EVALUATION OF TURKISH TEXT INFORMATION RETRIEVAL USING RELATIONAL DATABASES VERSUS INFORMATION RETRIEVAL SYSTEMS

Ahmet ARSLAN

Master of Science Thesis

Computer Engineering Program

July, 2008

JÜRİ VE ENSTİTÜ ONAYI

Ahmet Arslan'nın "**Türkçe bilgi erişiminin, ilişkisel veri tabanlarının bilgi erişim sistemleri ile karşılaştırılması yoluyla değerlendirilmesi**" başlıklı **Bilgisayar Mühendisliği** Anabilim Dalındaki, Yüksek Lisans Tezi 30.06.2008 tarihinde, aşağıdaki jüri tarafından Anadolu Üniversitesi Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

		Adı-Soyadı	**İmza**
Üye (Tez Danışmanı)	**:**	**Yard. Doç. Dr. ÖZGÜR YILMAZEL**	**...…………**
Üye	**:**	**Yard. Doç. Dr. CÜNEYT AKINLAR**	**...…………**
Üye	**:**	**Yard. Doç. Dr. HAKAN G. ŞENEL**	**...…………**

Anadolu Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ……………… tarih ve ………… sayılı kararıyla onaylanmıştır.

Enstitü Müdürü

ABSTRACT

Master of Science Thesis

EVALUATION OF TURKISH TEXT INFORMATION RETRIEVAL USING RELATIONAL DATABASES VERSUS INFORMATION RETRIEVAL SYSTEMS

Ahmet ARSLAN

Anadolu University
Graduate School of Sciences
Computer Engineering Program

Supervisor: Assist. Prof. Dr. Özgür YILMAZEL
2008, 41 pages

Many of today's applications have a need for full-text search capabilities for various reasons. Although full-text search has traditionally been the domain of Information Retrieval (IR), nowadays popular Relational Database Management Systems (RDBMS) started to implement functionalities that support full-text indexing and searching.

The present thesis covers a comparison of the text retrieval performances of relational databases and IR Systems, as well as a comparison of the execution times during indexing and retrieval tasks over a Text REtrieval Conference (TREC)-like test collection for Turkish that contains 408,305 documents and 72 ad hoc queries. The effects of language specific processing for different systems are investigated. Also the effects of different query lengths and operators on retrieval performance are investigated. It is found that language specific preprocessing improves retrieval performance for all systems. Relational Databases are generally slower with longer queries.

Keywords: Information retrieval, Relational databases, Turkish language

ÖZET

Yüksek Lisans Tezi

TÜRKÇE BİLGİ ERİŞİMİNİN, İLİŞKİSEL VERİ TABANLARININ BİLGİ ERİŞİM SİSTEMLERİ İLE KARŞILAŞTIRILMASI YOLUYLA DEĞERLENDİRİLMESİ

Ahmet ARSLAN

Anadolu Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Yard. Doç. Dr. Özgür YILMAZEL
2008, 41 sayfa

Günümüzde birçok uygulama çeşitli nedenlerden dolayı tam metin arama özelliklerine ihtiyaç duymaktadir. Geneneksel olarak tam metin arama Bilgi Erşimi (BE) alanına girmesine rağmen, birçok İlişkisel Veritabanı Yönetim Sistemi (İVTYS) sağlayıcıları da tam metin arama özelliklerini ürünlerine eklemeye başlamışlardır.

Bu tezde Bilgi Erişimi Sistemlerinin ve İlişkiseş Veri Tabanı Sistemlerinin metin geri getirme performanslarının yanısıra geri getirme ve dizin oluşturma sırasındaki hızlarıda karşılaştırılmıştır. Bu karşılaştırma işlemi Türkçe için hazırlanmış olan içinde 408,305 döküman ve 72 test sorgusu içeren Text REtrieval Conference (TREC) benzeri bir döküman kolleksiyonu üzerinde yapılmıştır. Dillere özgü işlemelerin farklı sistemler üzerindeki etkileri incelenmiştir. Ayrıca çeşitli sorgu uzunluklarının ve sorgu işleçlerinin geri getirim performansı üzerindeki etkileri de incelenmiştir. Dillere özgü ön işlemelerin bütün sistemlerin geri getirim performanslarını artırdığı bulunmuştur. İlişkisel veritabanları genel olarak uzun sorgularda yavaş çalışmaktadır.

Anahtar Kelimeler: Bilgi Erişimi, İlişkisel Veritabanları, Türkçe Dili

ACKNOWLEDGEMENTS

I would like to thank my advisor Asst. Prof. Dr. Özgür Yılmazel for his guidance and support during my study. It was my pleasure to work with him during this study.

Ahmet Arslan

July, 2008

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVATIONS

CBR	: Clustered Based Retrieval
DBMS	: Database Management System
IR	: Information Retrieval
TREC	: Text Retrieval Conference
URL	: Universal Resource Locator

1. INTRODUCTION

Relational Database Management Systems have been the preferred way of managing data in many businesses for the past two decades. In recent years data that many businesses use are changing and moving from structured to unstructured (free form text). Although relational databases are designed to handle structured data, many businesses are still trying to use databases to manage their overall business data.

With the increase in unstructured text, developments of information retrieval systems have been gaining momentum. There are over 17 open source information retrieval libraries available with different features [1].

However businesses have been reluctant to move away from using databases because of database's familiarity. Many database vendors (IBM DB2[1], Microsoft SQL Server[2], MySQL[3], Oracle[4], PostgreSQL[5]) have recognized the need for free form text search and started implementing features that would support full-text search capabilities. Unlike structured information access, unstructured information access or free form text search is language dependent.

Database vendors are initially focusing their efforts in implementing these capabilities for the English Language. In this research we wanted to compare the capabilities of different relational databases and an open source information retrieval library on Turkish text documents.

We compare the retrieval performance of relational databases and IR Libraries, both for efficiency and effectiveness.

---

[1] http://www-306.ibm.com/software/data/db2/extenders/netsearch

[2] http://msdn2.microsoft.com/en-us/library/ms142571.aspx

[3] http://dev.mysql.com/doc/refman/5.0/en/fulltext-search.html

[4] http://www.oracle.com/technology/products/text/index.html

[5] http://www.postgresql.org/docs/8.3/static/textsearch-intro.html

We evaluate the effectiveness of each approach by running over a TREC sized test collection and comparing the relevancy. Efficiency is evaluated by comparing the searching and indexing times. The effects of query length and different Boolean query operators on retrieval quality are also evaluated.

We run experiments with out-of-the-box settings and also try to do language specific improvements for both relational databases and IR Libraries.

The organization of this thesis is as follows. Section 2 gives a background of the general concepts of Information Retrieval and briefly summarizes the related work on Turkish IR, Section 3 presents the details of our experimental setup including dataset and stemming algorithms that we used in this thesis, Section 4 contains the experimental results, and Section 5 provides concluding remarks and future work.

2. BACKGROUND

Over the past decade, businesses and organizations have spent substantial amount of money on relational database systems for managing and accessing their structured data. However, the amount of electronically stored unstructured data (web pages, manuals, reports, e-mails, faxes and presentations) is increasing rapidly. With the increase of free-form textual data, Information Retrieval Systems started to become new style of information access, taking the place of traditional relational databases.

2.1. Information Retrieval

Information retrieval (IR) is a very broad field of study. However from an academic perspective, information retrieval can be defined as follows:

"Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)" [2].

The main idea is to satisfy user information need by searching over a large document collection and retrieving the relevant ones. A document collection can be any type of source data, which can be used to extract text.

As shown on Figure 2.1 an IR system consists of several modules interacting with each other. It can be described as three main areas: Indexing, Searching and Ranking.

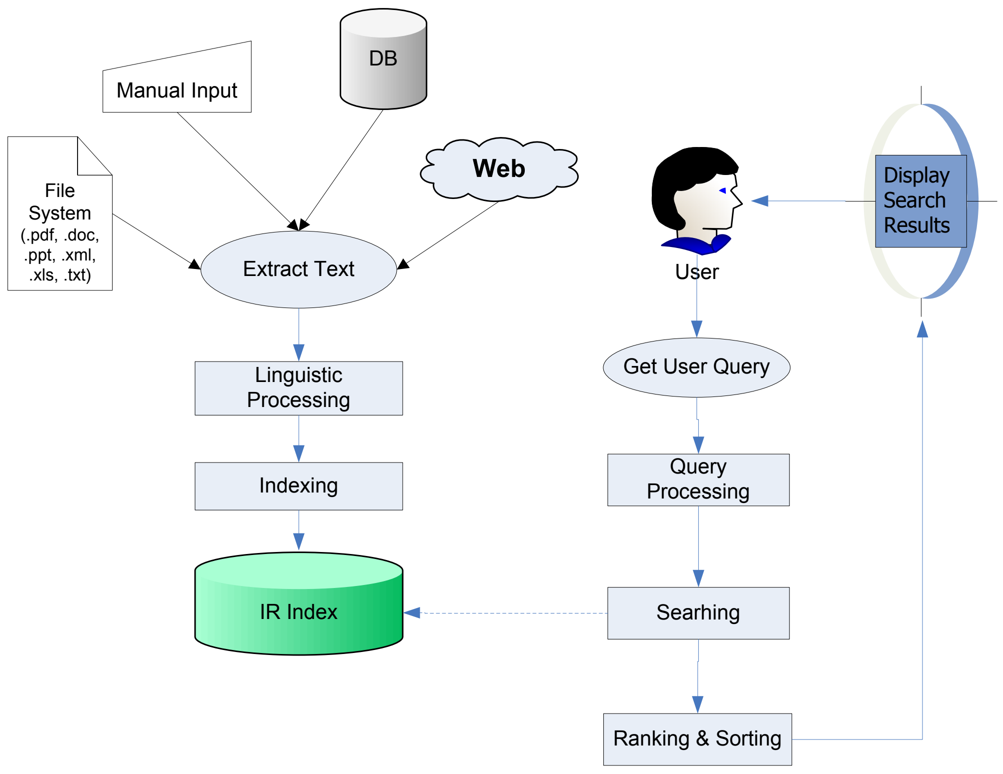


Figure 2.1. Information Retrieval process

2.1.1. Indexing

In order avoid linear searching through documents in a file system using "grep"; it is necessary to have the data stored in specially designed data structures. Inverted index is the most used data structure which allows making fast searches over the collection. The basic idea of an inverted index is shown in Figure 2.2. It keeps dictionary of terms that contains all the unique words in the collection. For each term there is a posting list that records which documents contains the term and the positions in the document. The dictionary in Figure 2.2 has been sorted alphabetically and each posting list is sorted by document identifier. The word "çok" occurs one time in document 1 at position 12, occurs two times in document 2 at positions 10, 14.

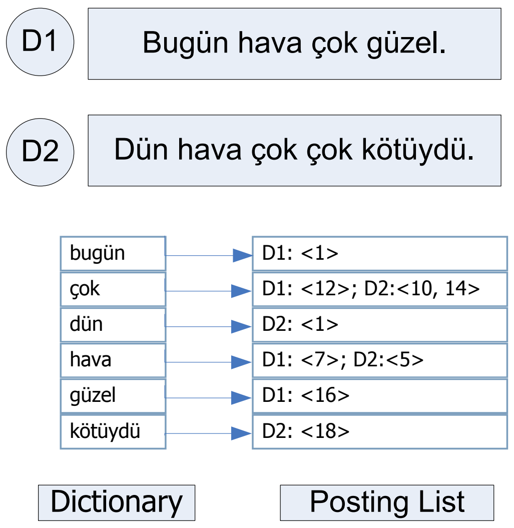


Figure 2.2. Inverted index example

2.1.2. Searching and Ranking

Using the inverted index, queries can be performed efficiently. Initially the query string is broken into words (terms). This process is called tokenization. Then each query term is searched over the dictionary. Since dictionary is sorted alphabetically this can be done using binary search which has a time complexity of $O(\log n)$. Then the posting lists of found query terms are processed according to query type. For example in Boolean retrieval model retrieved posting lists are merged using one of three Boolean operators (AND, OR, NOT). However in this model there is no scoring mechanism since in Boolean retrieval model index term weights are zero or one. Retrieved documents are totally equal and considered as relevant to the Boolean query. When the document collection size is large enough the retrieved set of documents can be too many for the user to examine all of the results. In order to satisfy user information need better, the retrieved result set must be displayed to the user after sorting by relevancy. This stage is called ranking and can be very imported for the Web search engines because web users examine first 10 or 20 of the results and don't look at the rest. Importance of ranking brings us to the vector space model.

In Vector space model each document and user query is represented as a vector with elements corresponding to each term in dictionary, with a non-binary index term weight. These term weights are used in computation of similarity between documents and queries. Dimension of the vector is equal to the number of unique terms in the dictionary. If a dictionary term does not occur in a document, corresponding weight in the vector becomes zero. The dictionary in Figure 2.2 has 6 unique terms; therefore the vector dimension is 6. The vector that represents document 1 would be

$$\vec{d_1} = (w_{1,1}, w_{2,1}, w_{3,1} = 0, w_{4,1}, w_{5,1}, w_{6,1} = 0)$$

Note that third and sixth elements of the vector are zero, since document one does not contain the words “dün” and “kötüydü”. And the query string “hava durumu” will be represented as

$$\vec{q} = (w_{1,q}= 0, w_{2,q}= 0, w_{3,q} = 0, w_{4,q}, w_{5,q}= 0, w_{6,q} = 0)$$

Note that all the elements of vector are zero except the fourth one, since query string contains only one word (“hava”) from the dictionary. After the conversion of documents and query to vectors then the similarity is calculated as follows:

$$\mathrm{sim}(d_j, q) = \frac{\vec{d}_j \bullet \vec{q}}{\left|\vec{d}_j\right| \times \left|\vec{q}\right|} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w^2{}_{i,j}} \times \sqrt{\sum_{j=1}^{t} w^2{}_{i,q}}} \tag{2.1}$$

Equation (2.1) is called cosine similarity of vectors of $\vec{d_j}$ and $\vec{q}$ where numerator is the dot product, while denominator is the product of Euclidean lengths. This measure is the cosine of the angle between *t*-dimensional query and document vector, shown in Figure 2.3, where *t* is the total number of unique terms in the dictionary.

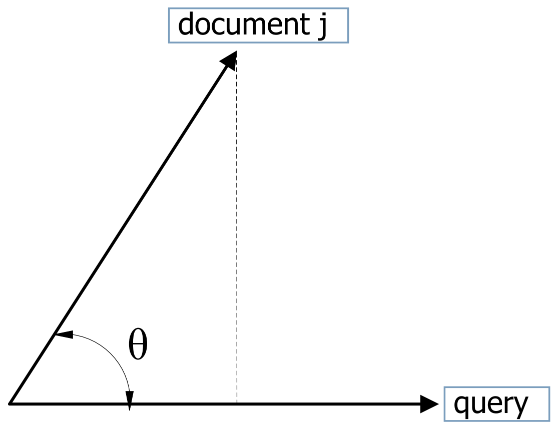


Figure 2.3. Cosine similarity. sim(d_j, q) = cos θ

There are many ways to calculate term weights ($w_{i,j}$). Gerard Salton and Christopher Buckley [3] described eight different term-weighting approaches in their work. The best known term-weighting scheme in IR is tf-idf weighting. It uses multiplication of term frequency and inverse document frequency to calculate a weight for each term in each document. Term frequency is the occurrence number of a term t in document d and denoted as $\mathrm{tf}_{t,d}$. Document frequency of a term t is the number of documents that contain that term and denoted as df_t. Inverse document frequency of a term is defined in Equation (2.2).

$$\mathrm{idf}_t = \log \frac{N}{df_t} \tag{2.2}$$

Where N is the total number of documents in the collection and df_t is the document frequency. The weight assigned to a term t in document d by the tf-idf weighting scheme is given in Equation (2.3).

$$w_{t,d} = \mathrm{tf}_{t,d} \times \log \frac{N}{df_t} \tag{2.3}$$

2.2. Turkish Information Retrieval

Turkish[6] is a language spoken by over 70 million people worldwide, making it the most commonly spoken of the Turkic languages. Other important Turkic languages are Azeri and Uzbek. In this thesis only the language of modern Turkey is considered.

Turkish is a part of the Turkic branch of the Altaic language family. The Turkish alphabet is slightly modified from the Latin alphabet, consisting of 29 letters, a certain number of which (Ç, Ğ, I, İ, Ö, Ş, and Ü) have been adapted or modified for the phonetic requirements of the language. The letters in alphabetical order are: a, b, c, ç, d, e, f, g, h, ı, i, j, k, l ,m, n, o, ö, p, r, s, ş, t, u, ü, v, y, z. Of these 29 letters, 8 are vowels (a, e, ı, i, o, ö, u, ü); the 21 others are consonants.

Turkish[6] is an agglutinative language (the largest are Hungarian, Finnish and Estonian) and frequently uses affixes. In linguistics, the term affix refers to either a prefix or a suffix. However in Turkish language there are only a few numbers of cases where prefix is added to a word. Table 2.1 gives an example of such rare case.

Table 2.1. Example usage of prefix in Turkish

Turkish	**English**	**Word Type**	**Prefix**	**English**	**Word Type**
mavi	blue	adjective	mas+mavi	deep blue	adjective
pembe	pink	adjective	pes+pembe	rose-pink	adjective

Turkish[6] extensively uses suffixes to form new words from stems. The suffixes used in Turkish fall approximately into two classes: derivational suffixes and inflectional suffixes. A derivational suffix makes a new dictionary entry from an old one; an inflectional suffix allows a dictionary-word to take its proper place in a sentence[7]. The majority of Turkish[6] words originate from the application of derivational and inflectional suffixes to a relatively small set of core vocabulary.

[6] http://en.wikipedia.org/wiki/Turkish_language

[7] http://en.wikipedia.org/wiki/Turkish_grammar

That’s the main reason that Turkish language gains more benefit than other languages from stemming. List of the words derived from a meaningful root by using derivational suffixes are shown on table 2.2, by using inflectional suffixes are shown on table 2.3.

Table 2.2. Example usage of derivational suffix in Turkish

Turkish	**Suffix**	**English**	**Word Type**
göz	göz	eye	Noun
gözlü	göz + lü	eyed, having an eye	Adjective
gözsüz	göz + süz	blind	Adjective
gözlük	göz + lük	eyeglasses	Noun
gözlükçü	göz + lük + çü	optician	Noun
gözlükçülük	göz+lük+çü+lük	the work of an optician	Noun

Table 2.3. Example usage of inflectional suffix in Turkish

Turkish	**Suffix**	**English**	**Word Type**
göz	göz	eye	Noun
gözler	göz + ler	eyes	Noun
gözüm	göz + üm	my eye	Noun
gözün	göz + ün	your eye	Noun
gözü	göz + ü	his or her eye	Noun
gözleriniz	göz + ler + i + niz	your eyes	Noun
gözlerimiz	göz + ler + i + miz	our eyes	Noun
gözleri	göz + ler + i	their eyes	Noun

The extensive use of suffix combinations can yield long words. For example the Turkish word “*Vedalaşamadıklarımız*” means "Those with whom we cannot say farewell to each other” in English.

The other main characteristics of Turkish other than agglutinative morphology are no gendering of words, vowel harmony and free constituent order in a sentence.

IR studies mostly focuses on English language and studies on Turkish language are less common.

Kemal Oflazer [4] described a full scale two-level morphological description of Turkish word structures. The description has been implemented using the PC-KIMMO environment and is based on a root word lexicon of about 23,000 root words. Most of the present studies still use this morphological analysis in their experiments.

Aysin Solak and Fazli Can [5] presented a stemming algorithm developed for the Turkish language. They tested the algorithm in terms of its effect on retrieval performance using 553 news articles and 71 queries.

F. Çuna Ekmekçioglu and Peter Willett [6] investigated effectiveness of stemming for Turkish text retrieval using 6,289 news stories extracted from Turkish newspapers and a set of 50 natural-language queries.

Hayri Sever and Yiltan Bitirim [7] evaluated the effectiveness of a new stemming algorithm, FINDSTEM, using 2,468 Turkish documents and 15 queries, and compared this algorithm with the other two previously defined Turkish stemmers. (One of them is developed by Aysin Solak and Fazli Can)

F. Canan Pembe and Ahmet Celal Cem Say [8] implemented and tested a linguistically motivated information retrieval system, which uses knowledge of the morphological, lexico-semantical and syntactical levels of Turkish. In their experiments, they used 615 Turkish documents from the Web and five natural-language queries.

Recently Fazli Can and Bilkent Information Retrieval Group created (Milliyet Colleciton) the first large-scale Turkish IR test collection and conducted first experiments on it. At SIGIR ’06, they examined the effectiveness of four different stemming techniques with eight query-document matching functions [9]. At SIGIR ’07 they presented cluster-based retrieval (CBR) experiments on the same test collection [10]. They made an important, positive impact on comparable Turkish IR research by making this TREC-like test collection available to other researchers [11].

Among existing studies, the SIGIR ’06 poster by Fazli Can and Bilkent Information Retrieval Group [9] is probably the most relevant to the present work, since they also used the same data set, the same evaluation measure and several stemming algorithms. We also used fixed prefix stemming at length five that they used in their work for comparison. While the aim of them is to determine which stemming algorithm and matching function provides more effective retrieval environment in Turkish, the present study covers comparison of open source information retrieval libraries and relational databases in order to determine which system is more suitable for an application that requires full-text search in Turkish language.

2.3. Full Text Search in Databases

Traditionally, relational database management systems are designed to manage structured data; IR systems are designed to manage unstructured free form text. However nowadays with the increase of unstructured data, popular Relational DBMS started to support full-text indexing and searching.

There are many studies which evaluated the performance of information retrieval libraries over Text REtrieval Conference (TREC)[8] collections. Database vendors also have evaluated their full-text search capabilities by participating in TREC competitions [12, 13].

There are publications that investigated the features and capabilities of the relational databases' full text search methods (IBM DB2 [14], Microsoft SQL Server [15], Oracle [16]).

The studies on comparison of IR systems are common, but there are no studies on Database Management Systems' information retrieval performances.

Other studies have been focused on hybrid IR–DB system solutions and integration of IR and relational databases. A systematic comparison of retrieval performances of IR libraries and relational databases has not been done.

[8] http://trec.nist.gov

3. EXPERIMENTAL SETUP

We want to show that Information Retrieval systems are more suitable than relational databases for applications that require full-text search in Turkish language.

In order to prove IR systems yield better retrieval quality and have better speed, in our retrieval experiments, we used one open source information retrieval library (Apache Lucene 2.3.2) and compared its performance to two open source relational database management systems with full-text search capabilities (PostgreSQL 8.3.1 and MySQL Server 5.0). We also have run our experiments over other popular relational database vendors such as IBM DB2, Microsoft SQL Server, and Oracle unfortunately their end-user license agreements do not allow publication of their results.

All of the experiments were completed on Dual Intel(R) Xeon(TM) 3.2 GHz machine with 2GB of RAM running Microsoft Windows Server 2003.

3.1. Linguistic Preprocessing Methods for Turkish Language

There are several pre-processing steps that can be performed during indexing to increase retrieval efficiency. Two of the well known methods are stemming and stop-word elimination.

3.1.1. Stemming Methods

For grammatical reasons, documents use different forms of a word, such as *kitap*, *kitaplar*, *kitapta* and *kitabım* [2]. Plural, gerund forms and tense suffixes are examples of variations which prevent a string match between a query and a document [17]. Additionally, there are families of derivationally related words with similar meanings, such as *demokrasi*, *demoratik*, and *demokratikleşme* [2]. In many situations, it would be useful for a search for one of these words to return documents that contain another word in the set, since a user who runs a query on "üniversite" would probably also be interested in documents that contain the word "üniversiteler".

Stemming is the process of removing inflectional suffixes (or sometimes derivational suffixes) from words in order to reduce them to a common base form. For instance:

araba, arabalar, arabam, arabası, arabada, arabalarımız $\Rightarrow$ araba

Stemming is an important issue for the enhancement of the IR performance especially for the agglutinative languages like Turkish.

We implemented and incorporated four different stemming algorithms for Turkish and used them to improve retrieval performance of the each retrieval system.

No Stemming: All words (tokens) are indexed with the out-of-the-box settings of each system without stop-word elimination. This has formed our baseline for comparison.

Snowball Stemmer: A stemmer implemented by Evren (Kapusuz) Çilden using Snowball[9], which is a small string processing language designed for creating stemming algorithms for use in Information Retrieval. Original algorithm [18] analyses Turkish words with an affix stripping approach and without using any lexicon.

Fixed Prefix Stemmer: Keeps the first *n* characters of a given term as a stem and removes the rest. Bilkent Information Retrieval Group [9] shows that using a prefix length of 5 provides an effective retrieval environment in Turkish. In our runs $n = 5$ is used.

Zemberek Stemmer: Zemberek[10] is an open source general purpose Natural Language Processing library written entirely in JAVA. We implemented a stemmer based on Zemberek's morphological analysis. Our stemmer removes the inflectional suffixes from terms.

3.1.2. Stopword List

Other well known trick, other than stemming in IR is stop word removal. Stop words are frequently used words that do not carry meaning in natural language and therefore do not help distinguish one document from other, such as

[9] http://snowball.tartarus.org

[10] http://code.google.com/p/zemberek

ve, *veya*, *de*, *da* and *ile* for Turkish. Removing extremely common stop words during indexing process significantly reduces the inverted index size on disk.

The stop word list used in this thesis contains 148 words and is given in Appendix-1, in alphabetical order.

3.2. Lucene

Apache Lucene[11] is a high performance, scalable Information Retrieval (IR) library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.

Analysis is the process of converting free form text into its most fundamental indexed representation, terms. Analysis process starts with a Tokenizer and continues with series of TokenFilters. An analyzer is an encapsulation of the analysis process. Analyzers are used for both indexing and query parsing.

For both index time and query time Lucene's sophisticated grammar-based StandardTokenizer class, which holds the honor as the most generally useful built-in Tokenizer, is used to break given free form text into tokens. Then the created token stream is fed into chains of several TokenFilters, giving the filter a chance to add, remove or change the stream as it passes through. Table 3.1 shows the Tokenizer and TokenFilter chaining pattern used in our Lucene runs.

Table 3.1. Analyzer building block

1	StandardTokenizer
2	StandardFilter
3	TurkishLowerCaseFilter
4	StopFilter
5	TurkishStemFilter

[11] http://lucene.apache.org

StandardTokenizer: Splits words at punctuation characters, removing punctuation. However, a dot that's not followed by white space is considered part of a token. Splits words at hyphens, unless there's a number in the token, in which case the whole token is interpreted as a product number and is not split. Recognizes email addresses and internet hostnames as one token.

StandardFilter: Designed to be fed by a StandardTokenizer.

TurkishLowerCaseFilter: Lowercases token text. We implemented our own TurkishLowerCaseFilter because the built-in lowercase filter that came with Lucene, fails with one character of Turkish alphabet. Although lower case of letter ‘I’ is ‘ı’ (LATIN SMALL LETTER DOTLESS I) in Turkish alphabet, Lucene’s built-in lowercase filter lowercases letter ‘I’ to ‘i’ as in English alphabet.

StopFilter: Removes words that exist in a provided set of stop-words.

TurkishStemFilter: Stems each token according to a given stemming algorithm. We implemented three types of TurkishStemFilter that applies the stemming methods explained in Section 4. To implement no stemming option we removed StopFilter and TurkishStemFilter from our analyzer building block. Totally 4 different Analyzer obtained that represent four stemming options.

Indexing: Document numbers (docno) are indexed verbatim; contents fields are indexed analyzed as described above. Four different Lucene Index are created using each stemming method.

Query Formulation: Title and description portions of the topics are directly used as an input to Lucene’s QueryParser. The same analyzer that used for indexing is used for query parsing. Default query parser operator (“OR”) is used in our runs because it yields better results than (“AND”) operator.

3.3. MySQL

MySQL has support for full-text indexing and searching however MySQL full-text search functions has no language specific linguistic support for any language (only stop-word elimination for English).

Indexing: For no stemming option, document numbers (docno) and contents were directly entered into a MySQL database table named documents. For the other three stemming options, document numbers are entered verbatim and contents are passed through analyzer that represents corresponding stemming option and then entered into a MySQL database table. Full-text indexes are built on the tables thus:

```
ALTER TABLE documents ADD FULLTEXT(contents);
```

Four different MySQL full-text index is created for each stemming method.

Query Formulation: MySQL supports three types of full-text searches:

- Natural Language Full-Text Searches
- Boolean Full-Text Searches
- Full-Text Searches with Query Expansion

Boolean full-text search mode with + operator and no operator (stands for AND, OR respectively) are also tested; however among all of these three full-text search types, natural language search yields best results.

For no stemming option, we removed punctuations from title and description portions of topic and then performed natural language search as follows:

```
SELECT docno, MATCH(contents)  AGAINST('<title>')
AS relevance FROM documents WHERE
MATCH(contents) AGAINST('<title>') LIMIT 1000;
```

For the other three stemming options, title and description portions of the topic are passed through analyzer that represents corresponding stemming option to obtain query.

3.4. PostgreSQL

PostgreSQL uses OpenFTS[12] (Open Source Full Text Search engine) which is an advanced search engine that provides online indexing of data and relevance ranking for database searching. PostgreSQL provides several predefined dictionaries for linguistic support, available for many languages, and Turkish language is one of them.

Indexing: In PostgreSQL tsvector is a data type, which represents preprocessed document, and optimized for full text search.

For the out-of-the-box settings option, document numbers (docno) and contents were entered into a PostgreSQL database table named documents. PostgreSQL provides a data type for storing preprocessed document and a function to_tsvector, which transforms document to tsvector data type.

A separate tsvector type column is created to hold the output of to_tsvector using configuration for Turkish language that comes with out-of-the-box settings of PostgreSQL.

```
ALTER TABLE documents ADD COLUMN ts_col tsvector;
UPDATE documents SET ts_col = to_tsvector('turkish', contents);
```

For the other three stemming options, document numbers are entered verbatim and contents are passed through analyzer that represents corresponding stemming option and then entered into a PostgreSQL database table.

A separate tsvector type column is created to hold the output of to_tsvector, but this time using configuration which behaves like no language is specified because data in the table already analyzed.

```
ALTER TABLE documents ADD COLUMN ts_col tsvector;
UPDATE documents SET ts_col = ('pg_catalog.simple', contents);
```

[12] http://openfts.sourceforge.net

PostgreSQL offers two kinds of indexes that can be used to speed up full text searches.

- GiST (Generalized Search Tree) based index
- GIN (Generalized Inverted Index) based index

Since GIN indexes are best for static data because lookups are about three times faster than GiST, GIN index was created to speed up the search:

```
CREATE INDEX text_index ON documents USING gin(ts_col);
```

Query Formulation: Tsquery is a data type for textual queries with support of boolean operators. PostgreSQL provides two functions to_tsquery and plainto_tsquery for converting a query to the tsquery data type.

- to_tsquery creates a tsquery value from querytext, which must consist of single tokens separated by the Boolean operators & (AND), | (OR) and ! (NOT).
- plainto_tsquery transforms unformatted text querytext to tsquery. The text is parsed and normalized much as for to_tsvector, then the & (AND) Boolean operator is inserted between surviving words.

Both of the functions tested however the results of plainto_tsquery were lower than the results of to_tsquery with OR operator.

For the out-of-the-box settings option, we tokenized title and description portions of the topics according to white spaces, removed punctuations and inserted OR operator ("|") between each token. Turkish language configuration is used for to_tsquery function. Then we submit our SQL queries as follows:

```
SELECT docno, ts_rank(ts_col, query) AS rank FROM documents,
        to_tsquery('turkish','<title>') query WHERE
      query @@ ts_col ORDER BY rank DESC LIMIT 1000;
```

For the other three stemming options, title and description portions of the topic are tokenized using the corresponding analyzer and then OR operator ("|") inserted between each token to obtain query. Simple dictionary is used as a configuration for to_tsquery function because query text is already analyzed. Then we submit our SQL queries as follows:

```
SELECT docno, ts_rank(ts_col, query) AS rank FROM documents,
  to_tsquery('pg_catalog.simple','<title>') query WHERE
     query @@ ts_col ORDER BY rank DESC LIMIT 1000;
```

PostgreSQL provides two predefined ranking functions;

- ts_rank: standard ranking function
- ts_rank_cd: cover density ranking function [19].

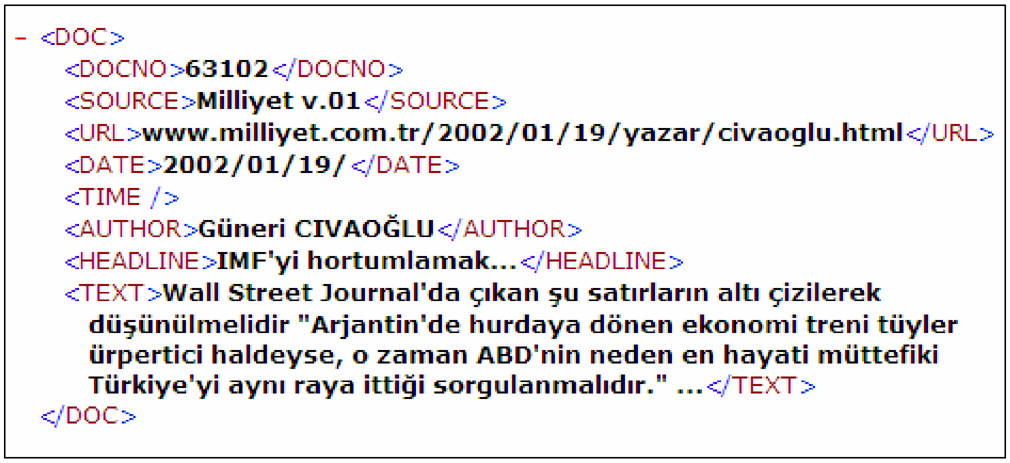
Both of the functions tested however standard ranking (ts_rank) which yields better results are used in our experiments.

3.5. Test Collection

In this study Milliyet Collection [11] was used. The collection consists of 408305 documents (news stories), 72 information needs (called *topics* in TREC) and relevance judgments, the collection is roughly 800MB in size. The collection is created and made available to the other researchers by Bilkent Information Retrieval Research Group.

3.5.1. Documents

Each document in Milliyet Collection consists of eight fields: {author, date, DOCNO, headline, source, text, time, URL}. An example of a Milliyet Collection document is shown in Figure 3.1. Among these eight fields only headline and text fields contain searchable textual information. So a new field named content is constructed, which is simply concatenation of headline and text fields. In our runs we used DOCNO as a unique identifier and content as a textual field.

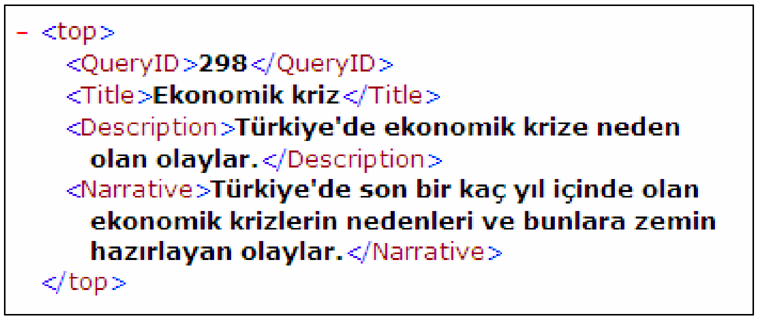


```
- <DOC>
    <DOCNO>63102</DOCNO>
    <SOURCE>Milliyet v.01</SOURCE>
    <URL>www.milliyet.com.tr/2002/01/19/yazar/civaoglu.html</URL>
    <DATE>2002/01/19/</DATE>
    <TIME />
    <AUTHOR>Güneri CIVAOĞLU</AUTHOR>
    <HEADLINE>IMF'yi hortumlamak...</HEADLINE>
    <TEXT>Wall Street Journal'da çıkan şu satırların altı çizilerek
      düşünülmelidir "Arjantin'de hurdaya dönen ekonomi treni tüyler
      ürpertici haldeyse, o zaman ABD'nin neden en hayati müttefiki
      Türkiye'yi aynı raya ittiği sorgulanmalıdır." ...</TEXT>
  </DOC>
```

Figure 3.1. Document 63102 in Milliyet collection

3.5.2. Topics

Information needs in Milliyet Collection consists of three parts (*title*, *description* and *narrative*) which are similar to a typical TREC *topic*. Title field consist a few words that best describe the topic. The description field is one or two sentence description of the topic. The narrative gives a more explanation about the topic. In the TREC terminology, each test information need is referred as a *topic*. An example of an information need in Milliyet Collection is illustrated in Figure 3.2.



```
- <top>
    <QueryID>298</QueryID>
    <Title>Ekonomik kriz</Title>
    <Description>Türkiye'de ekonomik krize neden
      olan olaylar.</Description>
    <Narrative>Türkiye'de son bir kaç yıl içinde olan
      ekonomik krizlerin nedenleri ve bunlara zemin
      hazırlayan olaylar.</Narrative>
  </top>
```

Figure 3.2. Topic 298 in Milliyet collection

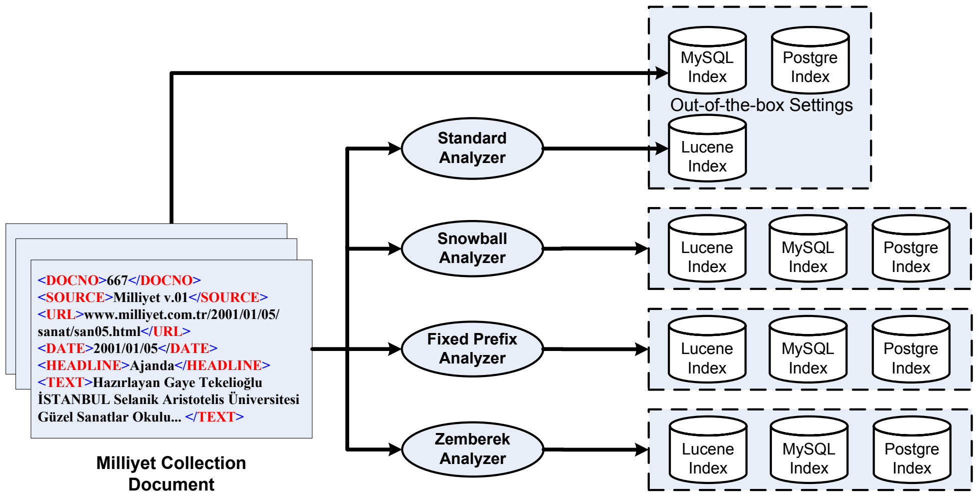
There is a distinction between a statement of information need (the topic) and the data structure that is actually given to a retrieval system (the query); however we didn't develop any special query construction technique to obtain queries from topics. We directly used title and description portions of the topics as our queries and ignored narrative portion.

3.5.3. Relevance Judgments

The relevance judgments are the "right answers" that contains the information of which documents are relevant to which topics. Since relevance judgments provided within the Milliyet Collection is created by using pooling [20] technique, it is an incomplete judgment set. Also the judgments are biased against our runs, because systems that used in this work didn't contribute anything to construction process of the pool of judged documents.

3.6. System Overview

Figure 3.3 summarizes the index creation process and Figure 3.4 summarizes the query construction and searching process, for each system with four different stemming options.



<DOCNO>667</DOCNO>
<SOURCE>Milliyet v.01</SOURCE>
<URL>www.milliyet.com.tr/2001/01/05/sanat/san05.html</URL>
<DATE>2001/01/05</DATE>
<HEADLINE>Ajanda</HEADLINE>
<TEXT>Hazırlayan Gaye Tekelioğlu İSTANBUL Selanik Aristotelis Üniversitesi Güzel Sanatlar Okulu... </TEXT>

Figure 3.3. System overview of indexing

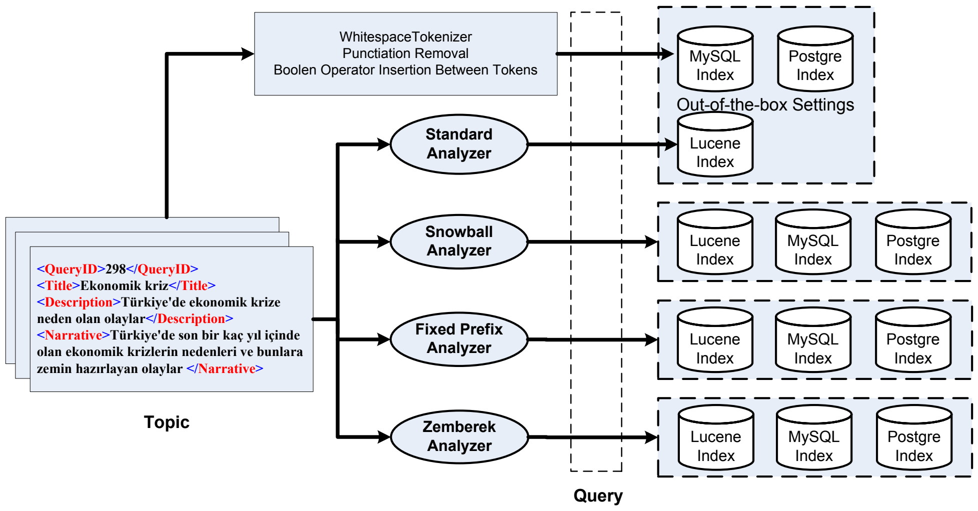

WhitespaceTokenizer
Punctiation Removal
Boolen Operator Insertion Between Tokens
MySQL Index
Postgre Index
Out-of-the-box Settings
Lucene Index
Standard Analyzer
Snowball Analyzer
Fixed Prefix Analyzer
Zemberek Analyzer
Lucene Index
MySQL Index
Postgre Index
<QueryID>298</QueryID>
<Title>Ekonomik kriz</Title>
<Description>Türkiye'de ekonomik krize neden olan olaylar</Description>
<Narrative>Türkiye'de son bir kaç yıl içinde olan ekonomik krizlerin nedenleri ve bunlara zemin hazırlayan olaylar </Narrative>
Topic
Query

Figure 3.4. System overview of searching

4. EXPERIMENTAL RESULTS

The choice of an evaluation measure is crucial since it directly affects the effectiveness of a retrieval system. Many of the traditional evaluation measures frequently used in information retrieval are derived in some way from recall and precision. (Precision is the proportion of retrieved documents that are relevant, while recall is the proportion of relevant documents that are retrieved). These evaluation measures are unable to deal with the problem of incomplete relevance data.

At SIGIR ’04, Chris Buckley and Ellen Voorhees [21] introduced a new evaluation metric called binary preference (*bpref*) which has been designed for comparing systems over test collections with incomplete relevance judgments, like the Milliyet Collection.

The *bpref* metric calculates a preference relation of whether judged relevant documents are placed before judged irrelevant documents in the ranked result list. Thus, it uses relative ranks of judged documents and ignores un-judged documents. In this thesis we initially concentrated on *bpref* measure to evaluate the effectiveness of different retrieval systems.

The evaluation measures presented in this thesis are calculated by using Chris Buckley’s trec_eval[13] package (version 8.1) which is the standard tool used by the TREC community for evaluating an ad hoc retrieval run, given the results file and a standard set of judged results.

In our calculations a cut-off level of 1000 is used, which defines the retrieved set as the top 1000 documents in the ranked list which is similar to official TREC usage:

```
trec_eval -c -M1000 official_qrels submitted_results
```

All retrieval systems are designed in a similar fashion to standard TREC-type ad hoc runs that retrieve maximum 1000 documents per topic.

[13] http://trec.nist.gov/trec_eval

We ran retrieval experiments over each system with four stemmers by using title-only queries. Title-only queries had on average 3 terms. We did another set of experiments with title & description queries (average length 16 terms) to measure the scalability of the matching algorithm each system uses. Our experiments consist of three systems, four stemming methods and two types of query lengths. We present the results of total 24 runs.

Table 4.1 shows the performances of each system with four stemmers obtained by using title-only queries. In terms of *bpref* values of title-only queries, PostgreSQL is the best performing one with out-of-the-box settings because it has a built-in Turkish snowball template dictionary that performs stemming and stop-word removal. PostgreSQL's standard stop-word lists are also provided by the Snowball[14] project. With linguistic preprocessing best performing system is Lucene with Zemberek stemmer.

Table 4.2 shows the performances of each system with four stemmers obtained by using title & description queries. In terms of bpref values of title & description queries, Lucene is best performing one with all settings. And also the overall winner for both title-only and title & description queries is also Lucene with Zemberek stemmer.

Table 4.1. Bpref values from title-only queries

	No Stem	Snowball	Fixed Prefix	Zemberek
Lucene	0.3505	0.4313	0.4413	**0.4506**
MySQL	0.2899	0.3388	0.3637	0.3644
PostgreSQL	0.4249	0.4187	0.4308	0.4389

[14] http://snowball.tartarus.org

Table 4.2. Bpref values from title & description queries

	No Stem	Snowball	Fixed Prefix	Zemberek
Lucene	0.3947	0.4867	0.5067	**0.5124**
MySQL	0.3019	0.3808	0.4139	0.4140
PostgreSQL	0.2672	0.2932	0.2808	0.2660

We observed that increase in query length, improved the retrieval performance (*bpref*) of all systems except PostgreSQL. However improvement obtained by increase in query length in relational databases caused impractical searching times.

We also observed that stemmers also improved the retrieval performance of all systems. However the improvement of Lucene is greater than the relational databases.

In our experiments, the most effective stemmer is Zemberek because all systems received their maximum *bpref* values with Zemberek Stemmer. That's why we also wanted to present the best representative of each system's interpolated recall - precision curve which is one the most commonly used method for comparing systems.

Eleven point recall-precision graphs of each system are presented for title-only queries on Figure 4.1, for title & description queries on Figure 4.2.

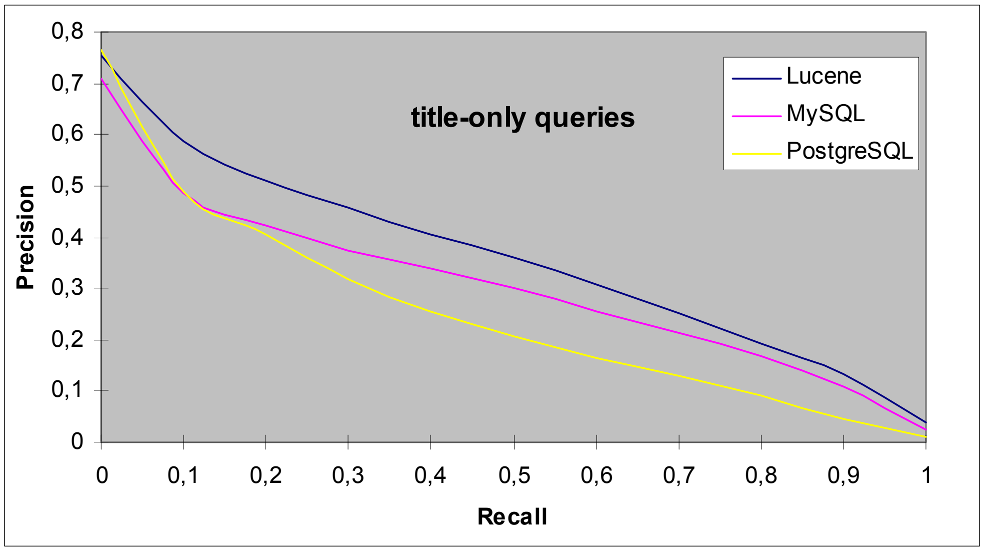


Figure 4.1. Interpolated precision - recall graph

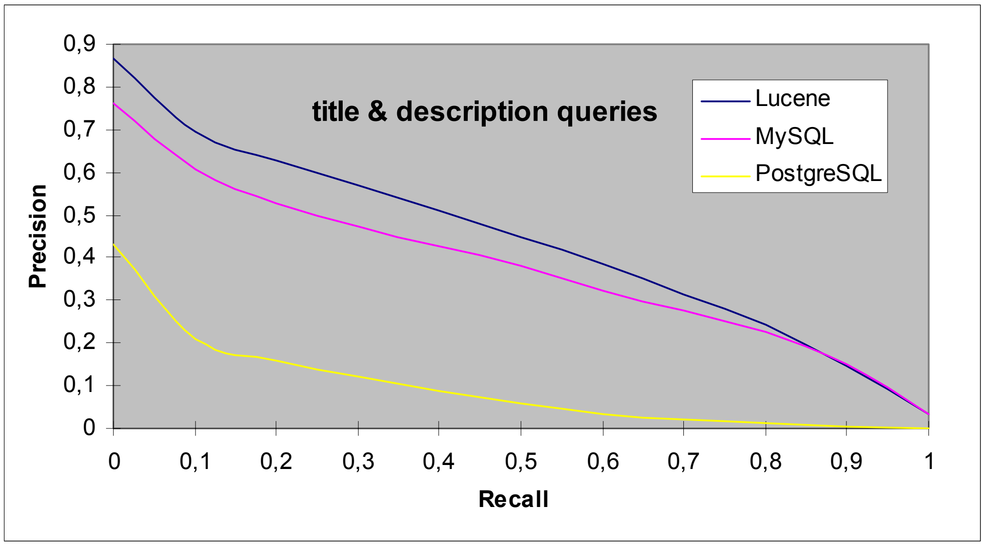


Figure 4.2. Interpolated precision - recall graph

The plots of each system for two different runs (title-only queries and title & description queries) are superimposed on the same graph to determine which system is superior.

For both graphs, Lucene whose curve is closest to the upper right-hand corner of the graph (where recall and precision are maximized) indicate the best performance.

"Comparisons are best made in three different recall ranges: 0 to 0.2, 0.2 to 0.8, and 0.8 to 1. These ranges characterize high precision, middle recall, and high recall performance, respectively" [22]. Lucene is the best performing one in all three recall ranges.

On Figure 4.3, we present a graphical comparison of the average searching times of best representative of each search engine for title and title & description queries.

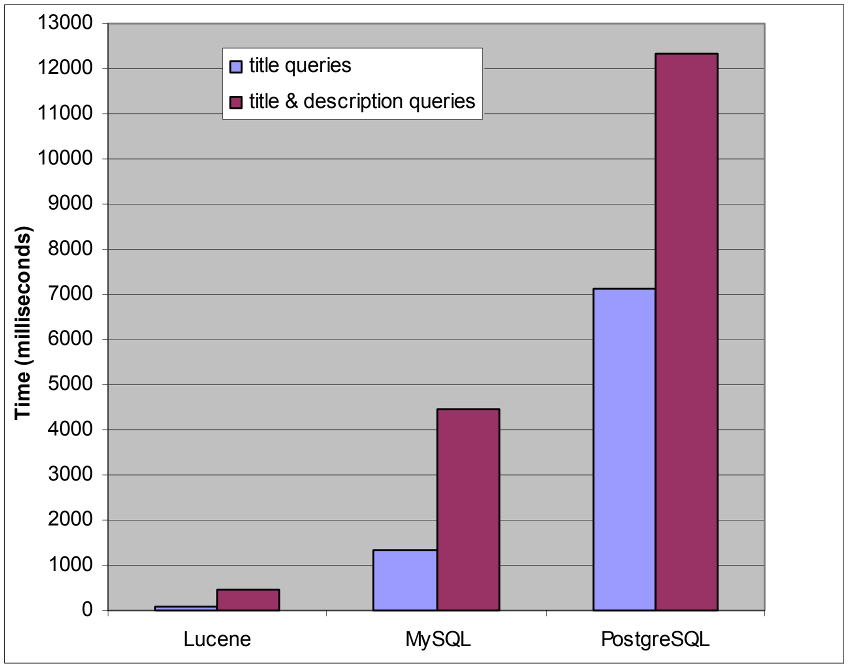


Figure 4.3. Average searching times

We observed that there is big searching time difference between Lucene and relational databases. Also we observed that searching times of relational databases with out-of-the-box settings is three times slower with title-only queries, eight times slower with title & description queries than stemmed and stop word eliminated settings. That's because without stop-word elimination and stemming the term document matrix consists over 500 Giga entries and the execution of the ranking algorithm that relational databases use becomes very slow. Table 4.3 gives the number of unique terms (in content field only) in Lucene Index of each stemming method. Most aggressive stemmer is fixed prefix stemmer.

Table 4.3. Number of terms in each Lucene index

Stemming Method	**Number of unique terms**
No Stemming	1,346,183
Snowball Stemmer	801,548
Zemberek Stemmer	597,658
Fixed Prefix Stemmer	262,347

On Figure 4.4, we present a graphical comparison of indexing times of each search engine with out-of-the-box settings.

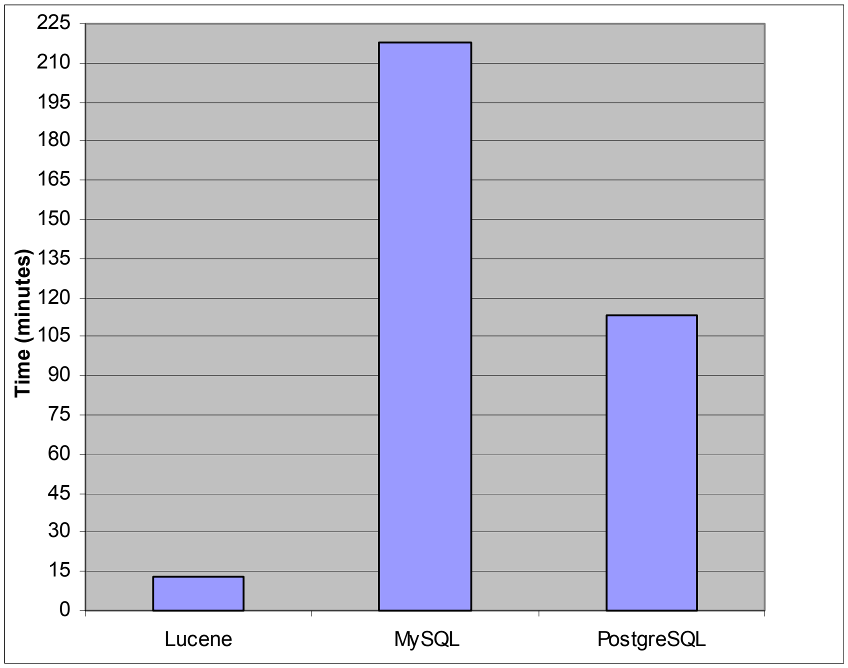


Figure 4.4. Average indexing times

Although relational database settings are optimized for static data in our experiments, their indexing times are much slower than Lucene. When it comes to the real world, collections are dynamic. New documents are added frequently. Relational databases indexes are created after the insertion of the all documents. When it comes to incremental indexing their executions times would become much worse. Lucene is very robust to handle inverted index updates.

Additionally we observed that in each search engine, Boolean query operator OR yields better retrieval quality than AND operator for this evaluation data set.

5. CONCLUSIONS AND FUTURE WORK

This thesis covers the comparison of Turkish text retrieval performances of relational databases versus Lucene - which is an open source information retrieval library - and presents the results obtained after performing tests with different stemming options and different query lengths. Execution times of each system during indexing and searching are also compared.

We show that information retrieval library with language specific improvements performed best effectively and efficiently for the Turkish evaluation set. The results of Lucene with Zemberek stemmer presented in this thesis are best results among the published works on this data set up to now. Lucene with Zemberek stemmer showed over 55% improvement over relational databases without any language specific improvements. And Lucene's execution time during indexing is ten times, during searching is fifty times better than relational databases.

Although relational databases provide easy to use full text search capabilities, without linguistic preprocessing their performance is quite low. Linguistic preprocessing improves the performance on relational databases. Also ranking algorithms in information retrieval library is more robust to increase in query length.

We will expand our experiments to compare relational databases and information retrieval systems to English and other languages; also we will add more open source retrieval libraries to our set.

REFERENCES

[1] Middleton C. Baeza-Yates R. A comparison of open source search engines; available at http://wrg.upf.edu/WRG/dctos.

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

[3] Salton, G., Buckley, C. Term weighting approaches in automatic text retrieval. IPM 24, 513-523, 1988.

[4] Oflazer, K. Two-level Description of Turkish Morphology, Literary and Linguistic Computing, **9**(2), 137–148, 1994.

[5] Solak, A., Can, F., Effects of stemming on Turkish text retrieval. ISCIS Conf., pp. 49-56, 1994.

[6] Ekmekçioğlu, F.C., & Willett, P. Effectiveness of stemming for Turkish text retrieval. Program, **34**(2), 195–200, 2000.

[7] Sever, H., Bitirim Y. FindStem: analysis and evaluation of a Turkish stemming algorithm. LNCS 2857: 238-251, 2003.

[8] Pembe, F.C., & Say, A.C.C. A linguistically motivated information retrieval system for Turkish. Lecture Notes in Computer Science, 3280, 741–750, 2004.

[9] Can F., Kocberber, S., Balcik E., Kaynak C., Ocalan H. C., Vursavas O. M. First large-scale information retrieval experiments on Turkish texts. In SIGIR 2006, 627-628, 2006.

[10] Altingovde S. I., Ozcan R., Ocalan H. C., Can F., Ulusoy O. Large-scale clustered-based retrieval experiments on Turkish texts. In SIGIR 2007, 891-892, 2007.

[11] Can F., Kocberber, S., Balcik E., Kaynak C., Ocalan H. C., Vursavas O. M. Information retrieval on Turkish texts. Journal of the American Society for Information Science and Technology, **59**(3):407–421, 2008.

[12] Mahesh K., Kud J., and Dixon P., Oracle at Trec8: A Lexical Approach, in Proceedings of the Eighth Text Retrieval Conference (TREC-8), 1999.

[13] Alpha S., Dixon P., Liao C., and Yang C., Oracle at TREC10. Notebook paper, 2001; available at http://trec.nist.gov/pubs/trec10/papers/

[14] Maier A. and Simmen D. DB2 optimization in support of full text search. IEEE Data Engineering Bulletin, **24**(4):3–6, 2001.

[15] Hamilton J. and Nayak T. Microsoft sql server full-text search. IEEE Data Engineering Bulletin, **24**(4):7–10, 2001.

[16] Dixon P. Basics of Oracle text retrieval. IEEE Data Engineering Bulletin, **24**(4):11–14, 2001.

[17] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. Modern Information Retrieval. Addison-Wesley, Wokingham, UK, 1999.

[18] Eryiğit G. and Adalı, E. An Affix Stripping Morphological Analyzer for Turkish, Proceedings of the IASTED International Conference Artificial Intelligence and Applications pp. 299-304, Innsbruck, 16-18 February 2004.

[19] Clarke, C. L., Cormack, G. V., and Tudhope, E. A. 2000. Relevance ranking for one to three term queries. Inf. Process. Manage., **36**(2):91–311, 2000.

[20] Spärck Jones, K., Van Rijsbergen, C. J. Report on the need for and provision of an "ideal" information retrieval test collection. British Library Research and Development Report 5266, Computer Laboratory, University of Cambridge, 1975.

[21] Buckley, C. and Voorhees, E. M. Retrieval evaluation with incomplete information. ACM SIGIR Conf, pp. 25-32, 2004.

[22] The Fifteenth Text REtrieval Conference (TREC 2006) Proceedings: Appendix - Common Evaluation Measures.

Appendix-1 A Turkish Stop Word List with 148 words

ama	burada	henüz	neden	rağmen
ancak	çok	her	nedenle	sadece
arada	çünkü	herhangi	o	siz
ayrıca	da	herkesin	olan	şey
bana	daha	hiç	olarak	şöyle
bazı	de	hiçbir	oldu	şu
belki	değil	için	olduğu	şunları
ben	diğer	ile	olduğunu	tarafından
beni	diye	ilgili	olduklarını	üzere
benim	dolayı	ise	olmadı	var
beri	dolayısıyla	işte	olmadığı	vardı
bile	edecek	itibaren	olmak	ve
bir	eden	itibariyle	olması	veya
birçok	ederek	kadar	olmayan	ya
biri	edilecek	karşın	olmaz	yani
birkaç	ediliyor	kendi	olsa	yapacak
biz	edilmesi	kendilerine	olsun	yapılan
bize	ediyor	kendini	olup	yapılması
bizi	eğer	kendisi	olur	yapıyor
bizim	etmesi	kendisine	olursa	yapmak
böyle	etti	kendisini	oluyor	yaptı
böylece	ettiği	ki	ona	yaptığı
bu	ettiğini	kim	onlar	yaptığını
buna	fakat	kimse	onları	yaptıkları
bundan	gibi	mı	onların	yerine
bunlar	göre	mi	onu	yine
bunları	halen	mu	onun	yoksa
bunların	hangi	mü	oysa	zaten
bunu	hatta	nasıl	öyle	
bunun	hem	ne	pek	