sourceforge.net/projects/monitor.

[6] Lucene Homepage. http://jakarta.apache.org/lucene/.

[7] Managing Gigabytes Homepage. http://www.cs.mu.oz.au/mg/.

[8] Nutch Homepage. http://lucene.apache.org/nutch/.

[9] QOS Project Homepage. http://qos.sourceforge.net/.

[10] SWISH++ Homepage. http://homepage.mac.com/pauljlucas/software/swish/.

[11] SWISH-E Homepage. http://www.swish-e.org/.

[12] Terrier Homepage. http://ir.dcs.gla.ac.uk/terrier/.

[13] Text REtrieval Conference (TREC) Homepage. http://trec.nist.gov/.

[14] Xapian Code Library Homepage. http://www.xapian.org/.

[15] Zettair Homepage. http://www.seg.rmit.edu.au/zettair/.

[16] ht://Dig Homepage. http://www.htdig.org/.

[17] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK, 2008.

[18] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, San Francisco, CA, 1999.