IMPROVING SEARCH ENGINE PERFORMANCE WITH CONTEXT EXTRACTION USING LUCENE, DBPEDIA-SPOTLIGHT, AND WORDNET

REMZİ DÜZAĞAÇ
B.S., Computer Engineering, Işık University, 2010

Submitted to the Graduate School of Science and Engineering
in partial fulfillment of the requirements for the degree of
Master of Science
in
Computer Engineering

IŞIK UNIVERSITY
2014

IŞIK UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

IMPROVING SEARCH ENGINE PERFORMANCE WITH CONTEXT EXTRACTION USING LUCENE, DBPEDIA-SPOTLIGHT, AND WORDNET

REMZİ DÜZAĞAÇ

APPROVED BY:

Assoc. Prof. Olcay Taner Yıldız (Thesis Supervisor)	Işık University	_____
Assist. Prof. Ali İnan	Işık University	_____
Assist. Prof. Arzucan Özgür	Boğaziçi University	_____

APPROVAL DATE: ..../..../....

IMPROVING SEARCH ENGINE PERFORMANCE WITH CONTEXT EXTRACTION USING LUCENE, DBPEDIA-SPOTLIGHT, AND WORDNET

Abstract

Search engines are common tools which retrieve information from considerable amount of data according to the user needs. The data size that needs to be handled and retrieving relevant information, are the main problems of every search engine. Additionally, in order to improve the performance of a search engine, there are various approaches and methods are applied. On the other hand, using context information besides words in the document is a quite new area. Including "Context Information" into the game is a promising field of work.

In this research, we use context information extracted from the documents in the collection to improve the performance of the search engine. In first step, we extract context using Lucene, DBPedia-Spotlight, and Wordnet. As the second step, we build a graph using extracted context information. In the third step, in order to group similar contexts, we cluster context graph. In the fourth step, we rescore results using context-clusters and context-information of documents, as well as queries. In the fifth step, we implement a data collection tool to collect gold-standard data. In the sixth and final step, we compare the results of our algorithm with gold-standard data set. According to experimental results, using context information may improve the search engine performance but the collection should be relatively big.

ARAMA MOTORU PERFORMANSININ SOLR, DBPEDIA-SPOTLIGHT ve WORDNET KULLANILARAK YAPILAN BAĞLAM ÇIKARIMI İLE ARTIRILMASI

Özet

Arama motorları, kullanıcıların ihtiyaçlarına göre ilgili bilgileri kayda değer miktarda veri içerisinden sunan araçlardır. İşlenmesi gereken verinin büyüklüğü ve ilgili bilgileri kullanıcıya sunmak arama motorlarının iki ana problemini oluşturur. Arama motoru performansını artırmak için pek çok yaklaşım ve metod bulunmaktadır. Bunlara ek olarak arama motorlarının performansını artırmak için dökümanın içerdiği kelimelerin yanında bağlam bilgisini kullanmak oldukça yeni bir alan. Oyuna Bağlam Bilgisini dahil etmek gelecek vaat eden bir çalışma alanı sunmakta.

Bu çalışmamızda, arama motoru performansını artırmak için döküman ve sorgulardan çıkardığımız bağlam bilgisini kullanıyoruz. İlk adım olarak Lucene, DBPedia-Spotlight ve Wordnet'i kullanarak bağlam bilgisi çıkarıyoruz. İkinci adımda, çıkardığımız bağlam bilgilerini kullanarak bir çizge oluşturuyoruz. Üçüncü adımda, birbirine yakın bağlamları gruplamak için çizge üzerinde kümeleme yapıyoruz. Dördüncü adımda, döküman ve sorguları bağlam çizgesini ve ilgili bağlam bilgilerini kullanarak sonuçları yeniden puanlıyoruz. Beşinci adım olarak referans verisi toplamak için bir uygulama geliştirip bu uygulama ile kullanıcılardan veri topluyoruz. Altıncı ve son adımda ise kullanıcılardam topladığımız referans bilgisi ile sonuçlarımızı karşılaştırıp yaptığımız çalışmanın performansını ölçüyoruz. Aldığımız sonuçların bize gösterdiğine göre bağlam bilgisini kullanmak arama motorlarının performansını artırabilir ancak kullanılacak döküman kümesi göreceli olarak büyük olmalı.

Acknowledgements

There are many people who helped me make my years at the graduate school most valuable. First, I would like to express my deepest acknowledgements to my supervisor Assoc. Prof. Olcay Taner Yıldız for his support and guidance throughout my thesis work. I am also thankful to Işık University for providing me all the facilities and for being a part of such a wonderful environment. Finally, I will always be indebted to my fiancée Merve and my family for their patience and encouragement.

To my family. . .

Table of Contents

List of Figures

List of Tables

List of Abbreviations

IR	Information Retrieval
PC	Personal Computer
WLC	The Wesbury Lab Wikipedia corpus (2010)
LCN	Apache Lucene
DBPD	DBPedia Spotlight
WNDB	Wordnet
LCA	Lucene Context Analyzer
DSCA	DBPedia Spotlight Context Analyzer
WNCA	Wordnet Context Analyzer
HDFS	Hadoop File System
EMR	Amazon Elastic MapReduce
MCL	Markov Cluster Algorithm

Chapter 1

Introduction

1.1 Motivation

Today we are living in a digital world and producing enormous data using various sources such as computers, mobile phones, tablet PCs etc. Between 2000 and 2012, the growth rate of Internet usage was 566.4% [1] Producing and storing large quantities of data make accessing worthy information quite a challen.

Capability of processing information of the human brain may provide a viable basis for new approaches. In order to process large amount of data, the human brain generates relationships between the current sensory information that represents the external world and previously stored information. This relationship allows human brain to determine actions. Neuroscience professor Rodolfo Llinas, describes in his book[2], "I of the vortex: From Neurons to Self", how the human brain works as follows: "We can look to the world of neurology for support of the concept that the brain operates as a closed system, a system in which the role of sensory input appears to be weighted more toward the specification of ongoing cognitive states than toward the supply of information – context over content."

The motivation of our study is to develop an approach to improve the performance of search engines by adopting the "context over content" approach of the human brain.

1.2 Information Retrieval

Information Retrieval (IR)is the study of retrieving relevant information from considerable amount of data according to the user needs. IR system consists of four main parts; crawling, indexing, querying and retrieving. For the each part, applied method and approach can vary according to the type of data to be handled. For example, retrieving articles of a newspaper from archives and fetching images from an album requires different methods and approaches. Despite of the diversity of methods, purpose of every IR system is retrieve relevant information for the user.

Search engines are applications of information retrieval as computer programs. They execute crawling, indexing, querying and retrieval steps. Web search engines are specialized search engines designed for the web. Web search engines crawl Internet, index and serve the results to the user through its user interface. Today there are several web search engines in use such as Google, Yandex, Yahoo etc.

First part of the search engine is crawling. Crawling is a step of data gathering from various sources, the most common source being Internet. Crawlers download documents and prepare them for the indexing process.

Indexing is the second part of a search engine. During the indexing step, indexers analyze the document collection to determine how they should be indexed and extract their contents that will be indexed. After content extraction, indexers prepare indexes to retrieve relevant documents rapidly.

Third part of a search engine is querying. Queries are formal statements used for requesting information from search engines. Search engines analyze queries and reply with the most relevant document list.

Retrieving documents is the fourth part of a search engine. Search engines select relevant documents from a document collection. In order to retrieve relevant documents, indexers assign scores to documents by using various parameters such as

zone, date, page rank etc. In addition, search engines may assign additional scores with respect to the queries. After the scoring step completed, search engines order result set with respect to their scores. In information retrieval, ordering the results of a query by using various scores is called ranking.

1.3 Context Sources

In our work, we use three diffrent sources to extract context information from documents. These sources are: Lucene, DBPedia Spotlight, and Wordnet.

Lucene [3] project of Apache Foundation [4] is the First source we used to extract context information. Apache Lucene is an open-source, high-performance, full-featured text search engine library written entirely in Java. It is a technology that is suitable for nearly any application that requires full-text search.

DBPedia Spotlight[5][6] is the second source we used. DBPedia[7][8] is an open source project to extract structured information from Wikipedia and it publishes the results on the web. DBPedia Spotlight is another open source project which automatically annotates DBPedia entities in texts. It provides a programming interface for spotting annotations and entity linkings.

Wordnet[9] the third source. Wordnet is a lexical and semantic database in English. It has been developed by the Cognitive Science Laboratory in Princeton University in 1985 and maintained from there since then. It provides conceptual-semantic and lexical relations between nouns, verbs, adjectives, and adverbs.

1.4 Problem

Search engines deal with the measurement of how close the source information matches with the user input; thus retrieving the most relevant information to the users. In order to calculate the relevancy, search engines use several parameters such as the popularity of a document, the date of a document, user preferences

extracted from usage logs etc. In addition to these parameters, some search engines also provide an auto-suggestion functionality to its users, based on a generated search history of similar inputs, which proposes a complete query of what the user might be searching for. How these parameters work depend on the words that constitute the search query. Therefore, the search engines generally are not able to consider the "meaning" of the input, in other words, it can not differentiate the context that the search input is related to. This is why the user sometimes might see search results that seem completely out of context comparing to what they searched for. This important problem is the main focus of this thesis.

How human brain operates with neurons has been the key concept for developing the approach to this problem. According to Llinas[2], the human brain has evolved to execute movement, which means that enabling the human to build interactions between its own sense of self and its external environment. Briefly, with the help of the sense organs, the brain generates sensorimotor images which represent the external world. By doing this, it can predict what to feel or how to move during an encounter with real-time sensory information, by comparing this external information with the internal sensorimotor image. Metaphorically speaking, prediction is a state of mind, where the brain acts as a search engine as it goes through internal sensorimotor images and comes up with a prediction that will determine the human reaction to the external world. This prediction can be observed as a search result. The internal images in this process are used in relation to the respective context that they belong to. In other words, "the significance of sensory cues is expressed mainly by their incorporation into larger, cognitive states and entities."[2]

This process in the brain provides us an inspiring idea about how the search engines might be improved by adopting the "context over content" approach. This is what this thesis contribute to the field of search engines: adding context-based search capability (retrieving documents that do not only contain same words with the search query but also belonging to similar contexts). Instead of focusing on finding the exact content, we take the context from the input

information into account and improve the relevancy rate of the results and thus the overall performance of the search.

1.5 Our Solution

In order to implement the context over content approach, first context information from documents and queries are extracted. In the scond step, in order to be able to find how many documents and queries are interrelated, extracted information is clustered. As a final step, the clustered context information is used to calculate new ranking scores to improve search engine efficiency.

The most important part of the proposed approach is to extract the context from the documents. In this part, we use Lucene, Wordnet, and DBPedia Spotlight as the context information sources. After the context extraction, we pair these extracted context data. Following the pairing step, we count pairs to find how many document contain the same context pairs. This count gives us an idea about the strength of the relationship existing between pairs.

As a second step, the pairs are clustered according to their counts. These clusters form contexts which groups the related documents together. After the clustering step, we label the documents with clusters. In addition to document labeling, queries are labeled with clusters as well. Finally, the results are scored according to the correlation between the query and document clusters.

This thesis is organized as follows:

- Chapter 2 reviews similar works in the area.
- chapter 3 describes how the proposed system work.
- chapter 4 explains experiment step and gives result of the experiment.
- Chapter 5 is the conclusion.

Chapter 2

Related Works

Improving search engine is a challenging problem. There are various approaches and a wide range of studies for each approach. However, using context knowledge, especially using Wordnet and Wikipedia as context sources, is relatively new area. Still, there are various studies on using context information. we review some of them in two groups. First group (A New World: Contexts and Concepts) is general approaches such as using semantic distance or context-aware retrieval. Second group (Wikipedia and Wordnet: Two New Actors in the game) reviews some studies which uses Wikipedia and Wordnet to improve a search engine performance.

2.1 A New World: Contexts and Concepts

2.2 A New World: Contexts and Concepts

In the study of Brown et al [10], Context Aware Retrieval (CAR) systems and its relationships to information retrieval and filtering are surveyed. They claim that CAR is a sparsely researched area and there are some topics of information retrieval and filtering which is not applied to CAR systems. The major difference between IR/IF and CAR applications is that CAR application contexts are gradually (continuous) changing. They list types of CAR applications as interactive/proactive, user-driven/author-driven. In user driven CAR applications,

sensors and user requests are evaluated and information is retrieved accordingly whereas in author driven applications, each document has a triggering context defined by the author and whenever user enters that context, the document becomes available for serving.

In this paper [11], two fields of information retrieval are brought together to present a reliable knowledge-based information extraction method: personalization of the information and retrieval context. This paper states that, this combination provides a unique approach towards the process of measuring relevance, since it introduces the concept of ontological information domain which refers to the enriched representational ground for content meaning, user interests, and contextual conditions (Mylonas et al. [11]). Therefore *relevance* is not just a measurement of similarity or difference between the content of the documents, but also it is about the relationship between the users history and the results as well. This is why this paper proposes a profiling method that ties users history with a set of contexts. With the use of fuzzy logic and clustering on the user behavior, user types can be extracted. This fuzzy representation overcomes the imprecision and uncertainty problems and in order to improve IR results, user data should be involved in the retrieval process.

In [12], authors make use of semantic distance for feature-free search query classification. They show advantages of utilizing the number of search results while grouping the web content according to their relevance. In contrast to other ML-based methods, they use a feature-free approach since it's a better fit for ever-changing web data. They use Naive Ranking Strategy, Normalized Google Distance, Jaccard, Overlap, Dice and Point-wise Mutual Information to map the query with a category and use the best ones out of every result.

In the study of [13], Concept-Based Information Retrieval is performed using explicit semantic analysis. They perform text content processing which is criticized by the previous article. While the previous research we discussed focused on the number of search results while not developing a full-fledged retrieval model,

this research rather suggests that a representation method would be a reliable way to solve the inaccuracy problem in the information retrieval field. Their concept-based approach (which seems similar to our context over content approach) involves three important contributions: using Explicit Semantic Analysis which refers to *real-life concepts resembling human perception*, integrating selection phase into concept-based indexing stage, and using three AI-based selection methods for information retrieval. Their work is innovative since they shift the focus from keyword and indexing back to the capability of generating contexts based on human concepts.

Atanas Kiryakov et al [14] introduces an architecture for semantic annotation, indexing and retrieval of documents via semantic resources. Their architecture claims to provide automatic semantic annotation with references to ontology classes and instances. They perform semantic indexing and retrieval where they combine ontology and information retrieval methods. Their platform called KIM has been developed for this purpose. Although they can not evaluate their results precisely due to lack of test/data or existing metrics for semantic annotation and retrieval, they conclude that using semantic entity knowledge for semantic annotation is worthwhile where entities can be stored in RDF repositories. Their prototype shows possible future tools that take advantage of entity databases.

In the study of (Mukhopadhyay et al[15]), they build a domain-specific graph-based search engine and use ontology for detecting the domain-relevance. The nodes in the graphs are the domains (topics) and edge weights are the number of mutual pages in both domains. High-weight edges represent related topics. This way, search results can be extended on topic similarity. They incorporate multiple domains in the same graph and discard links in a page that is on a domain out of interest. They use a seed list containing URLs of specific interest domains. For detecting if a webpage can classified in a domain cluster, they perform relevance calculation using WordNet and ontologies and apply a threshold.

2.3 Wikipedia and Wordnet: Two New Actors in The Game

Question answering systems take part rather than classical search engines, for providing answers to the users' queries with better quality. Question classification module has a significant role in question answering systems. The module maintains prior knowledge to the users' expectations when building a query with clear form. In the literature, imprecise question classification is pointed as a very important aspect of question answering systems with poor performance. In the area of question answering, judging the preciseness of the answer is crucial. In this paper [16], a novel question classification module is designed for bringing out the semantic features of the Wordnet and Wikipedia. To accommodate constantly changing open-domain question answering systems, the algorithm has dynamic and expendable properties. The method uses natural language processing (NLP) techniques along with world wide web. This combination results in a 2-folded structure. NLP spots the users' expectations for a specific query and world wide web offers various semantic resources such as Wikipedia and the Wordnet. The experiments were done with 5500 questions from a standard set and these are tested for 5 question collections from TREC datasets. According to the results, the proposed approach outperforms their equivalent methods from the literature with a 89.55% of accuracy. The effectiveness of the proposed method is promising for future attempts in the field of open-domain question answering.

This paper [17] studies whether lexical resources can be applied to increase the diversity level for one word, ambiguous queries. For this approach, two sense inventories of two different paradigms have been used: Wikipedia and the Wordnet. These two are compared and some conclusions are obtained. According to the results, Wikipedia has broader coverage of results with its internal graph structure and the relative number of visits taken by each sense. With simple and efficient algorithms, 70% of more Wikipedia senses covered from modified rankings. These experiments inspire further researches to use Wikipedia to construct search results. There are limitations of this research that should be noted; due

to the nature of the testbed, strong conclusions cannot be made for web searches and there are no study for diversity from the web users' point of view.

This article [18] states a knowledge based approach to web search result clustering and try to solve related research topics focusing on clustering short texts from the web. This information retrieval task is analyzed if it can take advantage from DBpedia spotlight state-of-the-art entity linking system. The approach uses DBpedia concepts as text seeds to collect topical concept labels. These labels are used to cluster based on their topical similarity. This paper aims at several key questions: (i) whether we can use DBpedia (ii) check if we benefit from topical information to capture semantic similarity. The approach takes web search snippets as inputs and cluster them topically using DBpedia. The results show that, clustering compact topically semantified representations achieved competitive quality for this method. The viability of using knowledge based methods outperform the bag-of-words model. For future studies, structured representations, semantic graphs and multilingual dimension of DBpedia are planned to be explored.

Chapter 3

System

3.1 System Overview

The system that we designed to implement the context over content approach consists of five different steps: corpus, context extraction, context clustering, labeling and scoring. First step is corpus, which contains preprocessed documents and queries for next steps. Second step is the context extraction. In this part, we use Lucene, DBPedia Spotlight, and Wordnet to extract context information from documents and queries. The third step is the context clustering. We cluster extracted contexts to form groups which contain similar contexts.Fourth step is labeling. In the labeling step, we use context information extracted from each document and clusters to determine the clusters that each document is related to. Also we label queries in the same manner with documents. Fifth and final step is scoring. In the scoring step, we calculate scores according to the correlations between queries and documents.

3.2 Corpus

A corpus is a collection of texts in digital form, over which processes such as retrieval or analysis will performs.[19] In this thesis, we use The Westbury Lab Wikipedia Corpus (WLC)[20], as our data source.

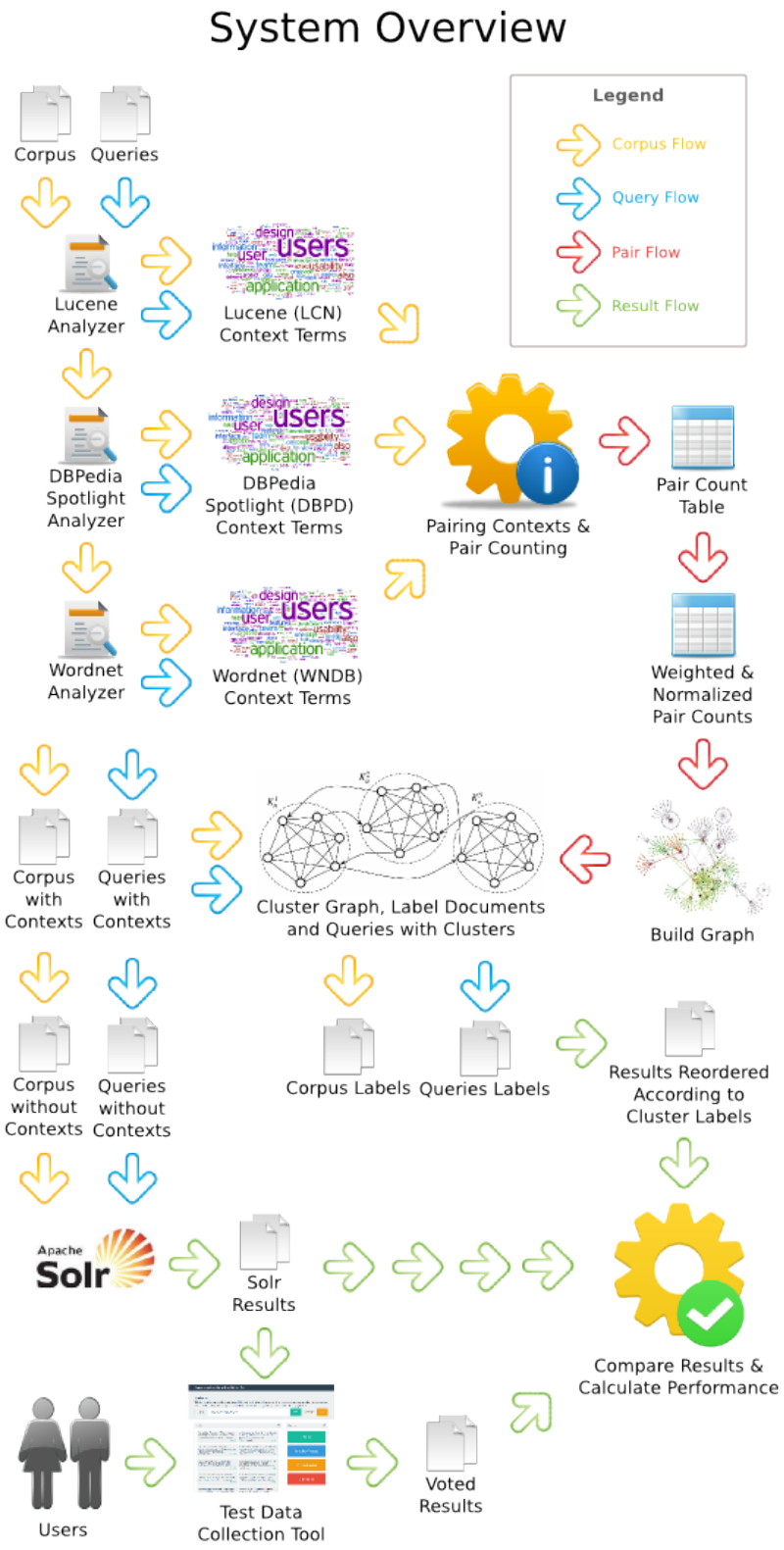

Figure 3.1: System Overview

WLC is a snapshot of all the articles in the English part of the Wikipedia that was taken in April 2010. It was processed to remove all links and irrelevant material (navigation text, etc) Afterwards, it only includes untagged raw text. WLC contains 990,248,478 words, over 2 million documents. The size of the file is 6GB in raw text format and 1.8GB in compressed bzip format.[20]

We apply some preprocessing steps to WLC in order to obtain a corpus which fits our needs. First preprocessing step is splitting the raw WLC corpus file. Since we need documents that are correlated with queries, we split WLC corpus into separate documents in a way that each document contains only one article.

Next preprocessing step is selecting documents. It would not be possible to use all the documents obtained from the WLC file, due to our limited computational power. Therefore, we randomly select 10000 documents from generated files. Since the average word count of Wikipedia articles is around 300 and complexity of the pairing process of context extraction part is $O(n^2)$, we decide to set the limit of each document size between 300 and 1000 words. In order to apply this limit to each document, we delete documents which have less than 300 words and split documents which has more than 1000 words into smaller documents. After this selection step, there are 8000 documents left.

Queries are the questions that are entered into search engine to retrieve results. In order to select queries, first we select "n" documents ($n \leq 30$) randomly. Then we select a portion of each document randomly and rewrite that portion as a query sentence. The queries are:

1. japan important political figures
2. newyork architectural landmarks
3. bowling pin shooting
4. comedy films originated from america
5. popular science fiction books

6. fundamentals of university education
7. delicious mushroom meal
8. dance music television shows
9. monuments centuries old
10. popular soap opera in united state
11. famous football players
12. historic city and landmarks
13. biggest ships in navy
14. rivers in white mountain
15. network security methods
16. most popular novels
17. super fast computers
18. famous meals famous cooks
19. art gallery in paris
20. space research and satellites
21. information and communication technologies
22. ancient world warriors
23. cute animals in africa
24. most popular albums in south america
25. historical achievement
26. biggest cities in world

27. popular languages
28. astrology astronomy stars
29. computer science and applications
30. video game console

3.3 Context Extraction

Context extraction is one of the fundamental steps of our work. The importance of this step comes from the fact that context information of the documents which is needed to apply the context over content approach, is retrieved in this step.

The context extraction part of our system, consists of three steps. First step is analyzing the context with context analyzers. Context analyzers extract terms that represent context information from documents. Second step is pairing. After the context extraction, we pair extracted contexts for each document. Then we group pairs into six different groups according to the combinations of their sources. The last step is counting pairs in respect to their sources.

3.3.1 Context Analyzers

Context analyzers are tools that we developed to extract context information from documents. We use three different sources to extract context information. These sources are listed as follows: Apache Lucene (LCN), DBPedia Spotlight (DBPD) and Wordnet (WNDB). Analyzers are named in respect to the source that they use to extract context information.

3.3.1.1 Lucene Context Analyzer - LCA

The Lucene context analyzer is the first context analyzer that we used to extract context information. It uses components of Apache Lucene[3] search engine library to extract information.

Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java. Apache Lucene is suitable for almost any application that requires full-text search. Since it is developed by Java, it is cross-platform. Lucene is written by Doug Cutting. It joined the Apache Software Foundation's Jakarta family of high-quality open source Java products in September 2001 and became its own top-level Apache project in February 2005[3][21].

Since Lucene is a full-featured search engine library, it has wide range of functionalities on indexing and searching. Main components of Lucene can be grouped into two category as "components for indexing" and "components for querying". Components for querying are responsible for building and running queries. They also retrieve results after query run and render results to the user. On the other hand indexing components consist of acquiring content, building document, analyzing document and indexing document steps. The LCA mostly depends on the capabilities of Lucene's document analyzing component[21].

The analyzer component of Lucene is responsible for extracting and processing tokens out of text. It uses different filters to process the provided document. Filter class in Lucene library is an abstract class, but there are several implementations of it. These implementations are mostly responsible for applying a specific process on tokens and texts, such as removing chars according to the given regular expression pattern, converting tokens to lowercase or generating stems of tokens etc.

Filters can work as a chain of process, i.e. output of a filter can be redirected to another filter's input. For example, the class which is responsible for the conversion of lowercase characters can redirect its output to the class which handles

Table 3.1: Lucene Filter Examples

Filter	Input	Output
HTMLStripCharFilter	Turkey.	Turkey
PatternReplaceCharFilter	Approximately 11.6 million visitors arrived	Approximately million visitors arrived
WhitespaceTokenizer	Approximately million visitors arrived	['Approximately', 'million', 'visitors', 'arrived']
StandardFilter	Istanbul's	Istanbul
TrimFilter	' Istanbul '	'Istanbul'
WordDelimeterFilter	PowerShot	Power, Shot
LowerCaseFilter	Istanbul	istanbul
StopFilter	[and, the, only, sea, route, between, two, cities]	[sea, route, between, cities]

stemming and the stemming class can redirect its output to the class which filters some specific words. Some of the filters take Java IO Reader as an input, while others take token stream instead. In order to provide token stream to filters, Lucene analyzers also use tokenizers which split text into tokens. In the LCA, ten different filters and one tokenizer used. Some examples showing filter usages is presented in the table 3.1 .

First filter that is used in the LCA is HTMLStripCharFilter. This filter strips out html constructs from the given input. Second filter is PatternReplaceCharFilter. This filter uses regular expression to replace matching strings with thetarget string. In LCA, PatternReplaceCharFilter is used for removing numbers and punctuations. Since both of these filters process IO from Java IO Reader instead of tokens, we use them before tokenizing and the other filters.

Furthermore, tokenizing is applied to filtered text with WhitespaceTokenizer. This tokenizer splits text on whitespace and adjacent sequences of non-whitespace

characters form tokens. This tokenizer returns token stream which can be used by other filters.

In the extraction flow of LCA, next step is processing tokens with token filters. First applied token filter is StandardFilter. This filter removes dots from acronyms and 's (apostrophe followed by s) from words with apostrophes. Then TrimFilter is applied on token stream. This filter removes leading and trailing whitespace from tokens in the stream. After trimming whitespaces, WordDelimeterFilter is applied to tokens. This filter splits words into subwords and performs optional transformations on subword groups according to given rules. For example it converts 'PowerShot' to 'Power' and 'Shot' tokens. Next LowerCaseFilter is applied. This filter converts token text to lower case.[21]

In IR, stop word term represents words that are commonly used such as a, an ,the etc. since they are common, they do not have an effect on distinguishing documents. Following filter in the LCA, is StopFilter, which removes given stop words from tokens. After stop word filter is applied, RemoveDuplicatesTokenFilter removes duplicate tokens and LengthFilter deletes tokens out of given range, is applied. After application of filters, LCA returns remaining tokens as the representatives of the analyzed document.

3.3.1.2 DBPedia Spotlight Analyzer - DSCA

DBPedia Spotlight context analyzer[6] extract terms that represent context information, using DBPedia Spotlight.

DBPedia[8] is an open source project, which extract structured information from Wikipedia and it publish the results (knowledge base) on the web. DBpedia knowledge base currently describes over 2.6 million entities. For each of these entities, DBpedia defines a globally unique identifier that can be dereferenced over the Web into a rich RDF description of the entity, including human-readable definitions in 30 languages, relationships to other resources, classifications in four

concept hierarchies, various facts as well as data-level links to other Web data sources describing the entity[22][7].

DBpedia Spotlight (DBPD), is a system for automatically annotating text documents with DBPedia URIs. DBpedia Spotlight allows users to configure the annotations to their specific needs. It also provides RESTful web service interface to easily communicate with external applications[6].

DBpedia Spotlight takes full advantage of DBPedia ontology for specifying concepts. The user can restrict annotations by specifying any of all 272 classes or sets of classes of DBPedia. DBPedia supports SPARQL queries on its knowledge base. Since DBPedia Spotlight uses DBPedia, SPARQL queries can be used to filter annotation results. Moreover DBPedia Spotlight can calculate different scores for task-specific requirements. These scores that DBPedia Spotlight calculates, are "prominence" which gives how many times a resource is mentioned in Wikipedia, "topical relevance" which shows how close a text is to DBPedia resource's and "contextual ambiguity" which represents the possible ambiguity when a candidate resource has a high relevance with another. Confidence and support can also be specified for queries[6].

In the DBPedia Spotlight context analyzer (DSCA), we use RESTful web service inteface of DBPD. The web service interface returns list of annotations and its properties according to support and confidence parameters. Each annotation has various properties to identify annotated string. An example of a query is shown as follows.

Figure 3.2: DBpedia Spotlight Web Service Query

```
1  curl -H "Accept: application/json" \
2    http://spotlight.dbpedia.org/rest/annotate \
3    --data-urlencode "text=Istanbul is the largest city in Turkey
     ,
4    constituting the country's economic, cultural, and historical
5    heart." \
6    --data "confidence=0.2" \
7    --data "support=20"
8
```

The result of the query presented in figure 3.2 is given.

Figure 3.3: DBPedia Spotlight Web Service Result (JSON)


```
{
  "@text": "Istanbul is the largest city in Turkey, constituting
  the country's economic, cultural, and historical heart.",
  "@confidence": "0.2",
  "@support": "20",
  "@types": "",
  "@sparql": "",
  "@policy": "whitelist",
  "Resources":   [
        {
      "@URI": "http://dbpedia.org/resource/Istanbul",
      "@support": "6635",
      "@types": "DBpedia:City,DBpedia:Settlement,
      DBpedia:PopulatedPlace,DBpedia:Place, Schema:Place,
      Schema:City,Freebase:/olympics/olympic_bidding_city,
      Freebase:/olympics,Freebase:/travel/travel_destination,
      Freebase:/travel,Freebase:/business/employer,
      Freebase:/business,Freebase:/protected_sites/listed_site,
      Freebase:/sports/sports_team_location,Freebase:/sports,
      Freebase:/protected_sites,Freebase:/film/film_location,
      Freebase:/film,Freebase:/location/dated_location,
      Freebase:/location,Freebase:/book/book_subject,
      Freebase:/book,Freebase:/business/business_location",
      "@surfaceForm": "Istanbul",
      "@offset": "0",
      "@similarityScore": "0.1449064463376999",
      "@percentageOfSecondRank": "-1.0"
    }, ... ] }
```


"SurfaceForm" and "Types" properties are used as context information in the DSCA. First parameter, SurfaceForm, represents the string that is annotated by DBPedia Spotlight. Second parameter, the types, shows related categories and sub-categories of annotated string. These categories and sub-categories, represent the context information that we need to extract. Types contains a list of related categories and sub-categories which are represented in hierarchical order and each item in the list also contains name of the data source. The items in the types property list, has a form of [data source]:/[category]/[sub-category]/[...]. For example, the result retrieved for Istanbul is "Freebase:/location/citytown". DSCA removes source information and splits rest of the string, which contains

categories and sub-categories, on "/" character and returns extracted tokens and SurfaceForm as context information.

3.3.1.3 Wordnet Analyzer - WNCA

WordNet context analyzer is another context analyzer of our system. It extracts terms that represent context information using WordNet[9].

Wordnet is a lexical and semantic database in English. It has been developed by the Cognitive Science Laboratory in Princeton University. Wordnet provides conceptual-semantic and lexical relations between nouns, verbs, adjectives, and adverbs. Traditional dictionaries are organized in alphabetic order to be readable and searchable by human-beings. However, Wordnet is a dictionary where words are linked together according to their semantic relationships. In this point of view, Wordnet is more like thesaurus than a dictionary[23][9].

The most important relation in WordNet is "synonym". Synonyms are words that have similar meanings. Furthermore, set of synonyms are represented with the term "synset" in WordNet. Synset is an abstract concept which states all relationships that the word has. Wordnet divides lexicons into five categories as nouns, verbs, adjectives, adverbs and function words. In our work only nouns and verbs are used for context extraction. Moreover, there are 27 relationship types in WordNet. These are "also see", "antonym", "attribute", "cause", "derived", "derived from adjective", "entailment", "holonym member", "holonym part", "holonym substance", "hypernym", "hypernym instance", "hyponym", "hyponym instance", "meronym member", "meronym part", "meronym substance", "participle", "pertainym", "region", "region member", "similar to", "topic", "topic member", "usage", "usage member", "verb group". Since our computational power is limited, we only use "Topic" relation along with synonyms in our work[23][9].

In WNCA, MIT Java Wordnet Interface (JWI)[24][25] is used. JWI is a Java library for interfacing with Wordnet which supports WordNet versions 1.6 through 3.0 . According to Finlayson's comparison, JWI is the highest-performance, widest-coverage, easiest-to-use library available[25].

WNCA analyzes documents in two sequential steps. First step is the pre-analyze step. In this step, WNCA uses LCA to process documents. Since the data that will be analyzed with WordNet should consist of words, WNCA tokenizes and filters documents using LCA and passes extracted tokens (words) to the next step.

Second step in WNCA is the analyze step. In this step, first, WNCA gets the stems of words that are retrieved from the previous step using JWI WordNet stemmer. Stemming is an essential step because any word that Wordnet does not contain, can not be used for extracting relations. Next, WNCA extracts senses of stems that are returned from stemmer. As a next step, WNCA retrieves all synsets of the extracted senses. After synset extraction, WNCA extracts synonyms, semantically related words and lexically related words. Synonyms, semantically related words and lexically related words are representatives of context information.

3.3.2 Pairing & Pair Counting

In the previous steps of context extraction, we extracted context information using the context analyzers. Using the correlation between contexts and documents is the base idea of our work. In order to measure the degree of correlation between the extracted context information and the documents, we pair them and count these pairs. Our assumption behind pairing and pair counting is that, if two terms that represent a context occur in the same document, both terms are related. Moreover, we also assume that the strength of he relationship increases each time they occur together.

After context extraction step is completed, we pair the terms that represent context information together and group pairs with respect to their source combinations. The pair groups that represent the combination of context sources are:

- LCN-LCN: Pairs are from both Lucene.
- LCN-DBPD: One of the pair is from Lucene the other one is from DBPedia Spotlight.
- LCN-WNDB: One of the pair is from Lucene the other one is from Wordnet.
- DBPD-DBPD: Pairs are from both DBPedia Spotlight.
- WNDB-WNDB: Pairs are from both Wordnet.
- DBPD-WNDB: One of the pair is from DBPedia Spotlight the other one is from Wordnet.

In order to find correlation strength of each pair, we count all extracted pairs. Since there is a big amount of data (millions of pairs), Map reduce paradigm on Apache Hadoop[26] is used for counting.

MapReduce is the new programming paradigm in parallel and distributed computing areas. In 2004, Google published a paper which is about data processing on large clusters and they proposed MapReduce framework for Google distributed file system[27]. After that, D. Cutting et al. [28] had started developing an open source MapReduce framework called Hadoop. In addition to this, Hadoop also has its own distributed file system, HDFS. The main purpose of Hadoop project is performing distributed algorithms on commodity hardware. Hadoop has high fault tolerance and allows data redundancy in its nodes. Hadoop uses functional programming model to implement a mapper and a reducer[29].

We use Amazon Elastic MapReduce service as our Hadoop cluster. Amazon Elastic MapReduce (EMR)[30] is a web service is provided by Amazon. EMR uses Hadoop MapReduce framework works effectively on large scale data sets

and supports many programming languages such as Ruby, Perl, Python, and C++ .

EMR service provides a library to develop MapReduce applications on Amazon EMR clusters. This library is called MRJob[31] library. MRJob is used in Python and accesses to Hadoop jar file. It provides a high level Hadoop API as a Python wrapper.

Table 3.2: Merged Pair Count Sample

PAIR	LCN LCN	DBPD DBPD	WNDB WNDB	LCN DBPD	LCN WNDB	DBPD WNDB
istanbul|city	5	0	0	10	5	0
istanbul|historic	5	0	0	8	3	0
city|metropolitan	8	10	5	3	3	15
city|travel	25	6	8	10	12	20
historic|architecture	20	10	13	10	15	25
historic|movie	10	3	2	12	9	5

We write a mapper as a Python script to count word pairs in the input file. Normally, in Hadoop, we need to write a reducer which will be run after mapper codes. In Elastic MapReduce framework, the aggregate parameter is passed to reducer option. This Python script is interpreted as a Hadoop application. Then we run scripts over EMR.

The counting scripts produce results which contain pairs, sources and counts. After counting is completed, we merge these counts into a matrix. Each row of matrix contains term pair and counts with respect to the source combinations. An sample matrix of merged pair counts is shown in Table 3.2.

3.4 Context Clustering

Clustering can be defined as unsupervised grouping of patterns into clusters. In general, clustering algorithms group similar instances according to distance metrics such as Euclidean Distance, Manhattan Distance, term frequencies etc. In our work, we use the count of term pairs, similar to term frequency, as the distance metric, which also represents the relation between words[32].

Graph clustering is another method of clustering. Graphs are data structures which consist of vertices (nodes) and edges. Edges connect vertices and they may have properties such as weights. Graph clustering is a clustering method which groups similar vertices using edges and their weights as distance metric. [33]

On the other hand context clustering is a more abstract concept that represents grouping similar documents by using context information as distance metric. Using term frequencies in documents is a common way to cluster context. However, since the relationships between terms more like a network that can be represent as graph, in our work, we represent relation of words in graph format and apply graph clustering to cluster contexts.

In order to cluster contexts, first, we build a graph of terms. Terms are connected via distances generated from pair counts that are previously calculated. In order to calculate distances of terms, first we normalize pair counts using max-min normalization. Then we merge counts from different source combinations (LCN-LCN, LCN-DBPD, DBPD-WNDB etc) by assigning weights for each source combination. Then we sum all weighted scores from source combinations.

Second step is clustering graph. In order to cluster our graph of terms, we use Markov Cluster Algorithm (MCL Algorithm)[34]. The MCL algorithm is a fast and scalable unsupervised cluster algorithm for graphs. It is based on an estimated simulation of a graph. Dongen explained the paradigm of MCL alogrithm in his PHD thesis as: "A random walk in G that visits a dense cluster will likely not leave the cluster until many of its vertices have been visited."[35]. The MCL

algorithm does random walks on graph and calculates probabilities of going to other nodes using "Markov Chains". Since the Markov process does not show cluster structure in its underlying graph, MCL algorithm adds another operator called "inflation" to calculation process. The inflation process uses entry-wised Hadamard-Schur product combined with diagonal scaling. It is responsible for both strengthening and weakening of flow which allows flow to connect other regions of graph[35].

Finally, we export cluster context-term mapping which shows clusters that contain context terms. In the next section labeling documents with clusters and scoring are explained.

3.5 Labeling & Scoring

In this part of our work, we label documents with clusters, in order to find which document belongs to which cluster. For labeling, we use context clusters which are generated previously and the context information which is extracted for each document to assign clusters to documents. We also label queries with clusters as well. Then we use co-existing clusters between documents and queries in order to score results. In order to label documents and score results we use Apache solr[36] as standalone full-text search engine.

Apache Solr[36] is the popular, fast, scalable, opensource search engine that built on Apache Lucene Project. Solr is standalone full-featured search engine that provide all capabilities that we need to index and retrieve documents. Since Solr is a wrapper of Lucene library, it provides all capabilities of Lucene. We use Solr in order to index document (labeled or not labeled) and retrieve them.[36]

First, we index documents using SOLR and retrieve results for the queries which we have listed in corpus section. There are 30 queries and we reorder result set of each query.

Second, we label documents. In order to label documents we need context information of each document. In the context extraction step we have extracted contexts for each document so we have context terms that represents contexts of each document. We also have cluster context-term mapping which has generated in clustering step. We label documents using context clusters and context-terms. Each document has several clusters and cluster-counts for each cluster.

Third, we label queries. In order to label, we need context information for each query. We also have extracted contexts of queries using context analyzers. Then we label queries with using context information that represents queries and cluster context-term mapping.

Next, we calculate new scores of documents. In order to calculate new scores we use dot product of cluster-counts of documents and queries. Then we sort results for each query according to new calculated scores. Since we have seven different clusters for seven different context source combination, in scoring step we generate seven different lists for each query.

Chapter 4

Experiments

4.1 Experimental Setup

The experimental setup consist of two main steps. First step is the implementation of the system itself. Second step is collecting data to compare results of our system.

4.1.1 System Implementation

The system that we have implemented to apply context over content approach consists of five different steps. These are data sources, context extraction, context clustering, labeling, and scoring.

First step is data sources. The data source part consists of two components corpus and queries. Corpus is the collection of documents that we analyzed in the next steps. As a corpus we have selected Westbury Lab Wikipedia Corpus (WLC). WLC is a snapshot of Wikipeda (April 2010). WLC contains over 2 million articles. We select 8000 document randomly in the preprocessing step as it is mentioned previously. We select 30 queries for the testing step. In order to select queries, first we select 30 different documents randomly. Then we select a small portion of each selected document. Since selected portions may not form sentences that can be used as queries, we rewrite them as queries.

Second part of the system is context extraction. In the implementation of context extraction step, we used Java language. Context extraction application is a multi-thread application that is designed to process multiple documents at the same time. The computer that we used to run context extraction application has eight core Intel i7 2.3 mhz processor and 32 GB ram. Context extraction application is also responsible for pairing terms. Total completion time of extraction and pairing process is 4 days. Next, we count extracted pairs using Amazon elastic map reduce (EMR) service. We use 100 different computers for counting on EMR. Each computer has 7.5 GB ram and 4 EC2 Compute Units. After counting is completed, we merge and normalize counts using Python scripts. In order to merge scores from different context sources, we use weights that shown in table 4.1.

Table 4.1: Weight Sets for Merging

	LCN LCN	LCN DBPD	LCN WNDB	DBPD DBPD	DBPD WNDB	WNDB WNDB
1	0.167	0.167	0.167	0.167	0.166	0.166
2	1	0	0	0	0	0
3	0	1	0	0	0	0
4	0	0	1	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	1	0
7	0	0	0	0	0	1

Third part of the system is context clustering. We use MCL implementation distributed by Linux Mint. We use a computer which has eight core Intel i7 2.3 mhz processor and 32 GB ram for clustering. We cluster context terms according to merged scores which merged by using weights in table 4.1. After clustering is completed we had 7 different clusters.

Fourth part of the system is context labeling. We use Python scripts to map context information of documents with the context clusters. Then we score using context-labels and context-clusters then we sort results using new calculated scores for each cluster set.

4.1.2 Data Collection & Comparing

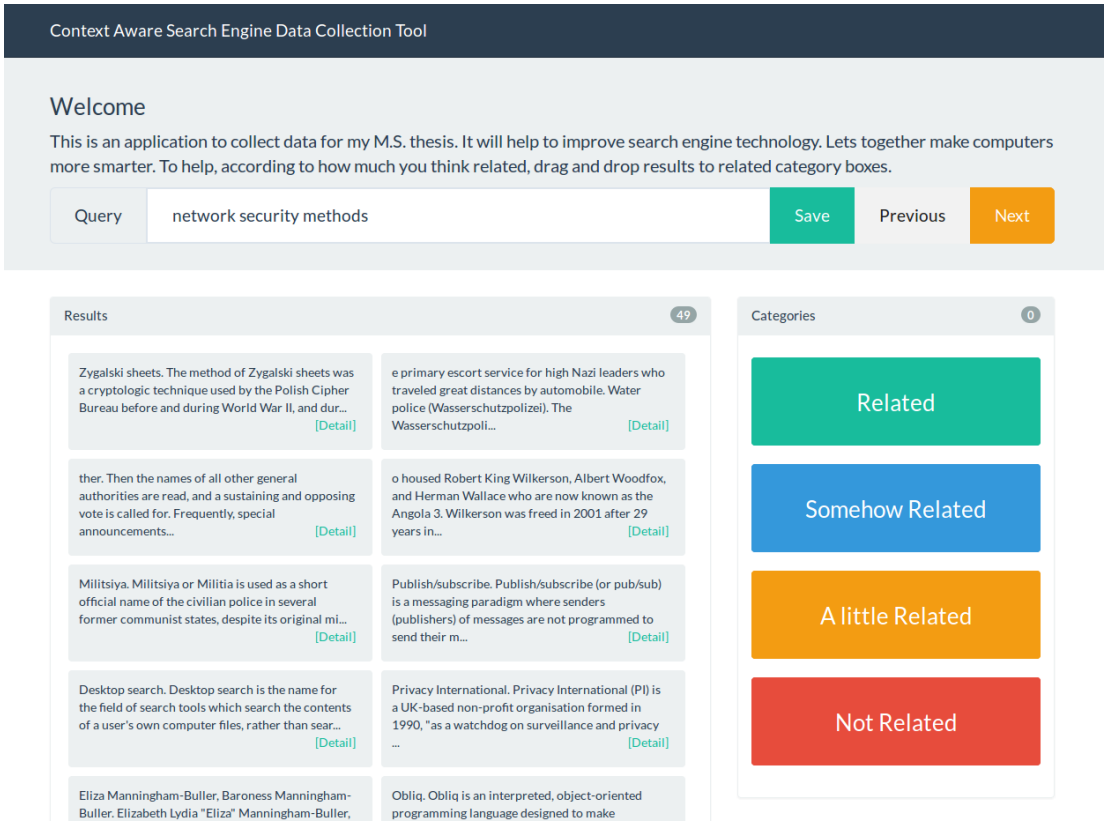

Figure 4.1: Test Data Collection Tool

In order to measure performance, we need reference data to compare re-scored results. Therefore, we have designed and implemented a web application for data collection. The application retrieves top 50 results from SOLR without any extra scoring and shows these results to the user without ordering. The user can categorize the results into four categories. These categories are “not related”, “a little related”, “related” and “very related”. Then the application stores the user selections as scores. Each category has score between zero and three according to its degree of relatedness. A screen-shot of the application shown in figure 4.1.

The categorized results of 30 different queries are obtained by our application and are given to several different users. Each query has a set of results for each user. Then we calculate means of scores that are given to each result of query. Then we reorder results according to new scores.

Following, in order to measure performance, we calculate precision, recall, and f-measure. First we calculate precision, recall, and f-measure for default result sets of SOLR. Second, since we have seven clusters for each merged scores (merged with parameters in Table 4.1), we calculate performance for each cluster. Third we compare results. Results of comparison are given in the next section.

4.2 Experiment Results

In order to measure performance of our study, we use metrics that frequently used in the literature. These metrics are precision, recall, and f-measure. We also use Spearman's rank correlation coefficient to compare rankings.

Since we have 7 different group of weights (Table 4.1)) we have 7 different clusters for each group of weights. Characteristics of each group is shown as follows:

- First group includes all source combinations into calculation equally.
- Second group focuses on LCN
- Third group bring forwards LCN-DBPD combination.
- Fourth group reveals characteristics of LCN-WNDB combination.
- Fifth group focuses on DBPD
- Sixth group bring forwards DBPD-WNDB combination
- Seventh group focuses on WNDB

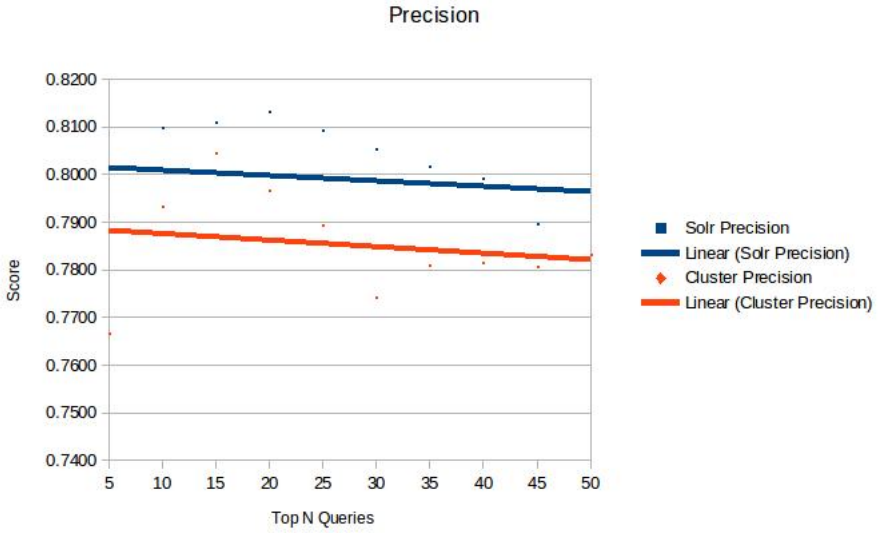

Figure 4.2: Precision

Comparison of these groups are shown in Tables 4.2, 4.3, 4.4. According to results first and second groups are slightly better than the others but in general all clusters has similar affect on calculations.

Next we compare our results with results of Solr. Comparison for top 20 results of Solr and first cluster, is shown in Table 4.5. According to results, precision of Solr is slightly better. On the other hand recall of cluster is a little better than Solr. In general performance for both Solr and Clusters are almost same. Precision (Figure 4.2), recall (Figure 4.3), and f-measure (Figure 4.4) charts are also indicate similarities between performances. Spearman's rank correlation coefficient is also used to compare rankings of the queries. Comparing with Spearman also indicate that there are not any significant improvement on rankings (17 of 30 queries are better). Results are shown in Table 4.6

The corpus that we use, contains 8000 documents. This size of corpus is small for a search engine. Using small corpus might be the reason that clusters have not formed good enough to distinguish concepts. Since concepts are not distinguished well enough, rescoring with clusters are not changed results significantly.

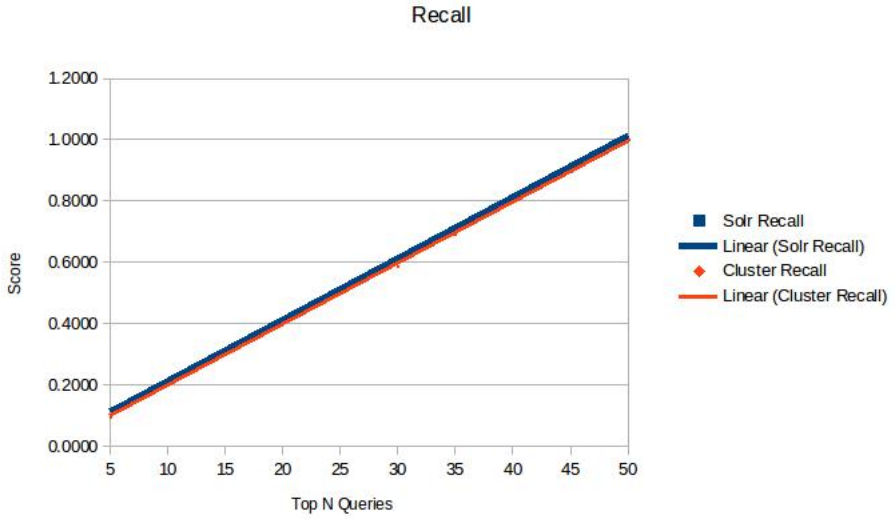

Figure 4.3: Recall

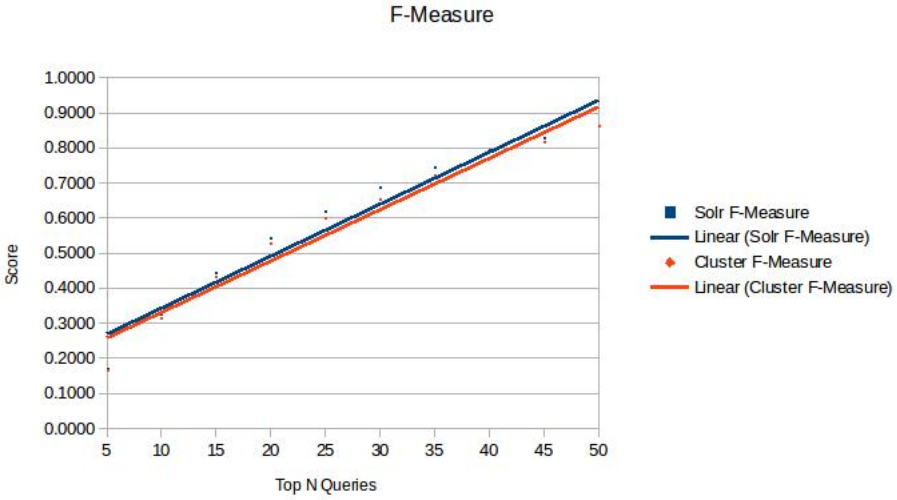

Figure 4.4: F-Measure

Table 4.2: Cluster Precision Table for Top 20 Result

Query ID	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7
1	0.70	0.70	0.55	0.75	0.55	0.60	0.70
2	0.85	0.90	0.80	0.85	0.80	0.85	0.85
3	0.85	0.90	0.85	0.85	0.85	0.85	0.85
4	0.95	0.95	1.00	0.95	1.00	0.95	0.95
5	0.95	0.95	0.95	1.00	0.95	0.95	0.95
6	0.85	0.85	0.95	0.85	0.95	0.85	0.85
7	0.55	0.55	0.35	0.55	0.35	0.40	0.50
8	1.00	0.95	1.00	0.95	0.95	1.00	1.00
9	1.00	1.00	1.00	1.00	1.00	1.00	1.00
10	0.80	0.75	0.75	0.75	0.70	0.80	0.75
11	0.95	1.00	0.95	0.95	0.95	0.95	0.95
12	1.00	1.00	1.00	1.00	1.00	1.00	1.00
13	0.90	0.90	0.90	0.90	0.90	0.90	0.90
14	0.95	0.95	1.00	1.00	1.00	1.00	0.95
15	0.85	0.80	0.80	0.80	0.80	0.80	0.85
16	0.95	0.95	0.95	0.95	0.95	0.95	0.95
17	0.65	0.70	0.80	0.60	0.85	0.85	0.65
18	0.25	0.30	0.30	0.25	0.35	0.30	0.30
19	0.90	0.90	0.85	0.90	0.85	0.90	0.90
20	0.90	0.90	0.95	0.90	0.95	0.90	0.90
21	0.85	0.85	0.85	0.80	0.85	0.80	0.85
22	0.75	0.80	0.85	0.80	0.80	0.85	0.75
23	0.55	0.55	0.55	0.55	0.55	0.55	0.55
24	0.75	0.75	0.70	0.75	0.70	0.70	0.75
25	1.00	1.00	0.95	0.95	0.90	0.95	1.00
26	0.35	0.35	0.35	0.35	0.40	0.30	0.35
27	0.25	0.25	0.35	0.25	0.35	0.30	0.35
28	0.75	0.75	0.70	0.75	0.65	0.60	0.75
29	0.90	0.90	0.80	0.90	0.80	0.80	0.90
30	0.95	0.95	0.95	0.95	0.95	0.95	0.90
Avg	0.80	0.80	0.79	0.79	0.79	0.79	0.80

Table 4.3: Cluster Recall Table for Top 20 Result

Query ID	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7
1	0.45	0.45	0.36	0.48	0.36	0.39	0.45
2	0.43	0.45	0.40	0.43	0.40	0.43	0.43
3	0.50	0.53	0.50	0.50	0.50	0.50	0.50
4	0.39	0.39	0.41	0.39	0.41	0.39	0.39
5	0.40	0.40	0.40	0.42	0.40	0.40	0.40
6	0.40	0.40	0.44	0.40	0.44	0.40	0.40
7	0.55	0.55	0.35	0.55	0.35	0.40	0.50
8	0.42	0.40	0.42	0.40	0.40	0.42	0.42
9	0.41	0.41	0.41	0.41	0.41	0.41	0.41
10	0.43	0.41	0.41	0.41	0.38	0.43	0.41
11	0.40	0.43	0.40	0.40	0.40	0.40	0.40
12	0.41	0.41	0.41	0.41	0.41	0.41	0.41
13	0.38	0.38	0.38	0.38	0.38	0.38	0.38
14	0.40	0.40	0.43	0.43	0.43	0.43	0.40
15	0.37	0.35	0.35	0.35	0.35	0.35	0.37
16	0.41	0.41	0.41	0.41	0.41	0.41	0.41
17	0.38	0.41	0.47	0.35	0.50	0.50	0.38
18	0.33	0.40	0.40	0.33	0.47	0.40	0.40
19	0.43	0.43	0.41	0.43	0.41	0.43	0.43
20	0.39	0.39	0.41	0.39	0.41	0.39	0.39
21	0.39	0.39	0.39	0.36	0.39	0.36	0.39
22	0.38	0.40	0.43	0.40	0.40	0.43	0.38
23	0.37	0.37	0.37	0.37	0.37	0.37	0.37
24	0.46	0.46	0.42	0.46	0.42	0.42	0.46
25	0.44	0.44	0.41	0.41	0.39	0.41	0.44
26	0.37	0.37	0.37	0.37	0.42	0.32	0.37
27	0.25	0.25	0.35	0.25	0.35	0.30	0.35
28	0.41	0.41	0.38	0.41	0.35	0.32	0.41
29	0.43	0.43	0.38	0.43	0.38	0.38	0.43
30	0.41	0.41	0.41	0.41	0.41	0.41	0.39
Avg	0.41	0.41	0.40	0.40	0.40	0.40	0.41

Table 4.4: Cluster F-Measure Table for Top 20 Result

Query ID	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7
1	0.55	0.55	0.43	0.59	0.43	0.47	0.55
2	0.57	0.60	0.53	0.57	0.53	0.57	0.57
3	0.63	0.67	0.63	0.63	0.63	0.63	0.63
4	0.55	0.55	0.58	0.55	0.58	0.55	0.55
5	0.56	0.56	0.56	0.59	0.56	0.56	0.56
6	0.54	0.54	0.60	0.54	0.60	0.54	0.54
7	0.55	0.55	0.35	0.55	0.35	0.40	0.50
8	0.59	0.56	0.59	0.56	0.56	0.59	0.59
9	0.58	0.58	0.58	0.58	0.58	0.58	0.58
10	0.56	0.53	0.53	0.53	0.49	0.56	0.53
11	0.57	0.60	0.57	0.57	0.57	0.57	0.57
12	0.58	0.58	0.58	0.58	0.58	0.58	0.58
13	0.54	0.54	0.54	0.54	0.54	0.54	0.54
14	0.57	0.57	0.60	0.60	0.60	0.60	0.57
15	0.52	0.49	0.49	0.49	0.49	0.49	0.52
16	0.58	0.58	0.58	0.58	0.58	0.58	0.58
17	0.48	0.52	0.59	0.44	0.63	0.63	0.48
18	0.29	0.34	0.34	0.29	0.40	0.34	0.34
19	0.58	0.58	0.55	0.58	0.55	0.58	0.58
20	0.55	0.55	0.58	0.55	0.58	0.55	0.55
21	0.53	0.53	0.53	0.50	0.53	0.50	0.53
22	0.50	0.53	0.57	0.53	0.53	0.57	0.50
23	0.44	0.44	0.44	0.44	0.44	0.44	0.44
24	0.57	0.57	0.53	0.57	0.53	0.53	0.57
25	0.61	0.61	0.58	0.58	0.55	0.58	0.61
26	0.36	0.36	0.36	0.36	0.41	0.31	0.36
27	0.25	0.25	0.35	0.25	0.35	0.30	0.35
28	0.53	0.53	0.49	0.53	0.46	0.42	0.53
29	0.58	0.58	0.52	0.58	0.52	0.52	0.58
30	0.58	0.58	0.58	0.58	0.58	0.58	0.55
Avg	0.53	0.53	0.52	0.53	0.52	0.52	0.53

Table 4.5: Solr-Cluster Comparision for Top 20 Result

Query ID	Precision Solr	Recall Solr	F-Measure Solr	Precision Cluster	Recall Cluster	F-Measure Cluster
1	0.650	0.419	0.510	0.700	0.452	0.549
2	0.800	0.400	0.533	0.850	0.425	0.567
3	0.800	0.471	0.593	0.850	0.500	0.630
4	1.000	0.408	0.580	0.950	0.388	0.551
5	1.000	0.417	0.588	0.950	0.396	0.559
6	0.950	0.442	0.603	0.850	0.395	0.540
7	0.400	0.400	0.400	0.550	0.550	0.550
8	1.000	0.417	0.588	1.000	0.417	0.588
9	1.000	0.408	0.580	1.000	0.408	0.580
10	0.950	0.514	0.667	0.800	0.432	0.561
11	0.900	0.383	0.537	0.950	0.404	0.567
12	0.950	0.388	0.551	1.000	0.408	0.580
13	1.000	0.426	0.597	0.900	0.383	0.537
14	1.000	0.426	0.597	0.950	0.404	0.567
15	0.900	0.391	0.545	0.850	0.370	0.515
16	0.900	0.391	0.545	0.950	0.413	0.576
17	0.650	0.382	0.481	0.650	0.382	0.481
18	0.500	0.667	0.571	0.250	0.333	0.286
19	0.850	0.405	0.548	0.900	0.429	0.581
20	1.000	0.435	0.606	0.900	0.391	0.545
21	1.000	0.455	0.625	0.850	0.386	0.531
22	0.850	0.425	0.567	0.750	0.375	0.500
23	0.450	0.300	0.360	0.550	0.367	0.440
24	0.600	0.364	0.453	0.750	0.455	0.566
25	0.900	0.391	0.545	1.000	0.435	0.606
26	0.500	0.526	0.513	0.350	0.368	0.359
27	0.400	0.400	0.400	0.250	0.250	0.250
28	0.850	0.459	0.596	0.750	0.405	0.526
29	0.750	0.357	0.484	0.900	0.429	0.581
30	0.900	0.391	0.545	0.950	0.413	0.576
Avg	0.813	0.422	0.544	0.797	0.405	0.528

Table 4.6: Comparision with Spearman's rank Correlation Coefficient

Query Id	Solr Spearman Score	Cluster Spearman Score
1	0.184	-0.220
2	-0.093	0.171
3	-0.181	-0.041
4	0.123	-0.149
5	0.161	-0.015
6	0.253	-0.006
7	-0.240	-0.018
8	0.140	0.044
9	-0.107	0.078
10	0.089	-0.347
11	-0.004	0.187
12	-0.044	0.078
13	-0.063	-0.203
14	0.037	-0.012
15	0.095	-0.005
16	0.219	-0.165
17	-0.113	-0.084
18	-0.026	0.035
19	0.001	0.034
20	-0.139	0.206
21	-0.160	-0.060
22	-0.163	0.111
23	0.218	-0.117
24	0.011	0.147
25	0.172	0.095
26	-0.045	0.311
27	-0.308	-0.029
28	0.100	-0.070
29	-0.214	0.184
30	-0.092	-0.007

Chapter 5

Conclusion

The main problem of search engines is retrieving most relevant documents to the user from large amount of data according to the given query by the user. Both, the size of the data that is needed to be indexed and finding relevant information according to the users need make the problem challenging.

In this thesis, we focus on retrieving relevant data instead of processing big amounts of data. Llian's context over content approach of human brain has inspired us to develop a method which will improve search engine performance using a similar way to human thinking process. In our work we propose a method which uses the context information of documents to improve the search engine performance. The steps we have applied as follows:

- A corpus is prepared out of a small portion of Westbury Lab Corpus.
- Queries are prepared using documents in corpus
- The Context information of documents in corpus is extracted using context analyzers (Lucene Analyzer, DBPedia Spotlight Analyzer, Wordnet Analyzer).
- For each document, extracted terms that represent context information are paired according to their sources

- Term pairs are counted, merged and normalized to represent how close the relationship is between terms.
- Graph structure is formed using terms and weights (scores calculated in the previous step).
- Graph is clustered using Markov Cluster Algorithm. Cluster context-term mapping is generated
- Documents and queries are labeled with clusters using their context information and clusters context-term mappings
- Results of queries are re-scored using labels of documents and queries.
- Test data is collected using web application that is specifically designed for this process.
- Both raw results and re-scored results are compared using collected test data.

As a result, we could not improve search engine performance significantly. The reason, mostly, is dependent to our corpus size. Because lack of computational power, we could use 8000 documents which is very small set for a search engine. Using limited amount of documents is prevented forming clusters well-enough. Rescoring with using non well-formed clusters did not make any significant change.

Our content over context approach, that we presented, forms a basis for our future work. In the future, we plan to implement our design on a distributed system such as Hadoop so that we can process much bigger corpus. We also plan to extend context sources that we used to extract context information.

References

[1] World internet users and population stats. URL http://www.internetworldstats.com/stats.htm.

[2] Rodolfo R. Llinas. *i of the vortex From Neurons to Self*. MIT Press, 2002.

[3] Apache lucene. URL http://lucene.apache.org/core/.

[4] Apache foundation. URL http://www.apache.org/.

[5] Dbpedia spotlight, . URL https://github.com/dbpedia-spotlight/dbpedia-spotlight/wiki.

[6] Pablo N. Mendes, Max Jakob, Andres Garcia-Silva, and Christian Bizer. Dbpedia spotlight: Shedding light on the web of documents. In *Proceedings of the 7th International Conference on Semantic Systems (I-Semantics)*, 2011.

[7] Dbpedia, . URL http://wiki.dbpedia.org/About.

[8] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 2014.

[9] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, November 1995. ISSN 0001-0782. doi: 10.1145/219717.219748. URL http://doi.acm.org/10.1145/219717.219748.

[10] P. J. Brown and G. J. F. Jones. Context-aware retrieval: Exploring a new environment for information retrieval and information filtering. *Personal Ubiquitous Comput.*, 5(4):253–263, January 2001. ISSN 1617-4909. doi: 10.1007/s007790170004. URL http://dx.doi.org/10.1007/s007790170004.

[11] Ph. Mylonas, D. Vallet, P. Castells, M. FernÁndez, and Y. Avrithis. Personalized information retrieval based on context and ontological knowledge. *Knowl. Eng. Rev.*, 23(1):73–100, March 2008. ISSN 0269-8889. doi: 10.1017/S0269888907001282. URL http://dx.doi.org/10.1017/S0269888907001282.

[12] Lin Li, Luo Zhong, Guandong Xu, and Masaru Kitsuregawa. A feature-free search query classification approach using semantic distance. *Expert Syst. Appl.*, 39(12):10739–10748, September 2012. ISSN 0957-4174. doi: 10.1016/j.eswa.2012.02.191. URL http://dx.doi.org/10.1016/j.eswa.2012.02.191.

[13] Ofer Egozi, Shaul Markovitch, and Evgeniy Gabrilovich. Concept-based information retrieval using explicit semantic analysis. *ACM Trans. Inf. Syst.*, 29(2):8:1–8:34, April 2011. ISSN 1046-8188. doi: 10.1145/1961209.1961211. URL http://doi.acm.org/10.1145/1961209.1961211.

[14] Atanas Kiryakov, Borislav Popov, Ivan Terziev, Dimitar Manov, and Damyan Ognyanoff. Semantic annotation, indexing, and retrieval. *Web Semant.*, 2(1):49–79, December 2004. ISSN 1570-8268. doi: 10.1016/j.websem.2004.07.005. URL http://dx.doi.org/10.1016/j.websem.2004.07.005.

[15] Debajyoti Mukhopadhyay and Sukanta Sinha. A new approach to design graph based search engine for multiple domains using different ontologies. In *Proceedings of the 2008 International Conference on Information Technology*, ICIT '08, pages 267–272, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3513-5. doi: 10.1109/ICIT.2008.46. URL http://dx.doi.org/10.1109/ICIT.2008.46.

[16] Santosh Kumar Ray, Shailendra Singh, and B. P. Joshi. A semantic approach for question classification using wordnet and wikipedia. *Pattern Recogn. Lett.*, 31(13):1935–1943, October 2010. ISSN 0167-8655. doi: 10.1016/j.patrec.2010.06.012. URL http://dx.doi.org/10.1016/j.patrec.2010.06.012.

[17] Celina Santamaría, Julio Gonzalo, and Javier Artiles. Wikipedia as sense inventory to improve diversity in web search results. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1357–1366, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1858681.1858819.

[18] Michael Schuhmacher and Simone Paolo Ponzetto. Exploiting dbpedia for web search results clustering. In *Proceedings of the 2013 Workshop on Automated Knowledge Base Construction*, AKBC '13, pages 91–96, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2411-3. doi: 10.1145/2509558.2509574. URL http://doi.acm.org/10.1145/2509558.2509574.

[19] Hinrich Schütze Christopher D. Manning, Prabhakar Raghavan. *Introduction to Information Retrieval*. Cambridge University Press, 2009.

[20] Westbury C. Shaoul C. (2010) the westbury lab wikipedia corpus, edmonton, ab: University of alberta. URL http://www.psych.ualberta.ca/~westburylab/downloads/westburylab.wikicorp.download.html.

[21] Otis Gospodnetic Michael McCandless, Erik Hatcher. *Lucene In Action*. Manning Publications, 2010.

[22] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. {DBpedia} - a crystallization point for the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154 – 165, 2009. ISSN 1570-8268. doi: http://dx.doi.org/10.1016/j.websem.2009.

07.002. URL http://www.sciencedirect.com/science/article/pii/S1570826809000225. The Web of Data.

[23] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller. Introduction to WordNet: an on-line lexical database. *International Journal of Lexicography*, 3(4):235–244, 1990. URL http://wordnetcode.princeton.edu/5papers.pdf.

[24] Jwi 2.3.3. URL http://projects.csail.mit.edu/jwi/.

[25] Mark Alan. Java libraries for accessing the princeton wordnet: Comparison and evaluation. In *Proceedings of the 7th Global Wordnet Conference. Tartu, Estonia.*, 2014.

[26] Hadoop. URL http://hadoop.apache.org/.

[27] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008. ISSN 0001-0782. doi: 10.1145/1327452.1327492. URL http://doi.acm.org/10.1145/1327452.1327492.

[28] Tom White. *Hadoop: The Definitive Guide, 3rd Edition*. O'Reilly Media / Yahoo Press, 2012.

[29] Alex Holmes. *Hadoop in Practice*. Manning Publications, 2012.

[30] Amazon elastic mapreduce. URL http://aws.amazon.com/elasticmapreduce/.

[31] Amazon elastic mapreduce. URL https://pythonhosted.org/mrjob/.

[32] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, September 1999. ISSN 0360-0300. doi: 10.1145/331499.331504. URL http://doi.acm.org/10.1145/331499.331504.

[33] Satu Elisa Schaeffer. Survey: Graph clustering. *Comput. Sci. Rev.*, 1(1):27–64, August 2007. ISSN 1574-0137. doi: 10.1016/j.cosrev.2007.05.001. URL http://dx.doi.org/10.1016/j.cosrev.2007.05.001.

[34] Mcl - a cluster algorithm for graphs. URL http://micans.org/mcl/.

[35] Stijn van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, 2000.

[36] Apache solr. URL https://lucene.apache.org/solr/.

Curriculum Vitae