Darko Stanisavljevic, BSc

Semantic stability of Wikipedia

MASTER'S THESIS

to achieve the university degree of
Master of Science
Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Assoc.Prof. Dipl.-Ing. Dr.techn., Denis Helic

Knowledge Technologies Institute

Graz, April 2016

Darko Stanisavljevic, BSc

Semantische Stabilität der Wikipedia

MASTERARBEIT

zur Erlangung des akademischen Grades
Master of Science
Masterstudium: Informatik

eingereicht an der

Techischen Universität Graz

Betreuer

Assoc.Prof. Dipl.-Ing. Dr.techn., Denis Helic

Institut für Wissenstechnologien

Graz, April 2016

Abstract

This master's thesis deals with the assessment of the semantic stability of Wikipedia. Semantic stability is a question of how quickly Wikipedia articles change in time. In a stable system, only the slow changes in articles are noticeable, whereas in unstable systems the content of the article changes more frequently. Thus, semantic stability answers the question if Wikipedia community has reached a consensus on the majority of articles. To assess the semantic stability, appropriate software tool is designed and implemented in the course of this master's thesis. The tool is able to calculate the semantic similarity measure for the given plain text inputs where every text input is a revision of an article. Based on the calculated similarity measure of the consequent article revisions, the tool calculates the semantic stability of the whole corpus over a predefined time interval. The basic principles on which the proposed software solution thrives are well documented and explained in detail as well. The Wikipedia is used as a source of the plain text documents because it is a free, large, public corpus of documents. Thus, the semantic space being processed for the purpose of this paper is defined by the fact that Wikipedia articles are written in natural languages. All the phases of design and evaluation of the software tool are presented. This paper shows how the Wikipedia articles in different languages are parsed, stemmed and indexed. It also describes an efficient way to represent the semantics of Wikipedia articles as a matrix of double values that is used by Rank Biased Overlap method to calculate semantic similarity. The calculated similarity measure is consequently used to show the semantic stabilisation process of 10 different Wikipedia language editions. Finally, the attained data and results are discussed.

Keywords: Wikipedia, corpus, semantics, similarity, stability, TF-IDF, RBO

Kurzfassung

Die Masterarbeit setzt sich mit der Auswertung der semantischen Stabilität von Wikipedia auseinander. Die semantische Stabilität ist die Frage wie schnell sich Wikipedia Artikel mit der Zeit ändern. In einem stabilen System sind nur die langsamen Änderungen im Artikel bemerkbar, wobei in einem instabilen System sich die Inhalte häufig ändern. Die semantische Stabilität zeigt somit ob die Wikipedia Community einen Konsens über den Inhalt der meisten Artikel erreicht hat. Um die semantische Stabilität der Wikipedia herzuleiten, wurde ein passendes Softwarewerkzeug konzipiert und implementiert. Die Software kann das Maß der semantischen Ähnlichkeiten der vorhandenen Klartexte berechnen wobei jeder Klartext eine Revision der Wikipedia Artikel darstellt. Basierend auf dem vorher berechneten Maß der semantischen Ähnlichkeit der Wikipedia Artikel und Revisionen, berechnet das Softwaretool die semantische Stabilität für den ganzen Dokumentenkorpus für ein gewisses Zeitintervall. Die Hauptprinzipien auf denen die vorgeschlagene Software basieren, sind ausreichend dokumentiert und erläutert. Wikipedia wurde als Quelle der Klartexte verwendet da sie kostenlos, umfangreich und öffentlich Korpus ist. Der semantische Raum der für Zwecke dieser Meisterarbeit behandelt wurde, wurde mit dem Fakt, dass die Wikipedia Artikel in natürlichen Sprachen geschrieben sind, charakterisiert. Alle Phasen des Designes und der Evaluierung der Software sind dokumentiert. Diese Masterarbeit zeigt wie die Wikipedia Artikel in verschiedenen Sprachen geparsed, gestemmed und indexiert werden. Es wurde auch gezeigt wie man die Semantik der Wikipedia Artikel in Form einer Matrize darstellen kann. Diese Matrize wird von der *Rank Biased Overlap* - Methode zur Berechnung der semantischen Ähnlichkeit verwendet. Danach wird das vorher berechnete Maß der semantischen Ähnlichkeit zur Herleitung der semantischen Stabilität der 10 Wikipedia Editionen in verschiedenen Sprachen verwendet. Schlussendlich werden die erhaltene Daten und Ergebnisse erläutert.

Schlagwörter: Wikipedia, corpus, semantics, similarity, stability, TF-IDF, RBO

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

Place	Date	Signature

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Ort	Datum	Unterschrift

Acknowledgment

This Master's thesis was written in year 2016 at the Knowledge Technologies Institute at the Graz University of Technology. No master's thesis was ever written by itself so I just want to express my gratitude to everyone who has been helping me, no matter if it was help during experimental part of my thesis or support during the actual writing of the master thesis. Special thanks goes to my supervisor Assoc.Prof. Dipl.-Ing. Dr.techn., Denis Helic.

Graz, in April 2016 Darko Stanisavljevic

Contents

List of Figures

List of Tables

1 Introduction

1.1 Thesis subject

Semantic similarity of two textual documents expresses the extent to which those two documents deal with the semantically similar topics or content. This concept is a key to understand the comparison of documents written in natural language. Semantic similarity plays an important role in automatic text categorisation, summarisation, machine translation and data mining. Semantic comparison of short texts is important because short texts, in form of questions or queries, are today massively used on the web. Furthermore, a huge portion of news and information on the web is presented in textual form so that calculation of semantic similarity between them becomes even more important.

Typically, semantic similarity is calculated by using document statistics. An advantage of statistical approach is that there is no need for predefined models describing the meaning of particular words (terms). Substantial resources are usually required to define such models. The method described in this thesis, Rank Biased Overlap method, is also a statistical method and it is introduced in [13].

The basic procedure carried out during the calculation of the semantic similarity is the modelling of the semantic space in accordance with the term distribution in a corpus of documents. In such a space, each document is represented by a vector and semantic similarity is calculated by performing vector operations on those vectors. This approach is based on the distributional hypothesis, according to which the terms with similar meanings show tendency to appear in similar contexts [9].

The concept of semantic stability shown in this thesis implies tolerance to deviations or perturbations in time over a set of vectors describing the text input. As mentioned earlier in this section, different sources of content on the web could be used for this purpose and one of such sources is a free on-line encyclopedia Wikipedia which will be used for the purposes of this master thesis.

1.2 Goal of the thesis

The goal of this thesis is to measure semantic stability of Wikipedia. To that end, we develop a software which is able to take a corpus of documents in some of the predefined natural languages and to calculate and show the semantic stability of the input corpus. The corpus should contain a complete edit history for every document, thus containing all existing document revisions. The following Wikipedia language editions are used as the input corpus:

- English
- German
- French
- Spanish
- Italian
- Czech
- Finnish (Suomi)
- Danish
- Greek and
- Swedish

The intention behind the choice of these particular languages is to have five Wikipedia editions with a large number of articles and five smaller editions so that the conclusion about the semantic stability of Wikipedia can be correlated with the corpus size.

1.3 Research methods

A systematic approach to calculating semantic stability was used in this thesis. To that end, a combined theoretical-experimental approach was used. In the course of this master thesis, we did the following:

- First, we collected the required state-of-the-art literature.
- Secondly, design and synthesis based upon the acquired knowledge and experience were done.
- Thirdly, we applied the quantification methods to evaluate and discuss the results.

1.4 Expected results

The results expected to be provided in the end are:

- to have a master's thesis paper defining the problem domain of semantic stability and describing the proposed software solution as well as the overview of the used software libraries and theory behind semantic stability calculation
- the design and implementation of the software solution for semantic stability calculation
- a conclusion based on experimental results achieved from Wikipedia article corpus
- a clear picture about the usability of the proposed software solution in praxis

1.5 Thesis structure

This master thesis is divided into six chapters. The first chapter provides a short introduction into the subject of semantic stabilisation and defines the goals of the thesis. The overall plan section of the first chapter states basic building blocks of the proposed software solution as well as their relations. It shortly describes the used software libraries and theoretical concepts behind the proposed software solution, thus giving the short overview of both:

- theory: semantic similarity and stabilisation methods and
- existing software libraries

The second chapter goes on to provide a detailed description of the theory behind the *plain text to vector space* conversion, semantic similarity and semantic stability calculation methods. The Wikipedia dataset is elaborated in this chapter in detail, as well. The next chapter, Data preprocessing, offers a detailed documentation about:

- the design and architecture of the proposed software solution
- software libraries and concepts used

The results which were attained by applying the proposed software solution on the dataset can be found in chapter four. The last two chapters contain the summary of the work done throughout the course of the master's thesis as well as the conclusion and the improvement proposals for the proposed software tool.

1.6 Overall plan

This section offers a brief preview of the theory topics as well as software libraries and concepts that had to be covered for the purpose of the software implementation presented in this paper. It will give the reader a better understanding of how different theory concepts and existing software solutions are intended to work together in robust software solution. The following chapters will discuss these topics in more detail.

1.6.1 Theory

Particularly important for this thesis is the theory describing:

- the evaluation of importance of terms in a single document or in a corpus of documents and their representation in the form of matrix - TF-IDF
- the calculation of semantic similarity measure
- the calculation of semantic stability over time

If the goal is to extract the semantics (meaning) of a single document, it is necessary to find out the most frequent terms in a given document. First, all stop words have to be ignored. In the discipline of information retrieval, stop words are the most frequent words which do not have any relevance to the meaning of document. For example, some of the stop words in English are: a, the, that, to, as and so on. When stop words are ignored, TF-IDF matrix can be calculated. Each row in this newly created matrix stands for a single document and each value in a row represents a weighted TF-IDF value of a term. The higher the value of term, the higher the semantic relevance of the term.

This thesis is based on the paper [1] which describes a method called Rank Biased Overlap or shortly RBO. The mentioned method is used to calculate the similarity measure of two given vectors, each of them representing the rankings of terms contained in a single Wikipedia article. Its main characteristic is that it takes the cumulative overlap of the given rankings as a measure for similarity.

The similarity measure mentioned in the previous paragraph is used as the basis for calculation of the semantic stability over time. This novel method, introduced in [1] states that for a given value of RBO threshold, an article is semantically stable if its RBO value at the point of time t is equal or higher than the threshold.

1.6.2 Implementation

Wikipedia, with its multilingual versions, is one of the most comprehensive, non-generic data sources containing millions of plain text articles, tables and different kinds of multimedia material. It has to be stated that data provided by Wikipedia acts as a live data because the number of revisions of a single article, at the moment of writing this thesis, averages out to 22. With the total of 21 million users actively contributing to its contents, the fluid nature and growth of this greatly diversified data source is guaranteed. The English version of Wikipedia alone has about 5 million articles. All these hard facts promote Wikipedia as the ideal data source for evaluation of different algorithms from the field of information search and retrieval as well as the information visualisation, data clustering and search engines. Furthermore, it is the ideal data source for extraction, vectorization and processing of the semantics - *meaning* of the plain text content, which is of great relevance to this thesis. Wikipedia articles with complete revision history are available for on-line download from official Wikipedia web site.

In order to get Wikipedia data, one can use the functionality of the available API or simply download XML dump files from the official web site. Because of the tremendous amount of data available, the use of API is appropriate only when a limited amount of articles is needed. It is not meant for this API to be used when full corpus of articles is needed. Typically, the available API should be used to get, for example, a list of articles belonging to specific category, a list of subcategories of a given category; it should be used to get only abstracts of articles needed and so on. For the purposes of this thesis, downloading XML dump files was a sound solution. Dump files for every single Wikipedia project are available on-line and updated at least once every month. For every language version of Wikipedia, there are several types of dump files with different content: abstracts only, a current version of all articles, a complete history of all articles. Those files, for some language editions, are tens of gigabytes big.

Once the input data is provided, it is necessary to represent the plain text data of Wikipedia articles in such manner that it is possible to perform mathematical operations on this data. One possibility is to generate TF-IDF matrix representing the whole corpus or only a sample of all articles. *Term Frequency - Inverse Document Frequency* is one of the methods in theory of Information Search and Retrieval used to represent the relevance of terms in a document belonging to a collection of documents - *corpus*. Different variations of this method are used in search engines as the primary tool to rank search results upon user text query. One of the hardest problems that had to be solved during implementation of the software for this thesis was to efficiently process millions of Wikipedia articles and generate complete TF-IDF matrix. This functionality of great importance is achieved by using two Apache libraries: Lucene® and Mahout®. The exact usage will be explained in detail in 1.6.2. Lucene® is an open-source, high-performance, full-featured text search engine library completely written in Java. In addition to the necessary functionality, it is a highly scalable solution which is exactly what is needed when processing millions of plain text articles. It does a complete

text analysis, stemming, lemmatization and that is just a small portion of what it actually can do. Mahout® is the second Apache library used. It basically takes a document index generated by Lucene® and vectorizes it, thus generating TF-IDF matrix. The symbiosis of these two great tools leads to an absolutely scalable solution used to process such a huge amount of plain text data. It can be deployed on a standard personal computer or a cluster by simply changing the configuration of both libraries. Mahout® works on the principles of *MapReduce* programming model that is associated with processing and generating large data sets with a parallel, distributed algorithm on a cluster. This useful library uses yet another Apache product named Hadoop® to write so-called *sequence* files, containing vectorized representation of Wikipedia articles. This format is usually used as a parallel input/output in MapReduce. Hadoop® is, just like the other two mentioned Apache libraries, an open-source project. It is n framework that allows the distributed processing of large data sets across the cluster of computers in an entirely scalable manner.

At the moment when the input processing is over, vectorized data is stored in the database. MySQL® is used as a storage solution because of its reliability, its read and write speed and the possibility of using bulk writes. Since it is widely used, community support for this database solution is really great, which was one of the main reasons to use this particular solution. On top of the database, an ORM (Object-relational mapping) is used in order to abstract database related code and make it more readable and portable. The complexity of data stored in the database is not high so there was no need to use some of the more advanced features provided by some other relational data base system.

All this data processing up to this point was in favour of preparing plain text article representation to be suitable for article similarity calculation. If all the revisions of a single Wikipedia article are taken, sorted according to the time when they were created and then transformed to TF-IDF vectors, it is possible to calculate how similar each consecutive pair of vectors (Wikipedia revisions) is. Thus, the semantic stability of article revisions over time is described. If this calculation is done over the whole corpus or a sample of a language edition of Wikipedia, it represents the semantic stability of the chosen Wikipedia edition. The comparison of the vectors is performed using RBO (Rank Biased Overlap) method proposed in [1]. This paper represents the cornerstone of this master's thesis by providing and elaborating the semantic stability measure.

One of the main aspects the software implemented in this thesis had to fulfill was the data processing efficiency. It is achieved through:

- multi-threading and
- special libraries that address the problem of large amount of data to be processed like Mahout® and Hadoop®

One typical example for the use of adequate libraries is the use of the Mahout® library to calculate TF-IDF vectors from the existing Lucene® index. Millions of terms contained in index are used to calculate very long TF-IDF vectors composed of decimal values (which

can add up to several thousand values). Without MapReduce mechanism implemented in Hadoop®, the processing performance would significantly drop. A good example of how multi-threading is used is the mechanism for indexing plain text articles with Lucene®. A plain text is read in parallel from several XML dump files at the same time and so, in a concurrent manner, indexed by Lucene®.

2 Technical background

2.1 Data indexing

Plain text in its raw form can not be interpreted by search algorithms as it is, but it has to be previously mapped to some other suitable form in the procedure of *data indexing*. This is the exact purpose of a text processor. It prepares, processes and indexes input data. Many problems are encountered when large amounts of data have to be processed. Even the simplest tasks become hard to solve in such situations. When lots of documents are to be processed, it is hard to recognize some typical structure common to those documents. One of the main problems by parsing is to recognize different types of indexable elements called *terms*. The way a text is interpreted by a processor depends upon the selection of terms being recognized as important by the processor. The input, plain text, is represented as a histogram of term occurrences in text thus building lexical semantics. In order to create the histogram, the processor has to undertake some or all of the following steps:

- normalize the document stream according to the predefined format
- split the document stream into desired searchable units
- identify potentially indexable terms in documents
- optionally, identify phrases made of more than one word and use them as indexable element
- remove *stop words*
- reduce terms to their basic grammatical form - *stem*
- extract indexable elements
- calculate *weight factors* of indexed terms

All this steps are important because of the fact that only indexed terms can be later used to search the text or to represent the semantic of the available plain text documents. Figure 2.1 illustrates some of the above mentioned steps. The text documents have extremely dynamical lexical content possibly containing hundreds of thousands of characters, symbols, punctuation marks, numbers and so on. This is the reason why it is hard to identify terms to be indexed. Also, it is not only important for a text processor to use different parsing rules; it also has

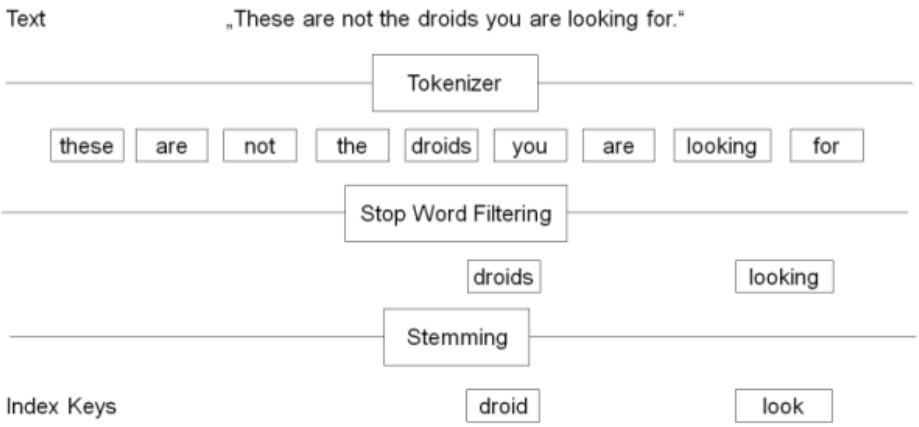


Figure 2.1: An example of how the plain text input in a form of a simple sentence has been tokenized, filtered and indexed.

to be able to select only those terms being semantically relevant without losing valuable information. This compromise between indexing only a subset of the tokenized data and not losing relevant information is one of the real challenges the document processors have to deal with.

2.1.1 Normalization and identification of indexable terms

The simplest and the most obvious way to tokenize the text is to use white space characters as a character to split upon. This approach is known as a *bag of words*. The name comes from the fact that the text is seen as a bag of words, completely diminishing the importance of sentence composition thus not taking into account the so-called *compositional semantics*. In simple terms, the order of words is completely neglected, which is contrary to the fact that the word order in a sentence defines the meaning of the sentence. When the indexable terms are identified, it is hard to know which words or groups of words are relevant enough to be indexed. The font size of some of the indexable terms in plain text could implicitly say something about those terms. Titles are, normally, written in bigger font so it could be a notion of the relevance of the term. One of the approaches is to assign higher relevance to the terms written in larger font size because of the fact that the abbreviations of organization names and some concepts are distinguished that way. Nouns usually have the greatest semantic relevance and other types of terms can be removed. In order to distinguish between different types of words in text or to find the basis - *stem* of the word and so on, it is necessary to process the text according to the linguistic rules of the actual natural language of the input text.

2.1.2 Basic terminology explained

One word in any natural language can take many different forms which are grammatically distinct but still have the same meaning such as, for example, *do* and *doing*. For every form of the word it is possible to define a canonical basic form called *lemma*. Sometimes it is not possible to lemmatise a group of words in such manner that all words have a unique lemma. Lemmatisation is the process of bringing the word having any given grammatical form to its basic form.

Stemmer is an algorithm used to derive a basic form, root or *stem* of a word currently inflicted by some grammatical transformation. The vast majority of existing stemming algorithms remove the suffixes of a given word according to certain rules so that the resulting term is not always the correct word stem but merely its approximation. For example, for word "dries" stemmer will output "drie" and not "dry" as it should be. The most famous stemming algorithm for stemming English is the algorithm designed by Martin Porter. At the same time it is the most common English stemming algorithm so it is *de facto* standard English stemmer. It has to be stated that the implementation of a stemmer for morphologically simple language as English is much easier than for some other languages (Hungarian for example). Figure 2.2 shows several results of different stemming algorithms.

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Lovins stemmer: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Paice stemmer: such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

Figure 2.2: The outputs of several different stemming algorithms processing the same input text.

Stop words are words having almost purely grammatical purpose and they are very common in a document. Stop words are not to be found in the list of lemmas. Also, stemming

and lemmatisation will not be done on those words. Some examples of stop words in English are: *the, a, I, when* and so on.

Ignored words are usually the words common for some topic and do not bring any additional information to the index.

English is, in the morphological sense, a poor language with simple grammar rules and as such it is highly suitable for automated text processing and indexing. Solving the problem of lemmatisation and stemming is the most important requirement for any kind of automated text processing for any natural language. For most common world languages, especially for English, there are many tools and algorithms available. Text extraction is not related to the process of text document tokenization but is a necessary step towards it.

2.1.3 Text extraction

Depending on the data source, the text document can be formatted in many different ways. Prior to the text processing, the plain text data have to be extracted from document. For example if text is given in XML file, all the XML mark-up data have to be removed. So, it can be concluded that different data types could have different meta-data that have to be filtered out. This process of data filtering obviously can lead to data loss in some extent. Formatting of the textual content means the some more important parts of the text, as title, are written in larger font or in different color. When plain text is extracted, this information is lost.

2.1.4 Tokens

Individual text characters without any context information are not useful at all and they have to be grouped in units of text called *tokens*. A token can be a single word from the text - a **term** or it can represent a sequence of characters, phrases, e-mail addresses and so on. A token is the smallest unit of text having both of the following two properties:

- linguistically relevant and
- methodologically useful

The first requirement for token is pretty much self-explanatory and it supports the statement from the beginning of this paragraph. The other requirement states that it is not only important to check if there are predefined delimiters on both sides of the string in order to recognize a token but rather to search for patterns thus finding the semantically relevant collocations in a given text in a given language [6]. Tokenization is the so-called bottom up approach to text processing, but it is not the only way to identify tokens. Some other methods include segmenting the input text in sentences and than trying to extract tokens from previously segmented sentences. The problem with this approach is that the identification of sentences

in text is not always an easy task. For some text sources it is not even possible to divide it into sentences at all. Another problem is parsing the found sentence because such parsing does not provide the results with the same meaning. Sentence can be parsed so that there are several output sets of tokens. Different output token sets do not describe the semantics of the sentence equally well so there is a problem how to choose the best token set representing exactly what the author of the text meant. All this implies the usage of semantic recognition to identify the sentence, and by means of performance it is not the best way to get the tokens. The task of extracting the *meaning* and understanding the text is done on much bigger units of text [8, p. 21-69]. So, there are two kinds of tokenization:

- low-level tokenization and
- high-level tokenization

Contrary to low-level tokenization, the so-called high-level tokenization implies the use of some kind of higher level text recognition even if it is a shallow linguistic processing. Without this step it is not possible to know if two or more neighbouring words should stay together as a single token or if those should be separate terms. Figure 2.3 is the simple example of tokenization.

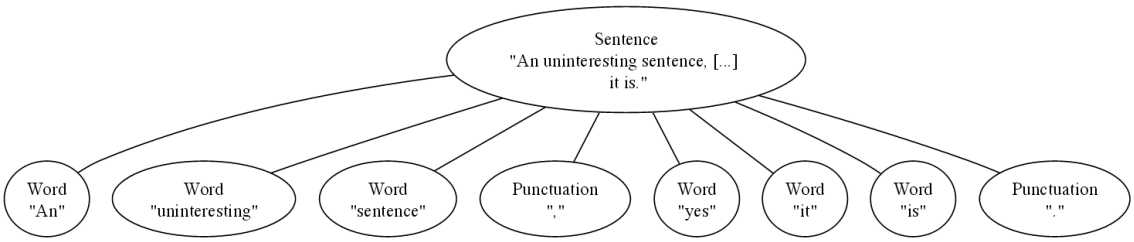


Figure 2.3: A simple example of tokenization.

The importance of good tokenization algorithms lies in the fact that the list of text tokens for a given documents is used as a cornerstone for many applications. The most prominent applications based on processing the text into tokens and indexing the tokens are *search engines*. When a user defines a query, the query is tokenized and afterwards compared with the list of indexed tokens. The same principle is used in a software used for plagiarism detection or simply for comparing given text documents. Also, the list of tokens representing a document is sufficient to classify the document according to its topic.

2.1.5 Indexing

As the amount of data available on the web is getting increasingly large, the need for search machines is growing too. A search machine takes the user query, tokenizes it and compares

the query tokens to the tokens representing indexed on-line documents. The quality of the user query result is proportional to the quality of indexing procedure applied on documents already indexed. By taking into account the previous sections of this chapter, it can be concluded how important the document preprocessing is in order to successfully index a large amount of data. In the ideal case, the document preprocessing should eliminate all the non-indexable elements and provide a list of tokens. The indexing application should choose which tokens will be indexed. The most simple way of doing this is by indexing every single token, but when huge amounts of data have to be indexed this is obviously not the best solution because of the enormous data storage requirements. In the case of English Wikipedia corpus, one can not propose indexing the full content of the XML dump file. As explained later in this paper, an XML dump file contains Wikipedia mark-up language markers that have to be previously filtered out together with all the terms in document that do not contain any relevance to the semantics of the given document. This section deals with proper indexing techniques.

Traditional indexing

One of the traditional ways to index terms from a document is to add those terms to index together with their position in respect to one another in the document. For example, if there are two terms to be indexed, both of them will be added to the index and their position in respect to each other will be stored as well. This type of indexing, although very popular at the beginning because of its simplicity, leads to an extensive use of data storage resources. This sort of indexing was easily used by search engines. For the two terms query, the search engine would highly rank the document containing both words appearing together. Keeping a lot of additional data in storage made this indexing technique not exactly appropriate for use by search engines. Processing the queries was not efficient enough because of the large amount of additional data.

From this observation, it can be concluded that keeping the track of the words appearing per document is not the best idea, especially if a very large amount of data has to be indexed. The solution is the *inverted index*. This concept is really widely used and there are several variations. The most relevant among them are:

- the simple inverted index and
- the full inverted index.

There is a list of words appearing in all documents in corpus and for every word in the list, there is a list of documents it appears in. This makes the document corpus search very efficient. This index structure is also optimized because of the fact that for a single search (term query) a list of all documents in corpus containing the searched term is returned. This implementation is very similar to the concept of a look-up table. Figure 2.4 illustrates the previous statement. For three given text documents, first the stop words are removed and then, the identified terms are indexed. Every term has its list of document occurrences, so the

look-up table is generated. When the table is queried against user search terms, the list of document occurrences for every term in the search query is returned and then the lists are compared to each other (intersection). The document contained in all the returned lists is the highest ranked search result. In case of the given example, for search query *blue sky*, the highest ranked result is the reference to *Document 2*.

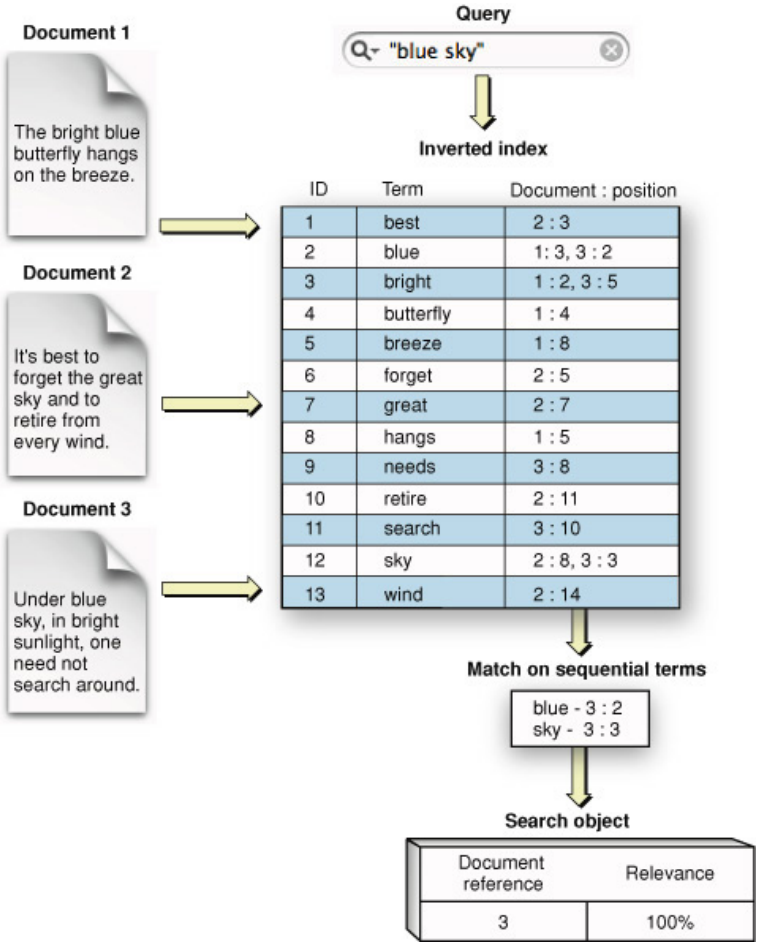


Figure 2.4: An example of how the inverted index is created. It also shows the processing of the search query as well as the results.

The second implementation of this inverted index concept is the so-called *full inverted index*. In this concrete implementation, every term in index has a list of documents it appears in as well as its position in those documents. The table 2.1 illustrates full inverted index for the given documents:

- Document 1: "Basketball is played during summer months."
- Document 2: "Summer is the time for picnics here."

Term	Frequency	Documents IDs and positions
basketball	1	[(1,1)]
during	1	[(1,4)]
found	1	[(3,4)]
here	2	[(2,7)],[(4,6)]
rainy	1	[(4,5)]
is	3	[(1,2)],[(2,2)],[(4,2)]
months	2	[(1,6)],[(3,1)]
summer	3	[(1,5)],[(2,1)],[(4,3)]
the	1	[(2,3)]
why	2	[(3,5)],[(4,1)]

Table 2.1: The content of the full inverted index for the given text documents.

- Document 3: "Months later we found why."
- Document 4: "Why is summer so rainy here."

2.2 Document representation and Information retrieval models

Today, the Internet could be seen as an ultimate source of data of any type. The sheer amount of data freely available on the web and the fact that the amount of data available exploded in the resent years and that the data available is not structured in any way implies that there is an urgent need for better and more efficient web search engines in the coming years. If only a portion of available data is needed, as it is always the case, there is a need for some kind of data retrieval model according to the given user query. As explained in section 2.1, all the data has to be indexed first and then properly represented. This section describes the basic concept of information retrieval and some of its main models.

2.2.1 Information retrieval in general

The term *Information Retrieval* in the world of computer science represents the activity that selects only the most relevant documents and preferably all the relevant ones from the given set of documents where the selection criteria is a user search query. The measure for the quality of the selected documents assumed to be relevant is called *precision* and the measure for the completeness of the retrieved documents is called *recall*. Although the term Information Retrieval can be confused with the term *Information Extraction*, these are two completely different concepts. Information extraction deals with the extraction of

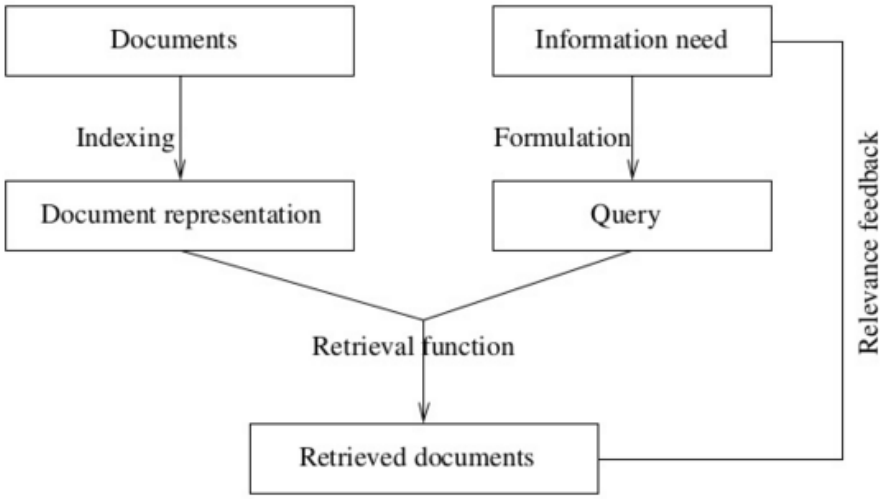


Figure 2.5: A general information retrieval model.

the meaning of some given text. Information Retrieval provides only the subset of given documents (goal is to find relevant documents) but in this activity it is not necessary to understand the meaning of the text. It simply uses the similarity of the search query terms and document terms of the indexed documents. This particular concept is illustrated in figure 2.5.

Previous section 2.1 explains how the documents are indexed and it is mentioned that, in accordance with the amount of data having to be indexed, the documents can be represented twofold:

- full-text representation
- reduced (partial) content representation

Although the full-text representation is the most complete representation providing the optimal performance by the means of precision and recall, when the amount of data to be indexed is as large as is the case in this thesis (Wikipedia corpus), the reduced content representation has to be used. As mentioned in the previous section, prior to indexing, documents are cleansed (stopwords, stemming and so on) and as such may not have the same meaning expressed by the indexed terms only. Hence, finding similarities between the document and the query terms is inherently flawed.

There are two classical information retrieval models:

- standard boolean model
- vector space model

Both of the named classical information retrieval models will be explained in the next two sections of this paper, but the special emphasis will be on vector space model as it is used in the implementation of the software solution for this master's thesis.

2.2.2 Standard boolean model

The standard boolean model is one of the most simple information retrieval models and it is based on the rules of boolean algebra. This model uses statements defined by the rules of boolean algebra as queries with precise meaning. The standard boolean model has been known in the recent years for several serious conceptual flows setting limits in real life applications. The first huge flaw of this information retrieval model is the fact that the strategy of this model is to decide, in a binary manner (yes or no), if a document from the collection of documents is important for user query. Hence, there is no ranking of documents by their relevance. In most cases, this strategy leads to too many or too few results recognized as important among all the given results. Another problem with this model is the fact that defining precise user queries in form of a boolean algebra statements is not always an easy task for every user. Although such query is very precise, it requires some expertise in the field of boolean algebra from the user, especially when it is about to write a more complex query. This is obviously the reason why the user queries are, in most cases, pretty simple and straightforward. Despite its disadvantages, this model is still a common model in many solutions nowadays, especially in bibliographic systems. As such it is a good basis for better understanding of information retrieval models paradigm.

2.2.3 Vector space model

Nowadays, the vector space model is probably one of the most common and one of the most popular models among the researchers in the field of information retrieval. When this model is compared to standard boolean model, the main difference is the fact that every indexed document term has its own *weight*. The term weight can take any real number value. The same is with query terms. The degree of similarity between indexed terms and query terms can be calculated and the similarity measure takes a real value in range [0,1]. Once the similarity between indexed documents and user query are calculated, the retrieved documents can be sorted by the relevance for user query.

The representation of documents with the vector space model is also known under the name of *bag of words* representations. The name is derived from the fact that it is assumed that indexed terms are independent from each other. This assumption also implies the loss of information about the correlation and position of indexed terms in documents in reference to each other.

In this section, the author of the paper will explain how the vector space model is used in real life and an adequate example will be included as well.

The vector space is defined by the number of unique index terms. An |V|-dimensional vector space consists of n vectors of length V where V stands for the number of terms (words). When terms are extracted from documents, a dictionary of those terms is created implicitly, and the previously mentioned V stands for the number of words in the dictionary. So, the terms are the axes of the vector space and documents are the points in that space, or the vectors connecting the origin of the vector space and the point. Hence there is exactly one vector for each of the documents from the collection of documents. So, in the case of web search engines, the vector space has very high dimensionality. In real life it could easily have tens of millions of dimensions. The crucial property of these vectors is that they are very sparse vectors, meaning they consist mainly of zeros. Each vector of the vector space has a value for each word in index dictionary. Since only a subset of words from the dictionary are contained in a single document, only the terms contained in the document will take a non-zero value. A single document typically has a few hundreds or thousands of words. Figure 2.6

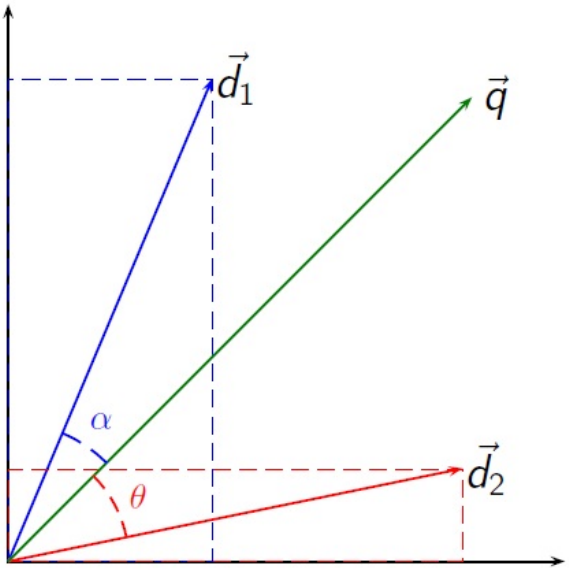


Figure 2.6: A Vector Space Model for two given documents d1 and d2 and query q[1]

depicts how different document vectors can be represented in a vector space model together with query vector. Figure 2.7 shows the *Term Document Matrix* which is a mathematical representation of the vector space model. Every column of this matrix corresponds to a document vector in the vector space model and every row of the matrix stands for a single index term from index dictionary.

[1] The figure shown is taken from Wikipedia and can be found on `https://en.wikipedia.org/wiki/Vector_space_model`, 07.01.2016

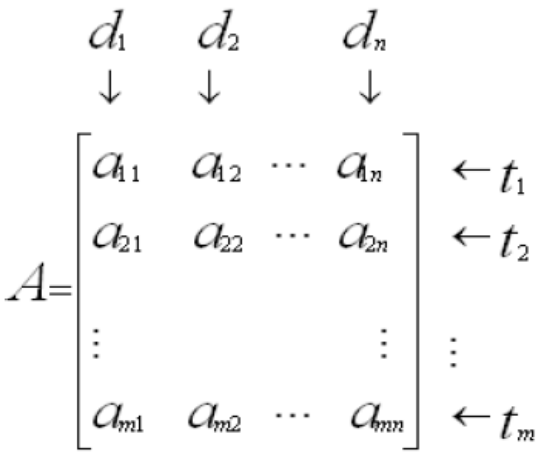


Figure 2.7: An illustration of the generalized Term Document Matrix. Every column of this matrix corresponds to a document vector in the vector space model and every row of the matrix stands for a single index term from the index dictionary.

So then, if a vector space model of a collection of documents is calculated, how should the user query be handled? The key idea is that the user queries are treated in the same way as documents. They are the vectors in the same vector space as the indexed documents. From all the facts mentioned in this section, two key ideas are relevant:

1. to represent user queries in vector space in the same way as documents
2. do the document ranking with regard to their proximity to the query in this space

As shown in figure 2.6 the proximity between thw query vector and the document vector corresponds to the similarity between those vectors and it is roughly inverse of the distance between them. If the reader of the paper recalls section 2.2.2, it is clear what the main difference between vector space model and a standard boolean model is. The first one is able to rank the retrieved documents by their similarity to the user query so that there are the retrieved results that are more important than the others. The standard boolean models approach *either-in-or-out* is far less flexible when it comes to ranking the results.

Now, when the representation of the document vectors in vector space model is well defined, it is necessary to define a way to calculate similarity between a document and query vectors. When the angle between two vectors is zero, it corresponds to maximal similarity. This implies the refinement of the second key idea which now states: do the document ranking according to the angle between document vector and query vector.

Cosine similarity

Once the documents are represented in the vector space model as explained in the previous section, it is necessary to quantify the similarity between two vectors from vector space model. In a vector space model with possibly millions of document vectors, cosine similarity can be used as a measure of similarity between two vectors defined as a cosine of the angle between those two vectors. When the angle between vectors is 0°, then the cosine of the angle is 1, meaning there is a maximal similarity between vectors so the vectors are identical. When the angle between vectors is 90°, then the cosine of the angle is zero and there is no similarity between the vectors.

- if $\angle(\overrightarrow{A}\,\overrightarrow{B}) = 0^\circ \implies \cos\angle(\overrightarrow{A}\,\overrightarrow{B}) = 1$, similarity is $1 \rightarrow$ vectors are identical
- if $\angle(\overrightarrow{A}\,\overrightarrow{B}) = 90^\circ \implies \cos\angle(\overrightarrow{A}\,\overrightarrow{B}) = 0$, similarity is $0 \rightarrow$ vectors are not similar

The mathematical expression 2.1 states that the similarity measure between vectors $\overrightarrow{A}$ and $\overrightarrow{B}$ can be calculated when the inner product (dot product, scalar product) of the named vectors is calculated and afterwards divided by the product of the magnitudes of those two vectors.

$$similarity = \cos\theta = \frac{A \cdot B}{\|A\|\|B\|} \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}} \tag{2.1}$$

The value of the cosine similarity measure is influenced by the angle between vectors and not by their magnitude. As all the weights in the document vectors are positive values, the cosine similarity measure takes the value on interval [0,1].

2.2.4 Term weighting: TF-IDF

One of the biggest disadvantages of the *Standard Boolean Model* is that the vector containing the terms of the document it represents does not give any information about how important those terms are in the context of the document. It simply states if the term is or is not to be found in the document. On the other hand, the *Vector Space Model*, by its definition, does not specify any weighting schema. So, basically, it is up to the specific user to decide how the terms will be weighted. *Term Weighting* represents the idea of assigning weights to the terms of a vector in the vector space. Each term of the document, in accordance with its specific position in the context of the analysed document, is of different importance. For example, the terms contained in the title of the document should have higher value as it is probable that those terms are more important for a correct representation of the document.

There are numerous term weighting algorithms. One of the term weighting possibilities is the so-called *Term Frequency - Inverse Document Frequency* weighting schema or shortly TF-IDF. This schema is one of the most used and well understood existing term weighting schemas [10].The three most important assumptions constituting this term weighting schema are [2]:

- the high frequency of the term does not automatically imply that this term is very important. Rare terms are not implicitly of a less importance
- multiple appearances of the term in a document do not automatically imply that it is less important than the term appearing only once
- long documents should not be seen as more important than the short ones

Term Frequency - Inverse Document Frequency is defined as a product of *Term Frequency* measure and *Inverse Document Frequency* measure. Those measures are more precisely described in the following two sections.

Term Frequency

Term Frequency represents the number of occurrences of the term t in document d. There are several ways to represent this measure but the most common methods are:

- raw representation:
$$tf(t,d) = f_{t,d} \tag{2.2}$$
where $f_{t,d}$ is the raw number of term occurrences in document
- boolean representation:
$$tf(t,d) = \begin{cases} 1 & \text{if } t \text{ found in document } d \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$
- logarithmically scaled representation:
$$tf(t,d) = \begin{cases} 0 & \text{if } f_{t,d} = 0 \\ 1 + \log(f_{t,d}) & \text{otherwise} \end{cases} \tag{2.4}$$

Normalization

If any of the previously mentioned term frequency representations is used, it becomes obvious that the terms found in very long documents will have higher frequencies than terms from shorter documents in general. This implies that all term frequencies have to be normalized. This way, the terms from longer documents are not preferred over the ones from the shorter documents. The most simple way to normalize term frequencies is to divide the term frequency of each term in the document by the total number of terms in the document.

Inverse Document Frequency

As stated in the previous section, the terms with high frequency in a single document from the collection of documents should be weighted with higher values as they probably give much information about the document. But if the term frequency is high in every single document from the corpus of documents, it implies that the term is probably a *stop* word. For example, the stop word *the* is the very common term in every document in the corpus of English documents. Because of this fact, the *Inverse Document Frequency* is used as a factor that diminishes the weight of the terms with high frequency among all the documents of a given document set. For a corpus of documents D, where the number of documents in corpus is denoted with N_D, Inverse Document Frequency is defined as:

$$idf(t,D) = \log(\frac{N_D}{n_i}) \tag{2.5}$$

that is, the logarithmically scaled result of a division of the total number of documents in the corpus by the number of documents in the corpus containing the term.

2.3 Semantic similarity

This chapter, together with chapter 2.4, represents the cornerstone of the author's master's thesis. It gives the reader a rather detailed dissection of the novel method for comparing two or more ranked lists with each other. This entire chapter is based on the scientific paper *Similarity Measure for Indefinite Rankings* written by W.Weber, A.Moffat and J.Zobel [13]. In that paper, authors gave an overview of the existing similarity measures and they proposed a novel method for comparing indefinite ranked lists. That novel method, developed and introduced by the previously mentioned authors is called *Rank Biased Overlap*. The application of this novel method in praxis as well as the analysis of the results attained by this method are shown in the paper titled *Semantic Stability in Social Tagging Streams* written by C.Wagner, P.Singer, M.Strohmaier and B.Hubermann [1]. This paper is of great importance for this thesis as well because it introduces some changes to the original *Rank Biased Overlap*

method. How important the topic of comparison of the lists for this thesis is will be shown in some of the following chapters handling the software implementation details.

Comparing two or more ranked lists is often very important. Only one simple example would be comparing two ranked lists that are a result of the execution of the same user query on two different search engines. When two lists are compared, it is important for the comparison algorithm to be able to:

- handle non-conjoint lists
- assign more importance to the elements of the list having higher weight
- keep monotonic property as the depth, up to which two ranked lists are compared, increases

The ranked-biased overlap comparison method fulfills all of the above-mentioned requirements. Before going further, the term *conjoint lists* has to be clarified. The two lists are said to be conjoint if and only if both lists are made of the exact same elements, where the order of elements is irrelevant.

At the very beginning of this chapter, one can ask why it is so important to be able to compare lists at all. The aforementioned example of comparison of the search engine query result lists is just one example. In every day life, a lot of data is presented in the form of a list, in the newspapers or magazines, name entries in a phone book, a list of best-sellers, a list of most popular celebrities of the year in magazines and so on.

Several properties characterize the nature of the lists that are to be compared with the rank-biased method. What is common for all these lists is that all of them are *incomplete*, meaning the elements of all lists together do not represent the entire set of elements defined by a domain. The list of the book best-sellers, for example, has ten entries and not the complete list of all the literature ever written. Another characteristic property of the lists is that they are all *top-weighted*. That is, the top (or the first) element of the list is always the element with the highest weight. The last element of every list is the element with the lowest weight value. This implies that head is more important than the tail, hence the name *top-weighted*. Third characteristic of these lists is that they are *indefinite*. Basically, the user is eligible to truncate all the elements of the list from some arbitrary depth until the end of the list. Contrary to this, another possibility is to use all the elements of the list for specific application, but the importance of the elements is inversely proportional to the depth of the element in the list. The processing costs for the elements at the end of the list are probably much higher than the benefit the application has from the existence of these high depth elements. A good example of this statements is, again, the search engine query result list. Once the user gets the list, a number of the top-ranked search results will be scrolled, usually the first few pages, and the rest of the results are mostly irrelevant for the user.

The tree properties of the lists described in the previous paragraph are related and implied by each other. The list is *top-weighted* so the weights of the list elements are decreasing with

the increasing depth. The decreasing weights are the reason why the user truncates the list at some depth. The truncation of the list by the user explains the incompleteness of the list.

The previously mentioned example of a search engine query result is a really good example why comparing ranked lists is important. The importance of ranked list comparison leads to a conclusion that a *rank similarity measure* is needed. It allows an objective and repeatable comparison of ranked lists. Such measure has to be able to handle such lists where an element of one list is maybe not included in the other list. It should differentiate and assign more value to the similarities of higher ranked list elements than to the similarities of the elements placed at the end of the lists. The authors of the paper [13] state that the *rank-biased overlap* method is the first method for calculating the similarity measure that is fully appropriate for measuring the similarity of indefinite rankings. At this point, it is important to describe the term *indefinite ranking* in more detail. The most important thing when working with indefinite rankings is the fact that although the list is said to be indefinite, only its prefix is taken into consideration when the calculation of similarity is done. Prefix can be defined as a small part of the list containing the top N list elements. The most important property describing the prefix is the size (length) of the prefix and it can essentially be arbitrary. The good way to choose the prefix length is to take some arbitrary length of the prefix and to check the similarity measure for the prefixes of the two given rankings. If the similarity measure value is high, then the prefix length has to be increased and the other way around; if the similarity measure value is lower, the length has to be decreased. So, the main aspect of measuring the full rankings is to select prefixes of the rankings so that the similarity measure of the prefixes of the two full rankings corresponds to the similarity measure when full rankings are analysed.

The basic idea behind this *similarity measure* is pretty simple. The idea is to calculate the overlap between two ranked lists up to some depth, where the depth is increasing in every iteration. The maximum depth is then parametrized by the user. After the overlap evaluation for each depth, the user has a fixed probability that the calculations is over. This fixed probability of calculation is represented as a Bernoulli random variable. The similarity measure *rank-biased overlap* is then calculated as a mean of the overlaps of the ranked lists at each evaluated depth. As the input parameter it needs the user's *persistence*, as the probability for going from evaluation of one depth to the next depth is defined. When probabilities for each of the ranks are multiplied, one gets the probability that the user will reach a certain rank and this value represents the weight of the overlap. When the values of the weights are considered, one can conclude that those values are geometrically decreasing but will never converge at zero, thus showing once more the *indefinite nature of the ranking*.

The measure that can handle rankings with all the properties described in this section, that are:

- top-weightedness
- incompleteness

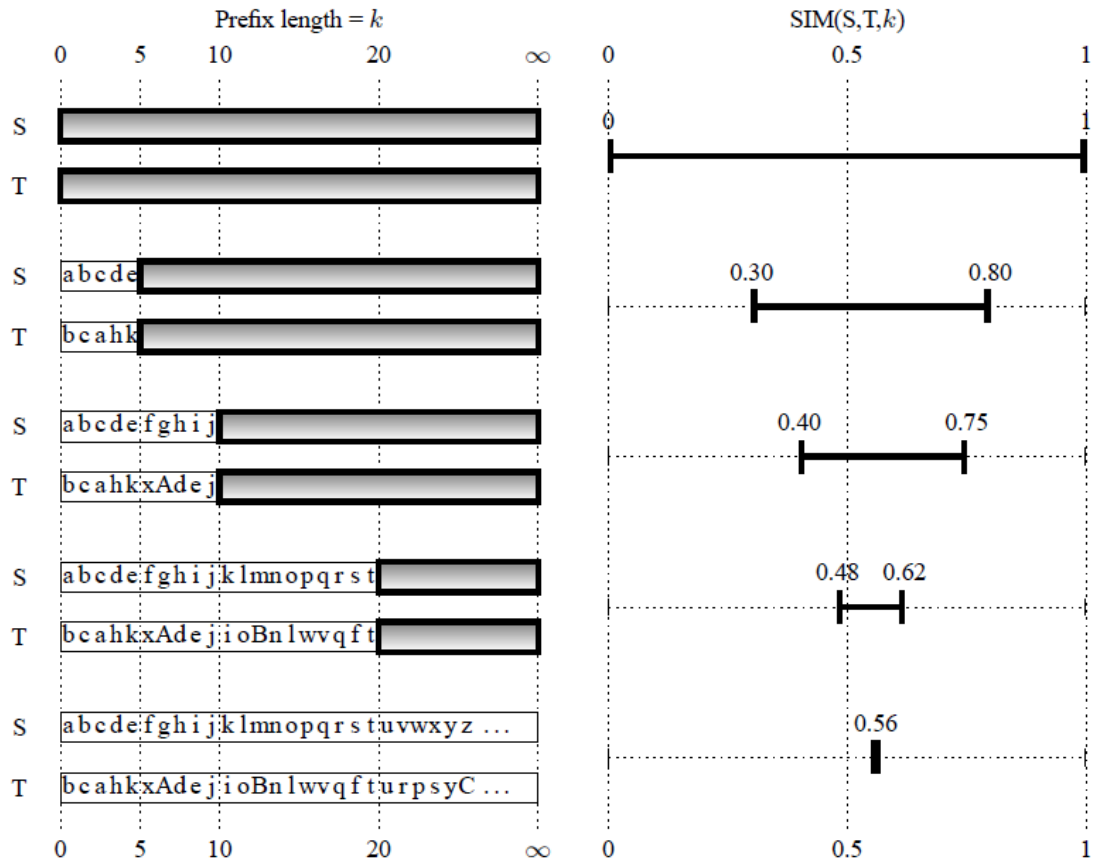


Figure 2.8: Prefix length in relation with rank similarity. At first, the depth of the compared rankings is zero - that is nothing is compared, so the similarity is anywhere on the interval [0,1]. With the increased depth of the compared rankings, there is an increased amount of data to be compared, so the range of possible values that similarity measure could take decreased. It can be concluded that increasing the amount of data to be compared means decreasing the similarity measure value.

- indefiniteness

is called *similarity measure on indefinite rankings* or an *indefinite rank similarity measure*.

Figure 2.8 illustrates the correlation between the length of the prefix and the corresponding range in which the similarity measure of the two rankings takes the value. When no parts of the rankings are compared, meaning that the depths of the comparison is zero, the similarity measure can take any value in the range between zero and one. As the depth of the comparison is increased, the range for possible values of similarity measure is monotonically decreasing.

2.3.1 Rank-biased overlap (RBO)

The authors of the scientific paper [13] show a comprehensive list of the existing similarity measures and from that literature overview it is clear that none of them meet all the requirements mentioned and explained in the previous section. So, the authors propose a novel similarity measure, namely *rank-biased overlap*.

The rank-biased overlap is overlap-analysis based method and it is very similar to the calculation of a mean overlap. The difference, when compared with a simple calculation of the average overlap, is that the obtained weights of the ever increasing depth are decreasing. This this way it is ensured that the importance of the rank prefix made of the finite number of elements can not be overpowered by the influence of the tail of the rankings that is made of infinite number of elements. So, it is important to notice that the series of the weights are:

- decreasingly convergent
- proportional to each other
- adding up to a sum that is bound (value between 0 and 1)

The similarity of the indefinite rankings measures the similarity over the prefix of the rankings and provides the user with lower and upper bounds (range), thus nearly describing the value the similarity measure can take when calculated over the full ranking. Naturally, when it comes to indefinite rankings, if single value similarity measure is wanted, the full evaluation of the indefinite ranking is needed. As it is *indefinite*, only an estimate with a reasonable precision can be achieved. Another important fact to notice, before going into more detail of this similarity measure, is the fact that the rank-biased overlap is a similarity, not a distance measure, so it is not metric. But the good thing is that the $1 - RBO$ is metric.

RBO on indefinite lists

For the sake of better understanding and more efficient communication, it is important to provide a mathematical representation of the statements in 2.3.1. The notation will be the same as in [13]. The necessary notations are:

- S, T - two indefinite rankings
- S_i, T_i - i-th elements of the ranked lists S and T
- $S_{c:d}$ - a set of elements from the indefinite list S beginning from index c up to the index d
- equality between $S_{1:c}$ and $S_{:c}$
- equality between $S_{d:\infty}$ and $S_{d:}$

The *Intersection* of the lists S and T at the depth d is denoted as:

$$I_{S,T,d} = S_{:d} \cap T_{:d} \tag{2.6}$$

The *Overlap* of the two arbitrary lists S and T is defined as a length of the intersection of the lists:

$$X_{S,T,d} = |I_{S,T,d}| \tag{2.7}$$

Agreement is defined as following:

$$A_{S,T,d} = \frac{X_{S,T,d}}{d}, \tag{2.8}$$

and one can understand it as a proportion of the overlap of the lists S and T and the depth at which the overlap is achieved.

Rank-biased overlap is, in some aspects, very similar to the average overlap as it was stated before. The average overlap can be defined as:

$$AO(S,T,k) = \frac{1}{k} \sum_{d=1}^{k} A_d, \tag{2.9}$$

and k is the maximal depth up to which the overlaps are calculated.

When two lists are given, the most natural and instinctive approach to similarity calculation is to take the top k elements from the lists and calculate the similarity as the proportion of the overlap of the lists and ranking length. The mathematical formulation is $|S \cap T|/k$. Although this method fulfils the requirement to be non-conjoint, the problem is, that this method is not *top-weighted* Now, this simple idea can be extended in a more advanced way. Instead of the simple overlap calculation for maximal depth k (meaning the overlap of the entire lists), one could calculate the cumulative overlap over the ranked lists for ever-increasing depth k. This exact approach is the one formulated and represented by equation 2.9. The disadvantage of this approach is the fact that this method does not consider the weights of the elements of the ranked lists at any way.

This simple but yet not satisfyingly good similarity measure is *top-weighted* because of its cumulative nature. Nevertheless, figure 2.9 shows an example how *average overlap* measure is calculated.

The way it is possible to make this measure even better is to somehow include the weights into calculations. The authors of the paper [13] propose the following:

$$SIM(S,T,w) = \sum_{d=1}^{\infty} w_d \cdot A_d, \tag{2.10}$$

d	$S_{:d}$	$T_{:d}$	$A_{S,T,d}$	$\mathrm{AO}(S,T,d)$
1	<a>	<z>	0.000	0.000
2	<ab>	<zc>	0.000	0.000
3	<abc>	<zca>	0.667	0.222
4	<abcd>	<zcav>	0.500	0.292
5	<abcde>	<zcavw>	0.400	0.313
6	<abcdef>	<zcavwx>	0.333	0.317
7	<abcdefg>	<zcavwxy>	0.286	0.312
n	<abcdefg...>	<zcavwxy...>	?	?

Figure 2.9: Example for an average overlap calculated over the increasing depth

where the w represents the vector of weights and w_d is, simply, the value of weight vector used for the calculation of overlap at the depth d. The vector w is deliberately chosen so that it is convergent and, as such, each overlap calculation from the series has a fixed contribution to the sum of overlaps for all depths. For this exact purpose the authors propose the usage of geometric progression where the following is true:

$$\sum_{d=1}^{\infty} p^{d-1} = \frac{1}{1-p} \tag{2.11}$$

In this way, an infinitely long sum can be calculated for p on interval $0 \leq p < 1$. If w_d is set to $(1-p) \cdot p^{d-1}$, implying that $\sum_d w_d = 1$, it develops a formulation for *rank-biased overlap*:

$$\boxed{RBO(S,T,p) = (1-p) \sum_{d=1}^{\infty} p^{d-1} \cdot A_d} \tag{2.12}$$

The parameter p has the ability to influence the velocity of weight convergence. The smaller value of p implies a much steeper convergence of weight, thus making the measure more top-weighted and the other way around; the higher p value is a flatter decline of the weights is and the measure becomes less top-weighted. When the p takes the extreme values in its own range, the following is true:

- when p is 0 $\rightarrow$ only the top-weighted element of the indefinite rank is analysed and the similarity is either zero or one
- when p is arbitrarily close to 1 $\rightarrow$ the weights are almost the same for all depths and the analysis is arbitrarily deep.

The similarity measure always takes the value somewhere within the range between zero and one. When the similarity measure takes the value 0 it means that the lists are completely different, in other words, *disjoint*. When it takes value 1, the lists are completely the same.

This similarity measure ranked-biased overlap can be seen as probabilistic similarity measure. For parameter p it is said in [13, 15] that it models the user's *persistence*. When the two infinite lists are compared, the process of comparison starts with the user looking at the first, top-weighted elements of the list, and the analysis depth is 1. The probability that the analysis will be done for the next depth is represented by parameter p. The probability that the analysis will be stopped at the current depth is subsequently represented by $1-p$.

Once the rank-biased overlap similarity measure is precisely and mathematically defined, there is only one more issue left open. What happens if there are tied ranks in an ranked list? Saying that some ranks are tied should describe the situation in which two or more list elements have the same rank and they take n consecutive positions in ranking but they are all equally eligible for the same rank in the list. The authors of the paper [13] state that this issue can be treated by modification of the equation 2.8 in the following manner:

$$\boxed{A_{S,T,d} = \frac{2 \cdot X_{S,T,d}}{|S_{:d}| + |T_{:d}|}} \tag{2.13}$$

Rank-biased overlap method deals with this situations by dividing the overlap of the lists S and T at the depth d multiplied by 2, by the amount of list elements present at the depth d instead of dividing by the depth alone as in 2.8. When there are no elements of the same rank value in list, there are no ties in list for depth d, the equations 2.8 and 2.13 are equivalent. But when this is not the case and there are some ties in a list for arbitrary depth d, the equation 2.13 can take a value greater than one.

The equations 2.12 and 2.13 represent the cornerstone of the software implemented as a part of this thesis.

2.4 Semantic stability

Once the author of this master's thesis was able to measure the similarity of two ranked lists, the most challenging task in this thesis was already done. For a given series of the revisions of the single Wikipedia article, it was possible to calculate the similarity of each consequent pair of revisions. The output of this process is a vector containing the similarities of revisions of the single article where all acquired similarity values are sorted by the time-stamp marking the time when the revisions were created.

The main goal of this thesis was exactly to visualise the process of semantic stabilisation of several Wikipedia editions in different languages. The paper [1] proposes a novel method

which is able to calculate the semantic stabilisation. Hence, this method is based on rank-biased overlap described in detail in 2.3.

The goal of this method is to calculate how many of the predefined resources have stabilized in a certain period of time. The resources used in paper [1] are social media tags (twitter, delicious, librarything) but the concept and idea are totally applicable on Wikipedia data set with absolutely no modifications. Although it calculates the percentage of the stabilized resources from the data set consisting of n resources, it is based on the stabilization process of a single resource (in the case of this thesis, a single resource is a single Wikipedia article with all its revisions). In addition to this, the method described allows the user to evaluate the exact number of the stable resources (Wikipedia articles) over a longer period of time. By making this possible, the comparison of the stabilisation process for different Wikipedia editions could provide some useful data. It is possible to observe if and how the size of Wikipedia edition plays the role in the process of stabilisation, for example.

This method first defines three parameters:

- n - the number of resources
- k - the threshold value for rank-biased overlap
- t - the number of consecutive tag assignments after which the stabilisation calculation is done (in this paper it simply represents the time interval for stabilisation calculation).

An article in Wikipedia corpus, whose two consecutive revisions (before and after the time-point t) have the rank-biased overlap similarity measure equal or greater than the threshold k, is said to be a *stable article* at point t. The method allows the percentage of the semantically stabilized articles from the Wikipedia corpus to be calculated.

A rather simple mathematical formulation of this method for inspection of stabilization process in a given data set is as following:

$$f(t,k) = \frac{1}{n}\sum_{d=1}^{n}\begin{cases}1, & \text{if } RBO(\sigma_{t-1},\sigma_t,p) \geq k\\ 0 & \text{if } otherise\end{cases} \tag{2.14}$$

This formula expresses the following idea: for each article in a Wikipedia corpus, the rank-biased overlap similarity measure is calculated. Inputs are the revisions before and after the time point t as well as the parameter p. If the calculated similarity is equal or greater than the threshold k, add 1 to the sum, otherwise add 0. With no more articles in corpus to iterate, divide the sum by the total number of iterated articles from the Wikipedia corpus. Thus, the result will be the percentage of the stable articles at time-point t for predefined threshold value k.

The figure 2.10 is taken from [1] as an illustration of the idea described in the previous paragraph. The author of this thesis found this diagram not to be the best way to present

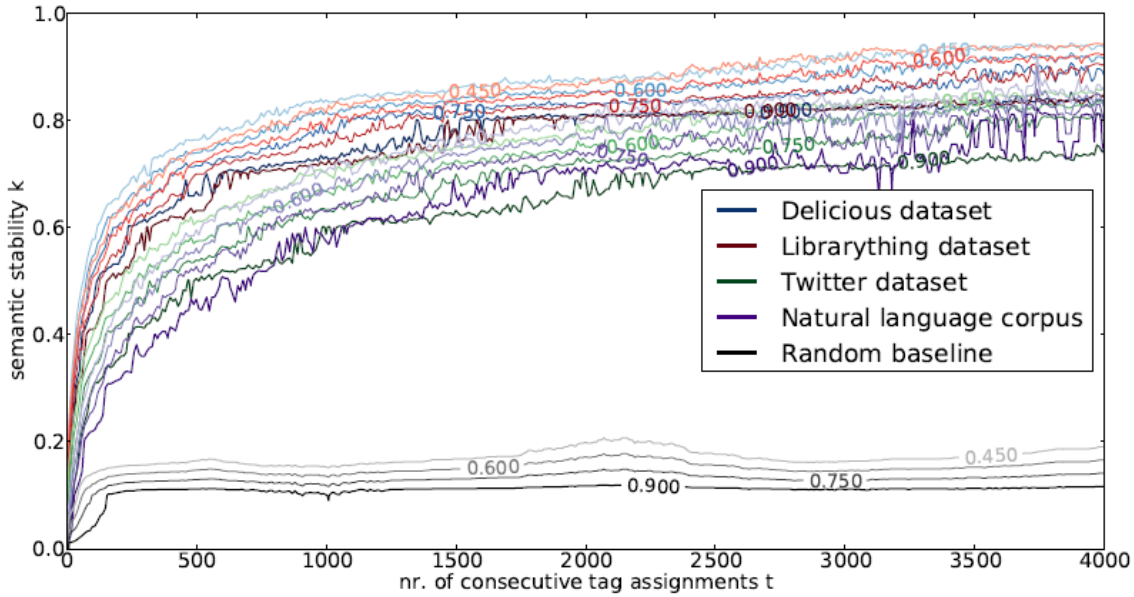


Figure 2.10: Semantic stabilization of different social tagging datasets, a natural language corpus and a synthetic random tagging dataset as a control. The x axis represents the consecutive tag assignments t while the y-axis depicts the RBO (with p = 0.9) threshold k. The contour lines illustrate the curve for which the function f(t, k) has constant values. These values are depicted in the lines and represent the percentage of stabilization f. One can see that tagging streams in Delicious and LibraryThing stabilize faster and reach higher levels of semantic stability than other datasets [1].

this kind of data as it is not easy enough to interpret the data shown in this figure. In the experimental part of this thesis, some more readable diagrams providing the same information about the stabilization process of articles will be included. Meanwhile, figure 2.10 can be used to show the process of stabilization of different data sources (some social media tagging streams) for different values of parameters p and k.

2.5 Dataset

One of the main challenges during the course of this master thesis was to implement the rank-biased overlap similarity measure as well as the semantic stability measure as explained in sections 2.3 and 2.4 respectively and to apply the implemented methods on the Wikipedia data set. This section will provide more insight in the Wikipedia data set and it will also briefly explain the concept of *Big Data*.

2.5.1 Big Data

The Wikipedia data set is available in many languages but what is common for all editions is that the articles consist mainly of text enriched with pictures. The number of documents available in each Wikipedia language edition is huge so the manual collection and analysis of those articles, when used as a data source, is not possible. This can be better described as a term *Big Data Concept*. As already mentioned, the Big Data describes all the data sources that are so huge that the manual processing and analysis of the data from such data sources is not efficient nor possible. Some real-world examples for *Big Data* are:

- consumer product retail companies are monitoring and analysing the data produced by user behaviour on social media networks like Facebook and Twitter in order to get a better insight into people's opinion about a group of products of interest
- manufacturers are monitoring the built equipment in order to recognize the signs of material wear. When the parts are damaged and not replaced in time it could lead to a loss of human lives (rail roads) and if replaced too early it could lead to an increase of costs. The sensors in such equipment produce a huge amount of useful data.
- in a similar manner, the producers are overseeing the social networks, but with a different goal as marketeers. Producers are looking for technical issues being reported to prevent warranty problems for their products.
- government makes data public at a national level for users to develop the applications for public service and public good and so on.

Analysing the content of the Wikipedia language editions in order to get an overview of the semantic stabilisation process of the source written in some natural language is also an activity from the field of *Big Data*.

The authors of the paper [4] define the big data through tree important aspects:

- volume
- velocity
- variety.

Furthermore, they present two definitions of *Big Data* derived from different sources:

> *"Big data is high-volume, high-velocity and high-variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making"*

> *"Big data is a term that describes large volumes of high velocity, complex and variable data that require advanced techniques and technologies to enable the capture, storage, distribution, management, and analysis of the information."*

Volume simply defines the magnitude (amount) of the data available. When big data concept is in questions, the amounts of data used are in magnitude of terabytes or petabytes. One terabyte of data can be stored on about 220 DVDs and that is enough storage space to store about 16 million Facebook photos. One petabyte has 1024 terabytes. Facebook processes about 1 million photos per second so now it is easier to get the right picture about the immense amount of data to be processed.

Variety describes the structural property of data. With such a huge amount of data, it is not that hard to imagine how much of the available data is heterogeneous. With the wide palette of available tools, the users are in a position to use different types of structured, semi-structured and unstructured data. Figure 2.11 shows data classification:

- structured data: data found in the tables of relational databases or excel spreadsheets. This type of data is the rarest and only 5 percent of the used data has the properties of well structured data.
- unstructured data: text, pictures, video content, sensor data, social network data, are just some of the examples for unstructured data. The term of *unstructured data* describes all the data which can not be simply used by machines for automatic evaluation or analysis.
- semi-structured data: this type of data belongs right in the middle of the spectrum between the structured and unstructured data. A good example for the semi-structured data are XML files. XML is a textual language used to exchange the data on the Internet. When the user manually defines data tags in an XML file, this file becomes machine readable, hence the name semi-structured data.

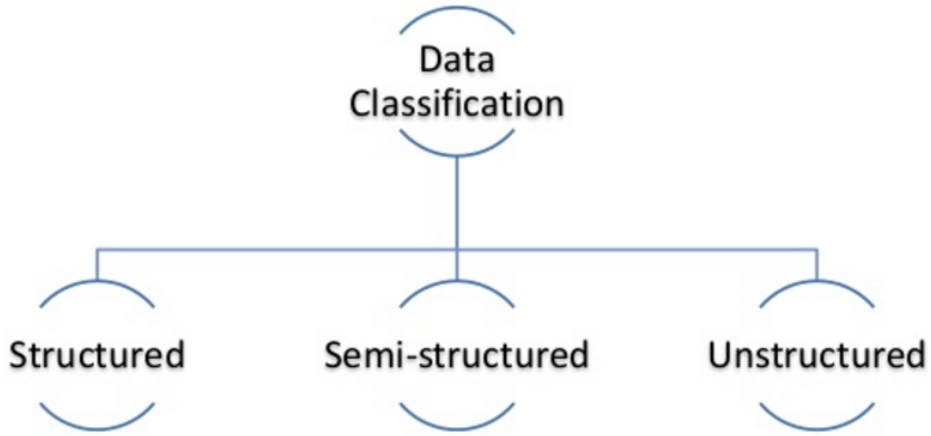


Figure 2.11: A classification of data.

Velocity describes the rate of change in the data being generated as well ass the reaction time for acting upon changes and updates of the analysed data. Today, when the mobile communication devices are wide spread, the rate at which the date is generated is ever increasing. It is the cause of the huge demand for real-time analytic tools. Just some of the generators of such high frequency data are, for example, conventional retailers (Wal-Mart, Carrefour, Metro and so on process more than one million transactions per hour), the real-time date produced by cell phones, sensor data and so on. The example of conventional retailers is particularly convenient to showcase how big data concept can be used to extract the very useful data about customer behaviour, their interests and past buying patterns, and than to use that data for analysis and creation of the real customer value. All these three different

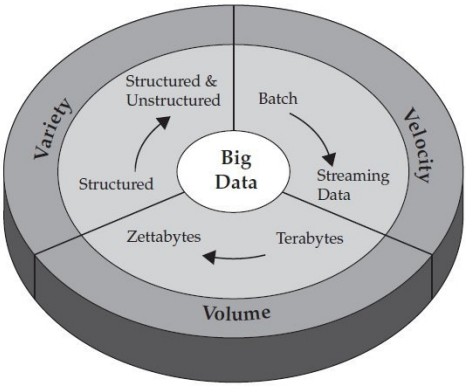


Figure 2.12: Concept of Big Data[2]

aspects of the big data concept are shown in figure 2.12.

Big data concept became really important and popular in recent years and as the evidence for this statement figure 2.13 is shown. It has to be noticed that the number of scientific publications on the topic of Big Data exponentially increased in the last five years. From all three main aspects of the big data concept mentioned in this section, the *volume* was the most important one during the course of this thesis. The Wikipedia data set will be explained in the next section.

2.5.2 Wikipedia data set

Wikipedia is one of the greatest, freely accessible web-based encyclopaedias and its content is open for editing by users. The articles in this on-line encyclopedia contain links to other articles and are designed to be navigated by users in order to visit other thematically relevant articles. The articles in Wikipedia are mainly a contribution of volunteer authors doing this

[2] The figure can be found on `https://apandre.wordpress.com/2013/11/19/datawatch/`, 07.01.2016

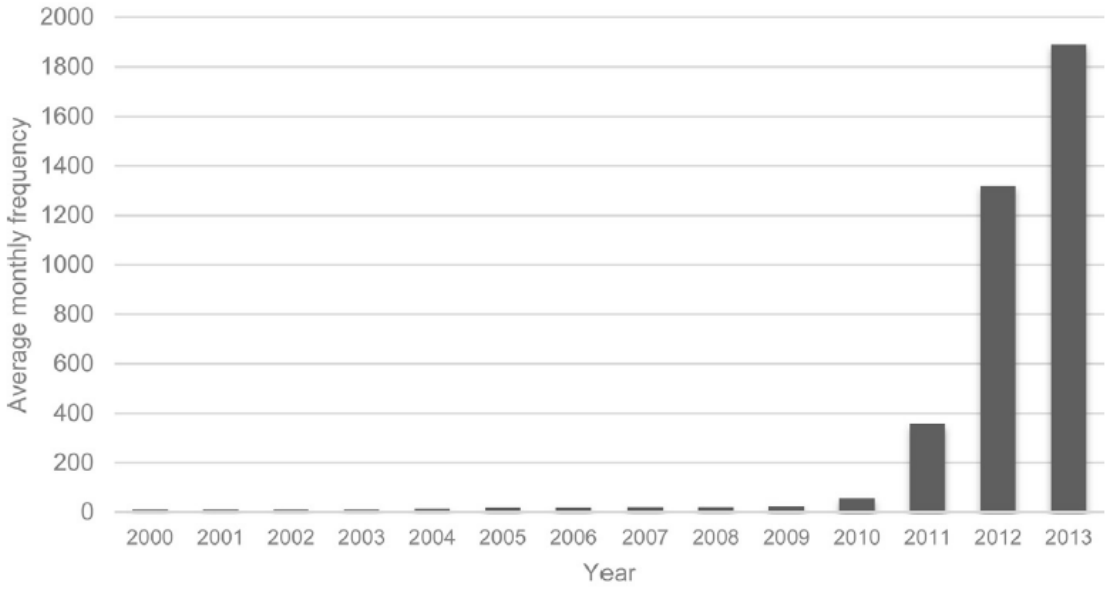


Figure 2.13: Frequency distribution of documents containing the term "big data" in research library ProRequest [4].

community service for free. Every willing user, having an Internet connection, can write new or edit the existing content on Wikipedia. Only in some special cases is the right to freely edit the content restricted to the user. These special cases are introduced mainly to discourage vandalism. A user contributing to Wikipedia project can do it anonymously or by creating and using a personal account.

The Wikipedia project has been launched in year 2001 and it gained a huge popularity almost instantly by attracting a significant number of visitors on a daily basis thus growing and becoming the reference for other similar projects. The number of visitors in a single day in June 2015 was stunning 374 million. There are currently about 40 thousand active users - contributors editing Wikipedia contents of almost 38 million articles written in over 290 world spoken languages. Among similar projects, Wikipedia project is second to none by the sheer size.

All these informations are implying that Wikipedia can be seen as an extremely dynamic source of unstructured data written in different natural languages [12]. There are two ways to get the content of Wikipedia in any language:

- by using Wikipedia API
- by downloading the XML dump files

Since the usage of API is not meant to be used as a way to download all articles of any Wikipedia language edition, the second aforementioned way of doing it has to be used.

Although the provided API has good functionality, the time needed to download the uncompressed data over the Internet, where a separate request has to be sent to the server for each revision of the article, is simply to long and utterly impractical. On the other hand, downloading the XML dump files for a wanted Wikipedia language edition is pretty straightforward. The Wikimedia[3]provides XML dump of all active Wikipedia projects. Some of them have complete content compressed in a single, compressed XML dump file while some other, larger Wikipedia editions have a compressed archive of several dozens of files. Also, there are several types of data dumps available:

- dumps containing articles of current versions only
- dumps containing all pages with complete edit history (both as 7z and bz2 archives)
- dumps containing extracted abstracts for Yahoo only

For the purposes of this master's thesis, dumps containing all articles with their complete edit history had to be obtained. The basic building block of all Wikipedia editions is a page. Every page represents an article and every article has at least one, but usually more than one revision. There are articles in bigger Wikipedia editions which have tens of thousands of revisions. Every single revision has:

- a timestamp
- an ID
- a name
- an author
- categories
- textual content

among other fields that are not of crucial importance for this paper. In order to be able to analyse the provided articles from the dump file, one has to previously clean up all the Wikipedia mark-up symbols from the plain text. The following example[4]shows the XML for a sample article:

```
<mediawiki xml:lang="en">
  <page>
    <title>Page title</title>
    <restrictions>edit=sysop:move=sysop</restrictions>
    <revision>
      <timestamp>2001-01-15T13:15:00Z</timestamp>
      <contributor><username>Foobar</username></contributor>
```

[3] https://dumps.wikimedia.org/, 07.01.2016
[4] https://en.wikipedia.org/wiki/Help:Export, 07.01.2016

```
      <comment>I have just one thing to say!</comment>
      <text>A bunch of [[text]] here.</text>
      <minor />
    </revision>
</mediawiki>
```

Listing 2.1: An example of a page in XML dump

For the purposes of this thesis, the idea was to analyse 10 Wikipedia language editions, five of which are randomly selected small language editions and the remaining five are the largest language editions, as stated in 1.2. Downloading complete XML dumps of the small Wikipedia editions was not a problem since those have no more than 2 GB of data. On the other hand, obtaining the XML dump for English or German editions of Wikipedia and fully analysing them was a greater challenge. For example, English Wikipedia is about 1 TB large, and it is the size of a compressed file. Since the goal of this thesis was not to analyse the full Wikipedia corpus anyway, the sampled data was used for 8 out of 10 Wikipedia editions analysed. Only Czech and Finnish Wikipedia corpus was fully analysed. For all other editions the sample of 10 thousand randomly selected articles with their complete revision history was used.

3 Data preprocessing and dataset

3.1 Software architecture

The main goal of this thesis, as stated in 1.2, was to analyse the semantic stability of Wikipedia. To that end, we developed a software solution able to analyse Wikipedia XML dump and to provide data showing the semantic stabilization process. Since the input data is huge, this software solution had to be implemented in such way, that the results describing the semantic stabilisation process could be acquired at least for a small Wikipedia editions in some reasonable time.

In this chapter, the software architecture will be presented as well as the necessary software libraries used to accomplish the goals of this thesis. Although there was no requirement to use a specific programming language, the author of this thesis decided to use *Java*. This decision is based on the fact that there are many existing enterprise solutions and free software libraries in the field of natural language analysis that are well positioned on the market today and what is even more important, those software solutions are fully documented.

Figure 3.1 shows the structural overview of the proposed software solution. The proposed natural language processing tool (NLP) can be seen as a composite system of five independent modules where modules are interacting in linear manner exclusively in one hedirection over well defined interfaces (figure 3.1, from the bottom towards the top). The idea behind this design is to make it possible to enable or disable certain modules so that the execution time of the proposed program could be shortened. As some of those five modules takes significantly more time to complete their task compared to other modules, it was preferable to have the possibility to execute them only once. Thus the time needed for their execution could be spared in the next run. One typical example for this is the situation where a given Wikipedia XML dump file is indexed in the first run and semantic stability analysis has to be done for different values parametrizing the semantic similarity algorithm. Input data, thus the index data too, stays the same so there is no need to wait one additional week to index the Wikipedia corpus again. It is enough to disable the indexing module in *configuration file* and to run the program with different algorithm settings. The five modules mentioned are:

1. module for parsing the Wikipedia pages from XML dump
2. module for linguistic processing and indexing

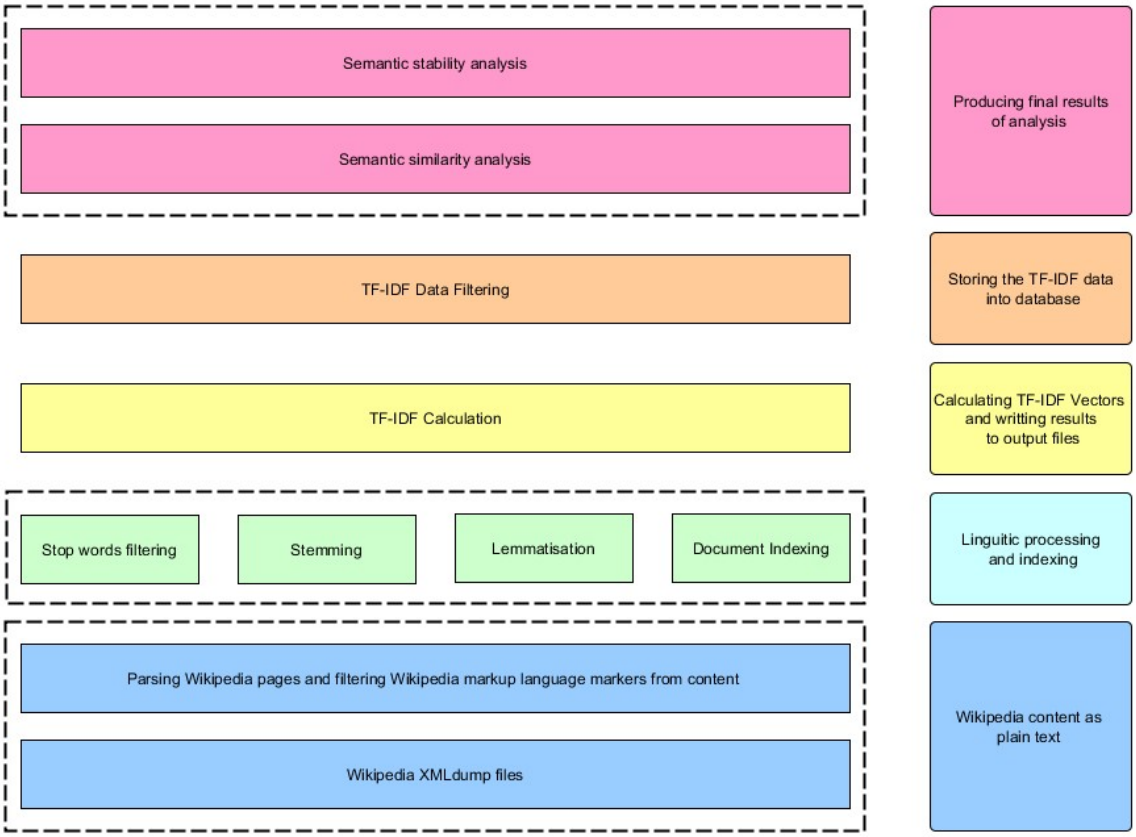


Figure 3.1: The architecture of the proposed software solution.

3. module for TF-IDF calculation
4. module for storing the filtered TF-IDF data into the database
5. module for calculating the semantic similarity and stability.

Each of the aforementioned modules will be presented in the following sections. At this point figure 3.2 is shown for a better understanding of the implemented modules and their roles in the proposed software tool.

All the used third party software tools listed in figure 3.2 are well known, tested and documented.

3.2 Development environment

The proposed software is designed to work on both Windows and Linux but it was developed, tested and used in Windows 7 environment. The main reason for this is the more simple

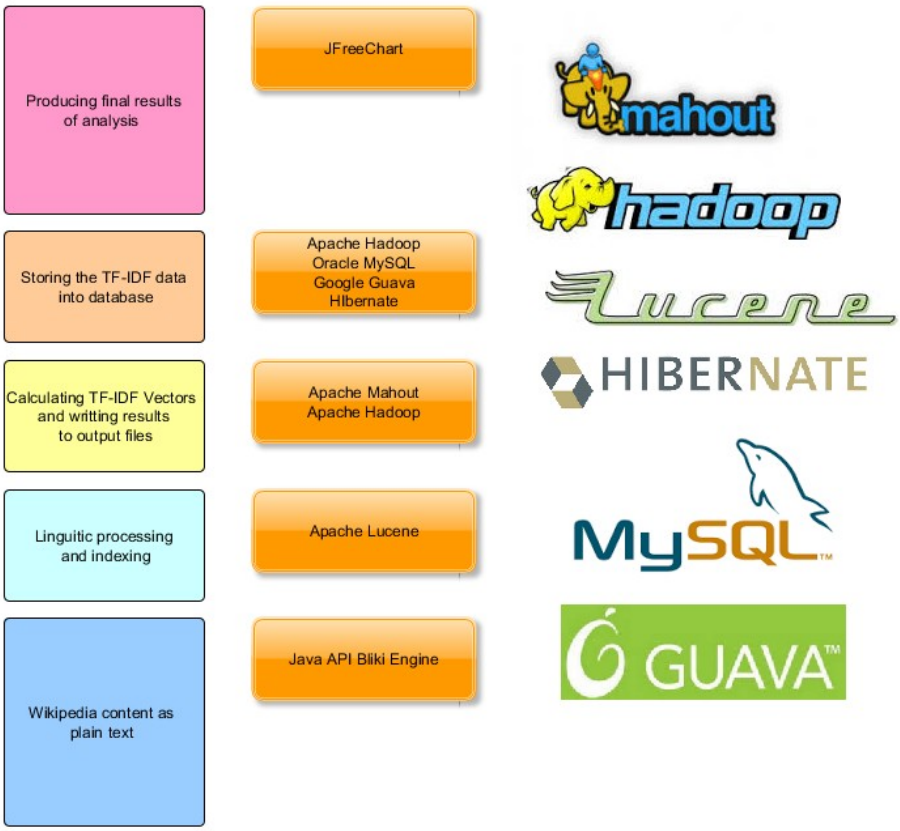


Figure 3.2: The used third party software tools per module.

installation of the needed software. Another important reason is the possible lack of driver support for the used SSD storage drives necessary to maximize the utilization of the hardware resources used to run the proposed software. The available hardware includes:

- AMD Phenom 2 - 4 x 3.2 GHz
- 8 GB RAM
- SSD used as a system drive
- 1 x Samsung 850 Pro SSD used as drive storing the XML dumps

The installed software:

- Java Development Kit 8 update 25
- Cygwin (needed in order to run Hadoop®)

Eclipse Moon was used as IDE (Integrated Development Environment) but it is up to the personal preference. The VisualVM[1] was used as a profiling tool but there are many other

similar tools. This one is free for use and the author of this thesis already used it in many occasions in previous projects so it was a a personal preference again.

3.3 Parsing Wikipedia pages from XML dump

The designated task of the first module mentioned in section 3.1, is to get the textual content of the articles and all its revisions from the Wikipedia XML dump and to filter out all the unnecessary data from it, thus extracting the plain text only. The removal of unnecessary data is meant to identify and remove the Wikipedia markup language symbols contained within the article text. Namely, Wikipedia has its own way of text formatting and in order to format the text of an article, one has to use Wikipedia markup language symbols[2]. The node structure of the XML dump is described in section 2.5.2. Only as a reminder, XML dump has one main node <*mediawiki*> and this node contains all the other <*page*> nodes. So, the naive solution and the first one the author of this paper tried to implement was to simply read the complete XML dump file with the Java DOM parser. The problem with this approach is that the dump file is simply too large to store its complete content into RAM at once, what DOM Parser does. So, this approach was discarded right at the beginning. The second, much better solution was the idea to use JAVA SAX Parser implementation. Unlike the DOM Parser, SAX Parser operates on a portion of the XML only. It is an event-driven parser that does not need to read the whole XML document at once. Because of that, it is much faster than its main rival. Although this is a huge benefit compared to what DOM offers, SAX has one big disadvantage. Any kind of XML validation requires a whole XML document, so the validation of the XML with SAX is not possible. Luckily, for the purposes of this thesis, it is to presume that the obtained Wikipedia XML dumps are valid anyway. Figure 3.3 is the UML diagram of the implemented software module. In this diagram, one interface and one class are labelled *Bliki Engin* are marked. They belong to the Bliki Engine library used to implement the needed functionality and will be discussed in the next paragraph.

The third party *Bliki Engine* library (bliki-core-3.0.19.jar) is an implementation of the SAX parser and it offers some other useful features for working with Wikipedia XML dump file. This library exposes the mentioned interface *IArticleFilter*. The class implementing this interface has to define the method *process(WikiArticle page, Siteinfo info)*. The input parameter *page* is an instance of the class *WikiArticles* that is an abstraction of a real Wikipedia article. This object contains all the data about an article and its revisions. One of the available methods is the method *getText()* which returns the Wikipedia article in its raw representation, thus with Wikipedia markup language markers. The second class provided by Bliki Engine library is *WikiPatternMatcher* and this class provides predefined *regular expressions (RegEx)*

[1] `https://visualvm.java.net/`, 07.01.2016

[2] Official Wikipedia page describing the Wikipedia Markup Language: `https://en.wikipedia.org/wiki/Help:Wiki_markup`, 07.01.2016

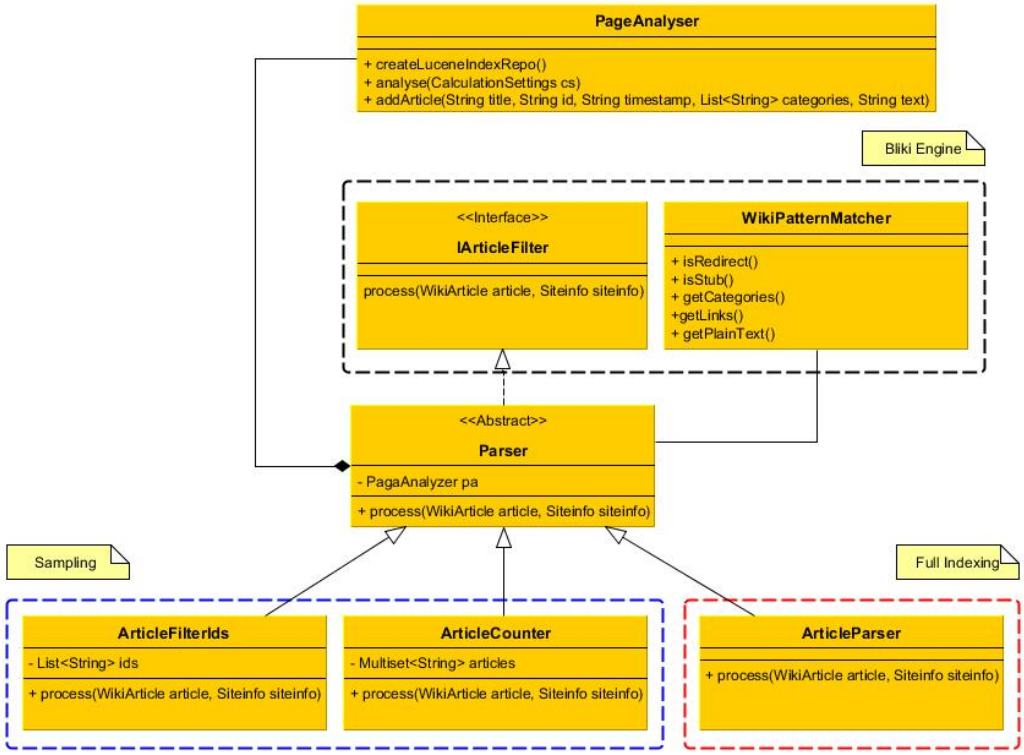


Figure 3.3: UML diagram of the implemented module responsible for the Wikipedia XML dump parsing and sampling with the *Bliki Engine* third party library.

designed to remove the Wikipedia markup language markers from the article text as well as to parse other important article properties such as ID, name, categories and timestamp.

It is mentioned in section 2.5.2 that small Wikipedia editions are exported as single XML dump file compressed to an archive file. This is the case for most of the small editions, but the bigger ones have the XML dump split into several independent archive files. The English edition, for example, has almost 100 archive files representing the XML dump. So, in case that a multi-core CPU is available on the machine running this piece of software, the ideal solution would be able to process more than one XML file in parallel. For a machine with 4 cores, the ideal number of XML files to read in parallel is 4. If that is not the case, only one core would be fully utilized and other 3 cores would stay idle. The parallel I/O operations on HDD are not recommended as those exact tasks are usually the bottleneck of the overall performance of the system. Because of this fact, an SSD is used as the drive for the input XML data storage and the parallel I/O brings the substantial overall performance boost in the proposed module.

The parallel execution of the module presented in this section is implemented with the Java Fork/Join Framework (FJF). This framework is very important for this project as it was

used for CPU utilization in other modules too, so it will be explained in more detail in a separate section 3.4.

This module can operate in two modes, where the mode to be used can be set in the tool configuration file. The two modes are:

1. full indexing mode
2. sampling mode.

The first mode was already explained in this section. All available articles from the XML dump will be iterated and processed. The second mode represents another way of implementing the exposed *IArticleFilter* interface. The purpose of this mode is not to process every single article of the Wikipedia dump, but to be selective (random selection) and to process only a portion of the available articles, thus processing only the sample data. The idea is to obtain the IDs of all articles available in XML dump (which is done in class *ArticleCounter*) thus creating a list of all articles. Then, the sample of N articles from the list is randomly selected and the second iteration is started. The second pass is done in class *ArticleFilterIds*. The ID of the current article selected in this second iteration is checked. If it is in the list of random articles and if it is positive, then the article is processed. The size of the sample N can be set by the user in the configuration file. The two classes mentioned in this paragraph are grouped together in figure 3.3 and marked with the label *Sampling* as they represent a single functional unit.

3.4 Fork/Join Framework

Fork/Join Framework implements the idea known in computer science as *divide-and-conquer*. It provides a very convenient way for using recursions in order to solve one big problem by dividing it into many smaller problems and solving them one by one in concurrent manner. This is the main idea of the divide-and-conquer strategy. This framework is not meant to be the replacement for the existing multi-threading possibilities of Java. It is introduced to the Java world in Java version 7 and it was designated to utilize the usage of recursive function calls. When this framework is used one does not have to worry about thread synchronisation, starvation of threads and other problems typical for the field of parallel computing. On the other hand it is applicable only on a problem that can be dissolved to a number of smaller, partial problems. When this is the case one has to initialize the *pool of worker-threads*, extend the Java class *RecursiveAction* and to override its public *compute()* method. Once this newly extended class is instantiated, a pool of workers should be invoked by the method *invoke(RecursiveAction ra)* where the input parameter of the method is instance of the class extending the *RecursiveAction* class. This approach can be applied when the result of the

[3] The figure used as the example is downloaded from site: `http://howtodoinjava.com/2014/05/27/forkjoin-framework-tutorial-forkjoinpool-example/`, 07.01.2016

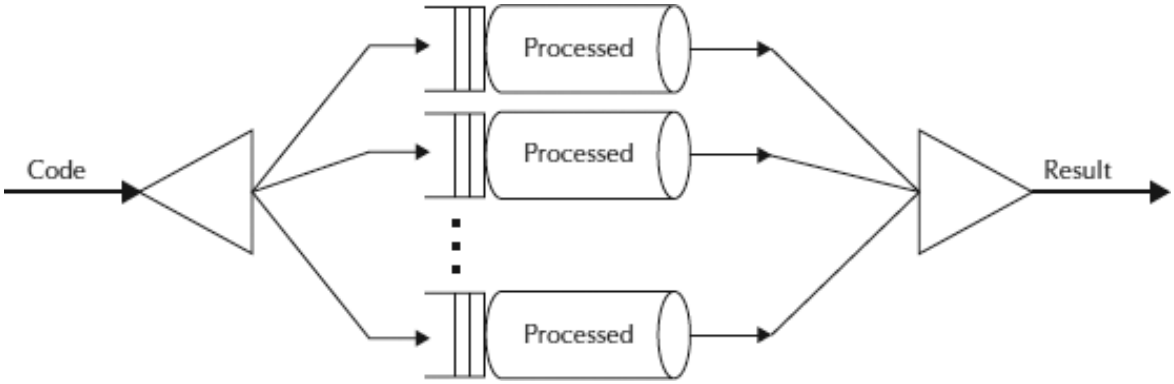


Figure 3.4: The *Fork/Join Framework*[3]

partial problem is an actual action and not a value. When the return values of partial problems are collections, instances of classes or other non-primitive or primitive Java types, the generic *RecursiveTask<T>* has to be used. The main principle stays the same [5, p. 171-199]. The afore explained functionality is depicted in figure 3.4.

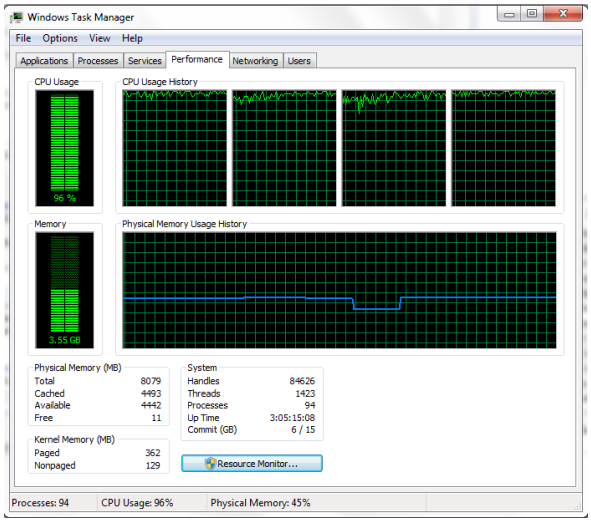


Figure 3.5: Utilization of the CPU when the semantic stability analysis software started, exploiting the benefits of multi-threading technology.

The Fork/Join framework is used to utilize multi-threading in the first module of the proposed software. The implementation details of that module are explained in section 3.3 so now, FJF can be explained using a practical example. The list of all input Wikipedia XML dump files is created and passed to the instance of the class extended from the Java class *RecursiveAction*. In the *compute()* method of the same instance, the list of files is divided in half and two instances of the *RecursiveAction* class are newly created, where both of them get the left and the right half of the XML files list, respectively. This is recursively repeated

on the list of XML dump files. Recursion partitions the list up to the point where there is a list with only one XML file for every worker-thread. This way, the main problem (read all files at once) is divided into several smaller problems (read only one file). For example, if there are 100 XML files and 4 worker-threads, 4 XML files will be read in parallel and other partial problems are waiting in a worker pool queue for the first free worker-thread. Figure 3.5 is a screen-shot of Windows 7 task manager showing the CPU utilization while the proposed software was running. All 4 CPU cores are fully loaded.

3.5 Indexing Wikipedia with Apache Lucene®

Section 3.3 describes how to acquire parsed, plain text revisions of all Wikipedia articles. The next module implemented is the module responsible for the *document indexing*. This module is the most time consuming of those mentioned in 3.1 but at the same time it is the least complex one. This is due to the fact that the document indexing functionality did not have to be personally developed. There are several solutions available and some of them are:

- Apache Lucene®
- Xapian (OpenSource)

For the implementation of the software for this thesis, Lucene® was used to index Wikipedia articles. The reason for this choice is the big community working on and with this tool, thus making the search for technical support less time consuming. Besides, the research showed that Lucene® and its sister project Apache SOLR are dominating the market of *full text search engines*.

Lucene® is presented as high-performance and high-scalability full text search engine. This is an open source project widespread in enterprise software solutions when huge amounts of plain text data have to be searched. The principles and ideas behind this full text search engine are the ones that can be found in every book from the field of *Information Retrieval*. In sections 2.1 and 2.2, the basic theory needed to accomplish the task required by this thesis is explained. This section will deal with the concrete challenges of natural language processing and indexing of Wikipedia articles. In order to index a plain text document with Lucene® library, the steps from figure 3.6 have to be carried out. Input of this second module is the plain text of the Wikipedia articles parsed in the previous module. For each plain text document parsed, a new instance of the Lucene® class *Document* is instantiated. The data to be indexed has to be described by the instances of the Lucene® class *Field*. The goal is to index the plain text data of an article (field Article) as well as its ID. So two field instances are needed. For every instance of the *Field* class several crucially important parameters have to be set. Figure 3.7 shows how it is done. The important thing to mention is that the *ID* does not have to be tokenized but it only has to be stored. The field *Article* has to be stored,

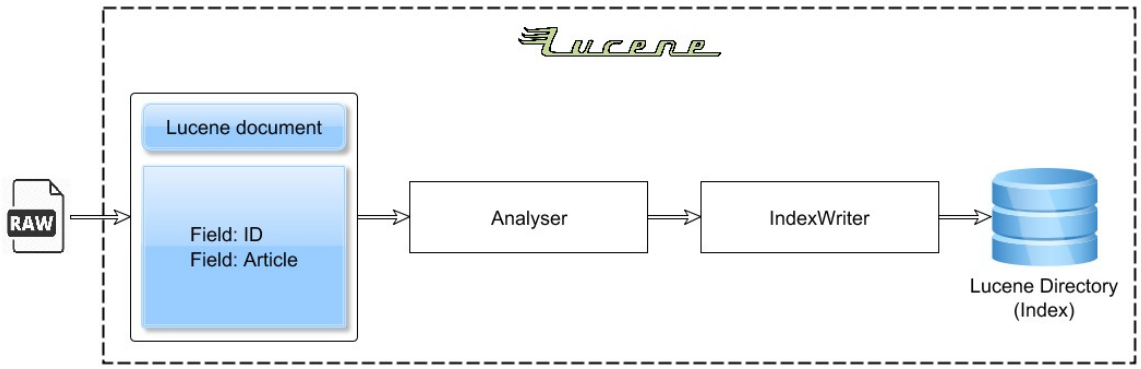


Figure 3.6: The steps needed to analyse and index plain text documents.

```
FieldType indexedType = new FieldType();
indexedType.setIndexed(true);
indexedType.setStored(true);
indexedType.setTokenized(true);
indexedType.setStoreTermVectors(true);

FieldType storedType = new FieldType();
storedType.setIndexed(true);
storedType.setStored(true);
storedType.setTokenized(false);

idField = new Field("Id", "", storedType);
contentField = new Field("Article", "", indexedType);

article = new Document();

article.add(idField);
article.add(contentField);
```

Figure 3.7: The code snippet responsible for the setup of the fields in Lucene® document instance used for the indexing of the plain text article.

indexed and tokenized and the term vector generated by Lucene® has to be stored too. Any other way would prevent the calculation of the TF-IDF vectors in the next, third module of the proposed software solution.

Before the indexing procedure is started, the instances of Lucene® classes *IndexWriter* and *Analyser* have to be created. The UML diagram depicted in figure 3.3 from the previous section shows the relation between the class responsible for the parsing of articles (the classes implementing the *IArticleFilter interface*) and the class *PageAnalyser*. The instance of this class encapsulates the management of the needed instances of mentioned Lucene® classes.

The class *IndexWriter* is responsible for the management and the writing of documents into the index directory and it is configured by the instance of the class *IndexWriterConfig*. This configuration of the index writer instance is nothing but the specification of the natural language analyser to be used. The Lucene® library already provides the analyser classes for many languages. For the languages that are not supported, it is possible to develop a separate analyser class of its own, but it is not a trivial task anyway. The choice of the analyser is up to the user because of the fact that it has to be specified in the configuration file of the proposed software by setting the language of the input Wikipedia XML dump.

Any analyser class in Lucene® is a composite of classes implementing tokenizer, stemmer and stop-word filter which are also provided by the Lucene® library. The functionality of all those components is shown in chapter 2.

One of the most important aspects of the Lucene® indexing mechanism is its ability to work in multi-threaded environment. As described in sections 3.3 and 3.4, multi-threaded reading of multiple XML dump files is possible. Reading the files in parallel implies the multi-threaded use of Lucene® *IndexWriter* when the multiple threads try to add documents to the index at the same time. This means that multiple threads providing the documents can share the single instance of *IndexWriter*. Multiple instances of the *IndexWriter* writing to the same index at the same time are not allowed. The great thing is that this class is thread-safe so multi-threading brings significant performance boost. Figure 3.8 shows how the input from multiple threads is handled by the thread-safe implementation of the *IndexWritter*. All the threads running in parallel provide the documents to be indexed to *IndexWriter*. But in this situation there are multiple instances of the class *DocumentWriterPerThread* (DWPT). Every worker thread activates and communicates with its own DWPT instance. Now, the *DocumentWriter* does not have to merge the document segment in memory and keep it there until the activation of the flush procedure. It can flush the documents brought by independent worker threads immediately to the Lucene® index directory. In the file system, the Lucene® folder contains only the Lucene® segment files. This folder is the input for the third module of the proposed software and will be explained in detail in the next section. The book [11, p. 28-62] provided the greatest insight in the topic of multi-threading and Lucene® in general. The authors of this book propose an even better way to achieve concurrency by providing a partial index directory for each thread running in parallel. In the end, when there are no more documents to be analysed, those partial indexes have to be merged. The problem with this approach is the need for more disk drives or more RAM and those resources were not unlimited for the author of this paper.

3.6 Generating TF-IDF matrix with Apache Mahout®

In previous section, section 3.5, it was precisely described how plain text documents from the Wikipedia corpus are indexed with Lucene®. As a result of this indexing procedure the

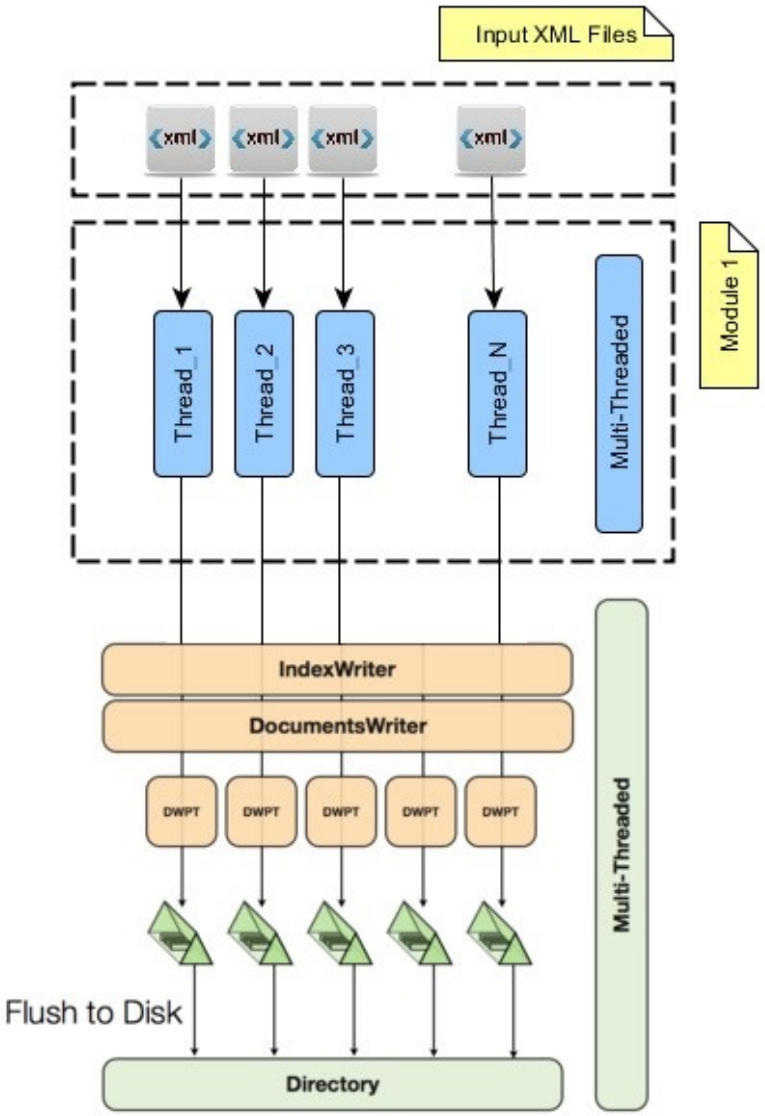


Figure 3.8: The functionality of Lucene® when used in multi-threading mode.

index directory is expected. This Lucene® index directory is represented in the file system of the storage drive as a single folder with multiple segment files. Every single segment file is a section of the index. The next goal is to generate a TF-IDF matrix out of the existing index. This matrix, as explained in section 2.2, consists of vectors where every vector represents a single article revision from the Wikipedia corpus. The values contained by the vector are the weights of the terms from the document represented by the vector.

In order to calculate this matrix, the Apache Mahout® library was used. This library is

built on top of Hadoop® and it is marketed as a scalable, machine-learning engine. It thrives on Hadoop® *MapReduce* paradigm to guarantee concurrency and scalability. The main tasks this library is designated for are clustering, data classification, collaborative filtering and dimensionality reduction of data. Furthermore, many different algorithms from the mentioned fields of study are available, some of which are:

- k-means clustering
- canopy clustering
- Naive Bayes/Complementary Naive Bayes classification
- Logistic Regression

and many more. One particularly important feature for this thesis provided by this library is the ability to generate sparse TF-IDF vectors from given plain text documents or from provided Lucene® index.

```
System.out.println("--> Calculationg TF-IDF vectors for every single article revision...");
// Create sequence files from Index
LuceneStorageConfiguration luceneStorageConf = new LuceneStorageConfiguration(
        conf, Arrays.asList(indexFilesPath), sequenceFilesPath,
        "Id", Arrays.asList("Article"));

SequenceFilesFromLuceneStorage sequenceFilefromLuceneStorage = new SequenceFilesFromLuceneStorage();
sequenceFilefromLuceneStorage.run(luceneStorageConf);
System.out.println("Sequence files generated");

// Generate Sparse vectors from sequence files
generateSparseVectors(sequenceFilesPath, sparseVectorsPath);
```

```
private static void generateSparseVectors(Path inputPath, Path outputPath) throws Exception {

    List<String> argList = new LinkedList<String>();
    argList.add("--input");
    argList.add(inputPath.toString());
    argList.add("--output");
    argList.add(outputPath.toString());
    argList.add("-wt");
    argList.add("tfidf");

    String[] args = argList.toArray(new String[argList.size()]);

    ToolRunner.run(new SparseVectorsFromSequenceFiles(), args);
}
```

Figure 3.9: The code snippet demonstrating how to use Mahout® in order to generate the TF-IDF matrix from Lucene® index

Mahout® provides two classes needed to implement the desired functionality. The classes are:

- *SequenceFilesFromLuceneStorage*
- *SparseVectorsFromSequenceFiles*

The usage of these two classes represents the two steps in the process of generating necessary TF-IDF matrix representing the Wikipedia corpus. The first step is to generate Hadoop® sequence files and the second step is to use generated sequence files to produce sparse TF-IDF vectors representing the documents. The first step is actually only preparation for the second step. Namely, Mahout® uses Lucene® index directory as input and transforms it into sequence files. This type of files is fully supported for concurrent I/O operations by Hadoop®. This is the main reason for index conversion. When index is transformed it is ready to be used as input for TF-IDF calculation. This exact functionality is provided by the class *SparseVectorsFromSequenceFiles* whereas *SequenceFilesFromLuceneStorage* transforms the index data into sequence files. The code snippet from figure 3.9 shows how these classes were used in the proposed software implementation. It shows how the class *SparseVectorsFromSequenceFiles* can be parametrized in order to acquire the wanted results. For the purpose of this paper only the simple TF-IDF matrix was needed so the most important parameter was the parameter defining the weighting schema (TF-IDF). Figure 3.10 shows

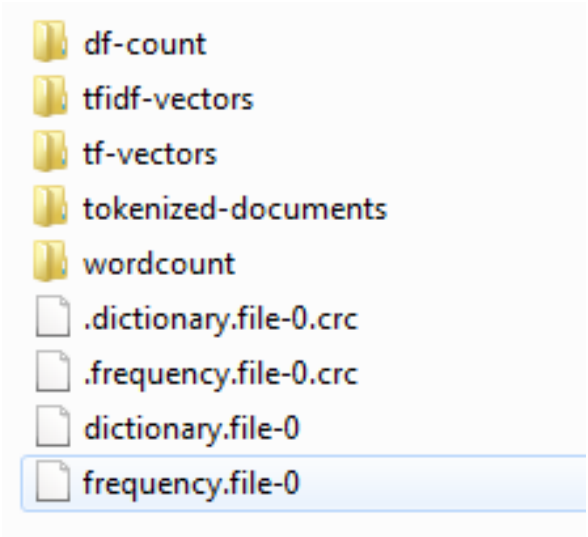


Figure 3.10: The structure of the output folder containing the data generated by Apache Mahout®

the results provided by Mahout® library. There are several output folders found in the file system and the sub-folder *tfidf-vectors* contains the wanted data. All the calculated vectors are stored in a Hadoop® sequence file. The usage of this data will be explained in section 3.7. Another useful output dataset produced by Mahout® library is the file *dictionary.file-0*. This file contains all the terms from the index as well as their mapping to integer values. Namely, the TF-IDF output sequence file contains the TF-IDF vectors in the form shown at the bottom of figure 3.11. Every vector representing the document consist of the *N key:value* pairs. Every pair contains the key representing the term and the term weight. The mappings

[4] `http://technobium.com/tfidf-explained-using-apache-mahout/`, 07.01.2016

```
Document 1 -> I saw a yellow car and a green car.
Document 2 -> You saw a red car.

Step 1: Word count
     car -> 3
   green -> 1
       i -> 1
     red -> 1
     saw -> 2
  yellow -> 1
     you -> 1

Step 2: Word dictionary
     car -> 0
   green -> 1
       i -> 2
     red -> 3
     saw -> 4
  yellow -> 5
     you -> 6

Step 3: Term Frequency Vectors
Document 1 -> {0:2.0,1:1.0,2:1.0,4:1.0,5:1.0}
Document 2 -> {0:1.0,3:1.0,4:1.0,6:1.0}

Step 4: Document Frequency
      -1 -> 2
       0 -> 2
       1 -> 1
       2 -> 1
       3 -> 1
       4 -> 2
       5 -> 1
       6 -> 1

Step 5: TFIDF
Document 1 -> {0:0.8407992720603943,1:1.0,2:1.0,4:0.5945348739624023,5:1.0}
Document 2 -> {0:0.5945348739624023,3:1.0,4:0.5945348739624023,6:1.0}
```

Figure 3.11: The data generated by Apache Mahout® for only two documents[4]

between the keys and the actual terms are stored in the dictionary file.

The book [11] served as the main material used to understand how Mahout® actually works and how it can be efficiently used.

3.7 Storing data in MySQL® database

As mentioned in section 3.6, the TF-IDF matrix is stored in the Hadoop® sequence file found in the sub-folder *tfidf-vectors*. A single row in the sequence file represents the TF-IDF vector of one article revision. The rows in the sequence file are not sorted, meaning that there is no guarantee that two consequent rows are the TF-IDF vectors of two consequent revisions of the same article. This fact is very important as the similarity calculation is done exclusively on consequent revisions (sorted by timestamp) of a single Wikipedia article. That implies that some kind of sorting of article revisions by date of creation is needed.

The most efficient way to do this is to store data in a database. The database engine of choice for this master thesis is MySQL® relational database. The simplicity, familiarity with the system, large community and performances are some of the reasons behind such a choice. Apart from the actual database engine used, the idea was to use JPA (Java Persistence API) in combination with Hibernate ORM (Object-Relational Mapping) in order to develop a system where it would be possible to change the database engine without any modification of the code. Using ORM as a persistence provider is the best solution. For example, if one would like to use MSSQL instead of MySQL®, the only step it would take to make this possible is to change the configuration file of the persistence provider. Aside from Hibernate there are some other persistence providers like EclipseLink, EJB (Enterprise Java Beans) and so on. A system developed upon this idea looks similar to the one depicted in figure 3.12.

When ORM is used, there is no need to design the actual tables in the destination database. One defines the entities using Java Annotations. The annotations help to define the role of member variables of the class annotated as an entity. This way, the actual table in the database is defined by Java class, thus achieving a higher level of abstraction. Replacing the target database engine means that there is no need to adapt the existing entities. The UML shown in figure 3.13 depicts the correlation between three defined entities. Once the program is started, the database tables are created and the entity classes are used as a template. The biggest gain achieved by taking this approach is the way that inserting data into the database tables is done. By simple instantiation of entity classes followed by calling the *commit()* and *persist()* methods upon the newly created instances, the new data is persisted into the database. This means that an instance of the entity class can be seen as a row in the corresponding table of the database. Every entry in the database table representing the entity class *Revision* is uniquely identified by the compound key consisting of the article ID and revision timestamp. This automatically solves the problem of revision sorting by date as the entries in the table are automatically sorted by primary key value. Figure 3.14 shows the reverse-engineered correlation between the tables in MySQL® database used in the proposed piece of software

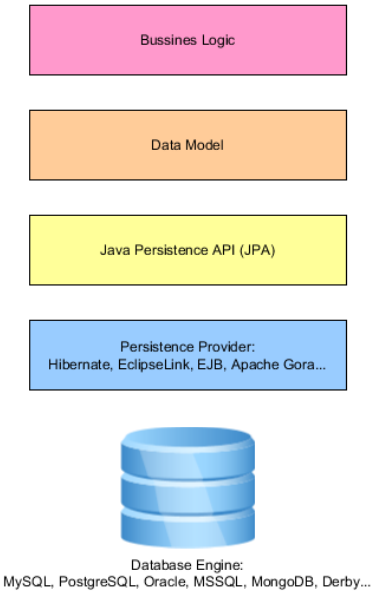


Figure 3.12: The storage architecture when JPA and a persistence provider are used.

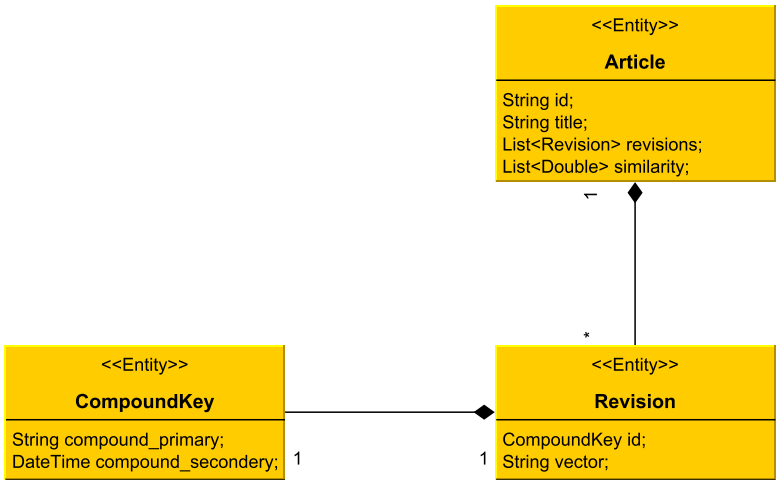


Figure 3.13: The correlation between the defined persistence entities.

and generated by Hibernate. The book [7] offers a good insight into this topic but there is plenty of material online that can be used to grasp the functionality provided by different persistence providers.

Now, when the infrastructure used to store the TF-IDF data is discussed, the parsing and processing of the data stored in the Hadoop® sequence files has to be explained. As mentioned at the end of section 3.6, the TF-IDF vector representing a single Wikipedia article

Figure 3.14: The correlation between the generated tables in MySQL®.

revision is stored in a single line of the sequence file. Hence, all the lines in the sequence file are iterated and parsed. At this point, it would be useful to mention that Fork/Join Framework was used again. The idea was to read and buffer the number of lines from the sequence file and then to use the *divide-and-conquer* strategy implementation provided by the FJF. All the buffered lines are then parsed in parallel. The line containing the TF-IDF vector of article revision is split into key:value pairs as explained in section 3.6. These pairs had to be sorted since only the top N terms are required by the rank-biased overlap method for similarity calculation. This is explained in section 2.3. Most of the TF-IDF vectors have several hundreds of key:value pairs and there are literally millions of vectors to be sorted. This only confirms the need for efficient sorting of key:value pairs. A Google Guava Java collections library provides a data-structure called *TreeMultimap* which was used for this purpose. It is only required to simply add the key:value pairs to this data structure and they will be automatically sorted by weight value. At the and of this procedure, one takes only a certain number of top weighted pairs into consideration. Since the actual term weights are not important for the similarity measure but rather the terms themselves, only the terms from the top N key:value pairs are stored in the database. The entity *Article* has an attribute *vector* and it is used to store these terms. So, currently, when an instance of the entity class

Article is available, one can get all its revisions sorted by the timestamp where every revision is represented by N top weighted terms.

Since the processing of those vectors from the sequence file runs in parallel as mentioned earlies, after some time, the FJF provides the results of processing of all buffered vectors at the same time. The most efficient way to write this new data (vectors containing N top weighted terms) is to use the so-called batch write capability of both JPA and MySQL. This feature only has to be enabled in the configuration file of the persistence provider. Instead of creating a separate *INSERT* statement for each revision, Hibernate generates a large statement updating all the revisions by a single *INSERT* statement. This approach results in a considerable performance boost.

4 Experimental setup and results

In section 1.2 the author defined the goals of this master's thesis. To shortly summarize it, the main goal was to develop a piece of software that analyses the semantic stabilization process of Wikipedia editions in different languages. Once the results are acquired it would be possible to compare the stabilization process of different Wikipedia language editions and, maybe, find some correlation between the sheer size of Wikipedia, the language it is written in and its stabilization process over time. The languages proposed by both the mentor and the author of this thesis are:

- English
- German
- French
- Spanish
- Italian
- Czech
- Finnish (Suomi)
- Danish
- Greek and
- Swedish

where the first group of languages (the first five languages from the list above) are referred to as *Large Wikipedia editions* and the other five make the group of *Small Wikipedia editions*. There is a twofold difference between the Wikipedia editions from these two groups:

- the number of articles
- the number of revisions per single article

Before the analysis of Wikipedia editions in proposed languages, an experiment with random generated data as the input of the software tool was planned. The aim of this experiment was to validate the similarity and stability calculation algorithms. The results are provided in section 4.2.

After all the proposed languages were analysed, an additional experiment was proposed and calculated. The idea of this last experiment was to use the same input Wikipedia language (in the case of this experiment, English Wikipedia was used) but to change the value of parameter p of the rank-biased overlap algorithm as explained in 2.3.1.

The whole chapter 2 as well as the part of this chapter is conceived as a way to prepare the reader to correctly interpret the results produced by the software implemented for this thesis. The results of the experiments are given in the form of plots. As the graphical presentation of the results is, in most cases, the most efficient presentation strategy in terms of simplicity and readability, it was selected to represent the experimental results of the implemented software tool.

All of the experiments are done with the same settings. The rank-biased overlap similarity measure algorithm is parametrized with the $p = 0.9$ which means that the first ten ranks of the ranking list have 86% of the weight of the evaluation as stated in [1]. The author of this thesis finds $p = 0.9$ appropriate because of the value of parameter d (depth of evaluation) chosen for rank-biased overlap. This means that the TF-IDF vectors will be checked for similarity only up to the depth of 20. Of course, one can take a much higher depth, but that will increase the computation time as well as the storage space. Namely, the TF-IDF vector representing a single revision of an arbitrary article can have several thousands of values, but not all of those values are stored. Only the values up to the depth needed for rank-biased overlap calculation are stored. So, if 20 elements are used for rank-biased overlap measure, the first 10 elements of the ranking weight 86% of the evaluation and the other 10 elements weight only 14%. It is exactly because of this fact that there is no need to do the similarity calculation for much higher depths as those are not regarded as very important. In every case, the top 20 (most-weighted) elements of the TF-IDF vector are more than enough to precisely describe the semantics of the article revision they represent. When $p = 0.9$, the maximal value for similarity measure (when rankings are identical) has the value 0.87 so the stability threshold k is set to $k = 0.4$.

Another crucial aspect of this experiments is the size of the Wikipedia XML dump file being analysed. Some of these Wikipedia editions are relatively small so the dump files are up to a few gigabytes in size. But English Wikipedia which is the largest one, is more than one terabyte in size (compressed) so it is clear that the sampling over a complete data has to be performed. Only some of the smaller Wikipedia edition are analysed completely. The size of the sample is 10000 randomly selected articles.

4.1 Semantic similarity and stability analysis

In the previous sections of chapter 3, it was explained how the Wikipedia corpus data, provided by an XML dump file, can be parsed and later indexed with Lucene®. Once the index directory is generated, the next step was the calculation of the TF-IDF matrix. As the vectors, representing the article revisions in the calculated matrix are not sorted by timestamp, the solution for efficient sorting had to be found. All the vectors are stored in a database of choice, so the sorting is not a problem any more. The article data is now ready to be analysed and used for semantic similarity and stability calculation and this last module of the proposed software will be discussed in this section.

As mentioned at the beginning of this section, all the article data is stored in the database. By calling a single method provided by the persistence provider, in this case Hibernate, one can get all the available instances of the entity class *Article*. Since there could be millions of article instances available, calculating the similarity between the revisions of an article one by one is not the most favourable way of doing it. The Fork/Join Framework will be used once more. Instead of getting all the available articles and iterating over them, only a portion of articles will be acquired and buffered. The *divide-and-conquer* strategy will by applied on the buffered articles and N articles will be processed at the same time. This process repeats until there are articles available. The principle behind this is completely the same as when the FJF was used and discussed in the previous sections of this thesis.

The calculations of semantic similarity and stability are two separate processes and the design of the proposed software is such that those two calculations are independent from one another. One can disable any of these two modules at any time and use the data from a previous run for one of them. But still, the stability calculation can be seen as a way of interpreting the results of the similarity calculation. Because of this fact, the author of this thesis sees these two calculations as two parts of the same module in the proposed software implementation.

Semantic similarity implementation

The semantic similarity module is designed to fulfill the *Factory Method* design pattern [3, p. 121]. At the moment, the goal of the software developed for this thesis is to use rank-biased overlap similarity calculation method. However, it is possible that at some later point an additional similarity measure will be needed. This way, with implemented factory method design pattern, the implementation of additional similarity calculation algorithm would be easy. The UML diagram shown in figure 4.1 represents the implementation details of the proposed software solution. The class *SimilarityForkJoin* acts as a client in this design. It simply requires a new instance of the concrete realisation of the interface *ISimilarityCalculation*. It provides only the type of the similarity calculation algorithm to be used. The aforementioned types can be defined in the enumeration class *SimilarityCalculationType*.

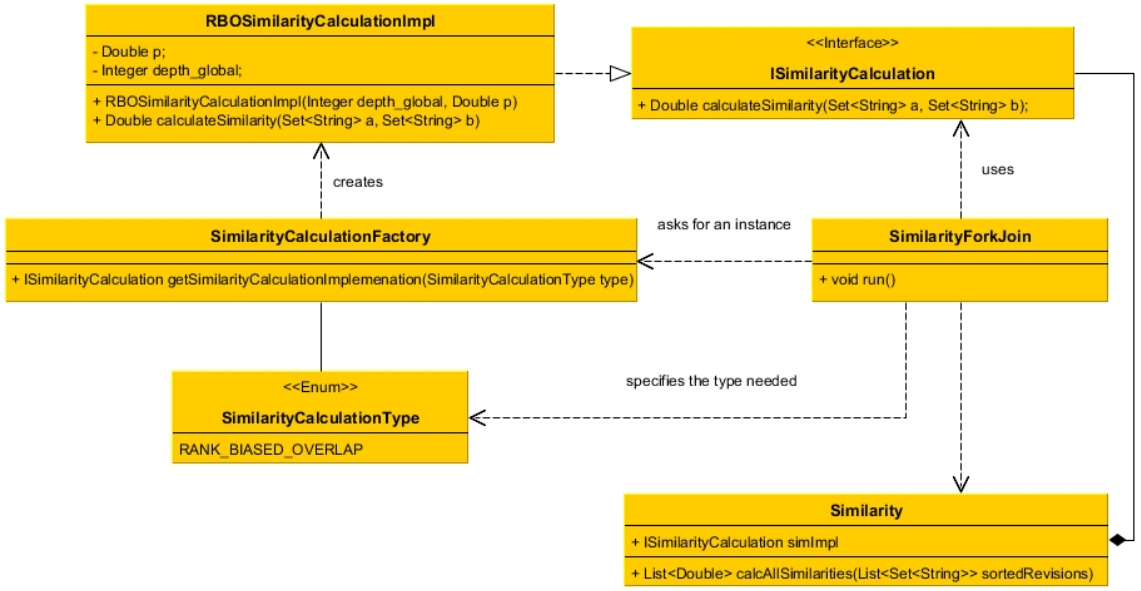


Figure 4.1: The UML diagram of the module responsible for semantic similarity calculation.

When a new similarity calculation algorithm is needed, one only has to implement a new class realising the corresponding interface and to register the new algorithm type in the class *SimilarityCalculationType*.

The implementation of the rank-biased overlap similarity measure is rather simple and it is based on the theory explained in section 2.3.1. For calculation, the equation 2.12 is used. The most important part in the implementation of this module, aside from parallelism, was to find an efficient way to calculate the intersection of the two sets of strings where one set of strings represents the N top weighted terms. The code snippet shown in figure 4.2 shows the implementation details. The used collections for implementation of this code, namely *LinkedHashSet* are the ones provided by the Google Guava library. The instantiation of these sets is done in $\mathcal{O}(1)$ so this part of the code performs very well. Once the similarity of all consequent revisions of an article is calculated, the obtained values form a *semantic similarity vector*. This vector is saved in the attribute *vector* that is defined in the *Article* class. Again, when all the threads running in parallel are done with the processing of the buffered articles, the results are flushed to the database with a single batch insert operation.

Semantic stability implementation

As mentioned in the introduction of this section, the semantic stability calculation process can be seen as a way of interpreting the results of similarity calculation. The module in the proposed software responsible for stability calculation is implemented according to the theory

```
public Double calculateSimilarity(Set<String> a, Set<String> b) {

    double sum = 0;
    int depth = depth_global;
    Set<String> tmp_a;
    Set<String> tmp_b;
    int min = Math.min(a.size(), b.size());
    if(min < depth_global){
        depth = min;
    }

    for(int d = 1; d <= depth; d++){

        tmp_a = Sets.newLinkedHashSet(Lists.newArrayList(a).subList(0, d));
        tmp_b = Sets.newLinkedHashSet(Lists.newArrayList(b).subList(0, d));

        SetView<String> intersection = Sets.intersection(tmp_a, tmp_b);

        sum = sum + ((2 * intersection.size()) / (tmp_a.size() + tmp_b.size())) * Math.pow(p, d-1);
    }
    return (1-p) * sum;
}
```

Figure 4.2: The code snippet showing the RBO implementation details.

presented in section 2.4. For the purposes of this paper, two different aspects of stabilisation process are examined:

1. semantic stabilisation of the Wikipedia corpus over some period of time
2. semantic stabilization of the Wikipedia corpus after the number of successive revisions

The idea behind the examination of the Wikipedia corpus stabilisation over the time is to choose a point in time t and count the number of articles existing at that point in time and the number of articles existing at that point in time that are also semantically stable. This is possible because of the fact that every article revision is uniquely identified in the database by the compound key consisting out of the article ID and the revision timestamp. The value of semantic similarity depends only on two revisions; the last revision before the time point t and the first one after it. If this calculation is repeated for several time instances, one gets a vector of values, which, when plotted, depict the line describing the stabilisation process of the article corpus. Figure 4.3 will be used to clarify how the previously described procedure works. Three articles with their corresponding revisions are available and four apparently random points in time are defined. The red, horizontal line represents the timespan between the timestamps of the first and the last revision of the same article. At time point $t1$, none of the articles have been created yet. The first revisions of all three articles are after the time point $t1$. At time point $t2$, the articles 1 and 2 are still not created but article 3 exists. If the semantic similarity value of the two revisions of article 3 (before and after the time point $t2$) is higher than the user defined stability threshold, the maximal stability of the given document corpus is achieved, $\frac{1}{1}$. At time point $t3$, all three articles exist. If one assumes that only 2 of them are stable, the mean stability would be $\frac{2}{3}$. The last time point is a bit different. All three articles have their latest revisions before time point $t4$. This can be interpreted as a sign that

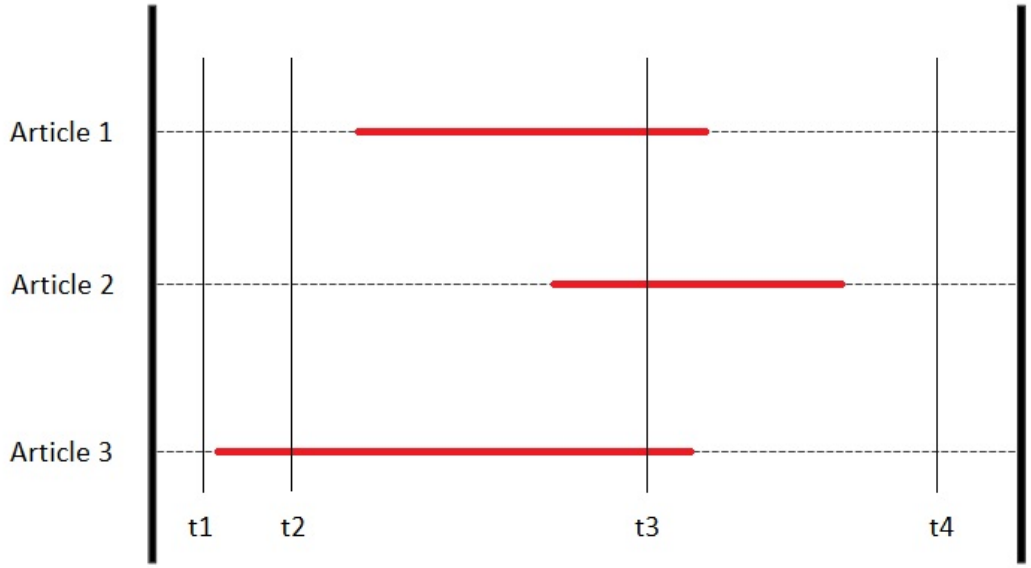


Figure 4.3: The span between the first and the last revision of the three articles.

the articles have not been modified in the recent time and therefore can be taken for stable. So, again, the maximal similarity is achieved $\frac{3}{3}$. The acquired mean stability values are used to plot this semantic stabilisation characteristic of the Wikipedia corpus.

Another way to inspect the stabilisation process of the document corpus is to find out how many successive revisions are required before a percentage of the available articles becomes stable (in reference to the stability threshold). The idea is very similar to the previously discussed one, but now it is assumed that all articles have the first revisions starting at the same date and time. This information (timestamp) is now completely neglected and only the number of revisions per article is important. So, at the beginning, the first value of the similarity vectors of all articles is examined. The stability threshold takes the maximal value at the beginning of the calculation, 1. If the desired percentage of the articles is stable, the next value of the similarity vector is inspected. If not, the threshold is decreased and the calculation is repeated until the value of the stability threshold, for which the desired percentage of articles is stable, is found. The interpretation of data marked with point *r* in figure 4.4is as follows: 95% of the articles in German Wikipedia edition are stable (for semantic stability threshold value 0.55) after almost 50 successive revisions.

This way, with two similar but still different kinds of semantic stability analyses discussed in this section, one gets more useful information about the examined corpus.

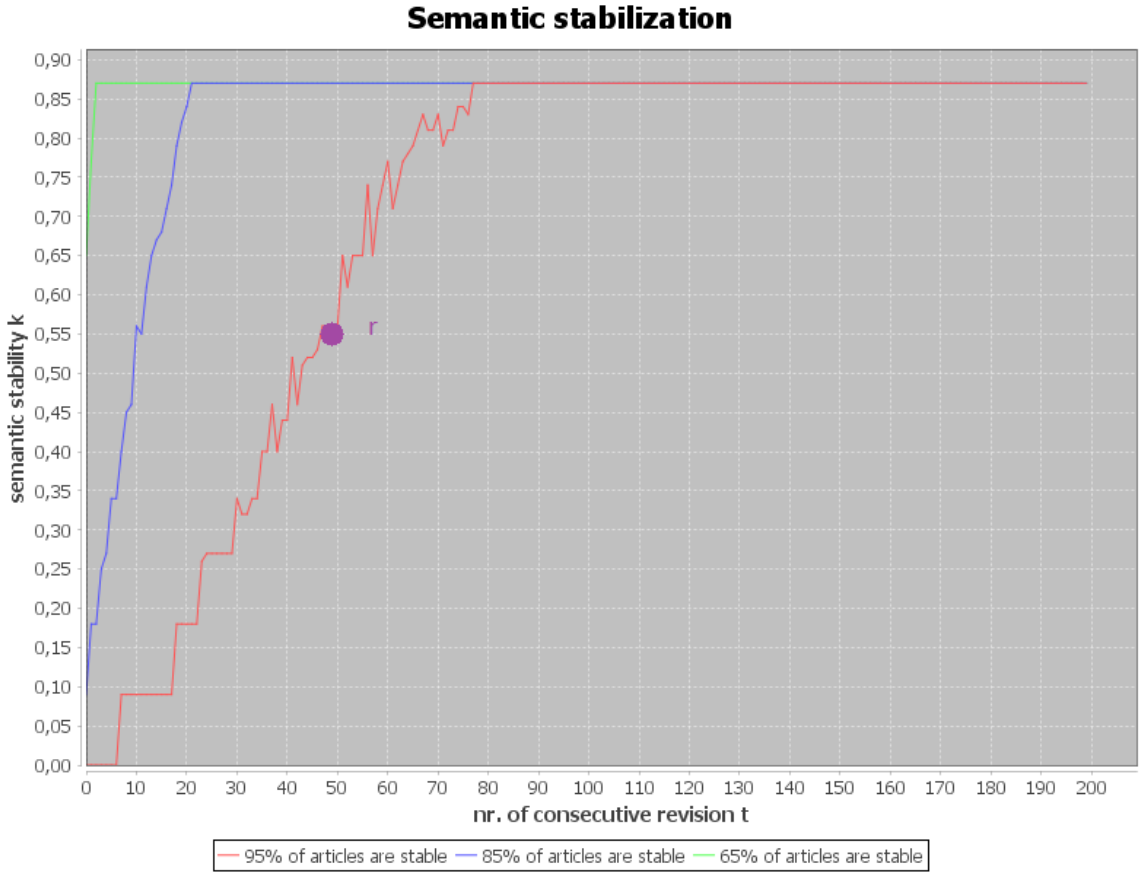


Figure 4.4: An example of the plot produced as the result of the semantic stability analysis of the German Wikipedia edition when stability is calculated as the dependence of the number of successive article revisions.

4.2 Calibration test with random data

The first experiment in this chapter is thought to be some sort of calibration test. In section 3.7, it is explained how Wikipedia articles are represented and stored in a relational database. Every single article in Wikipedia has at least one revision, but normally there are a lot more of them, usually several tens up to several thousands of revisions. Every revision is represented with a single TF-IDF vector in vector space and stored in the database. The revisions are then compared with each other by the rank-biased overlap algorithm explained in 2.3.1, thus generating the similarity vector. The data about Wikipedia editions is acquired through the analysis of the values in that similarity vector (there is exactly one similarity vector per Wikipedia article in the database). As explained in previous sections, the rank-biased overlap algorithm for two revisions that are the same (same TF-IDF vectors) outputs a maximal value 1 and for two revisions that are completely different it returns 0.

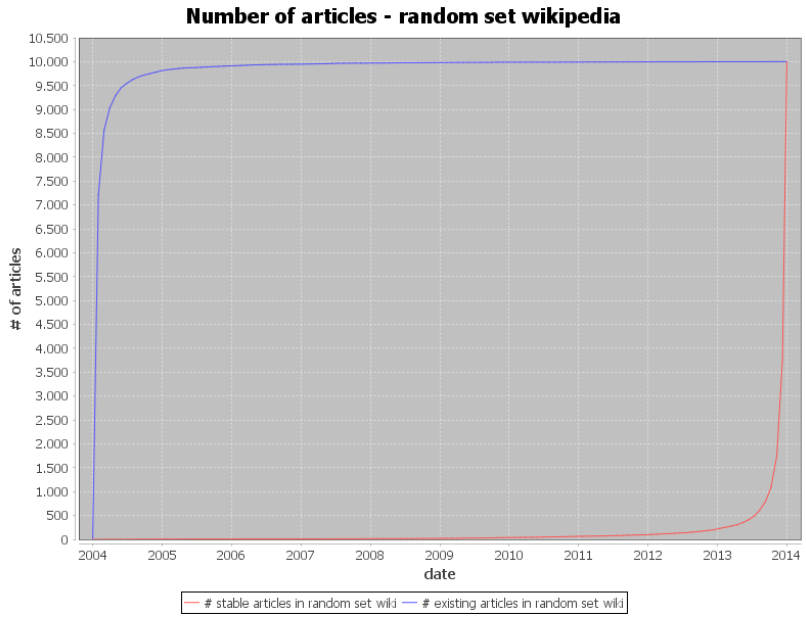


Figure 4.5: The cumulative number of articles and the number of stable articles over time where the articles are represented by vectors of random data.

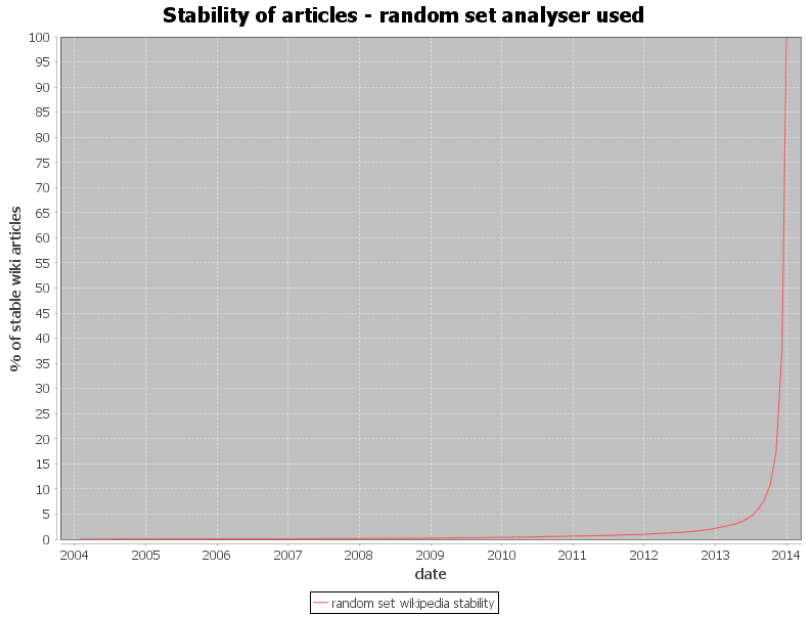


Figure 4.6: The percentage of the stable articles where the articles are represented by vectors of random data.

So, the aim of this experiment is to generate random vectors instead of real revision vectors and then to analyse the stabilization process. The time period for analysis is from January

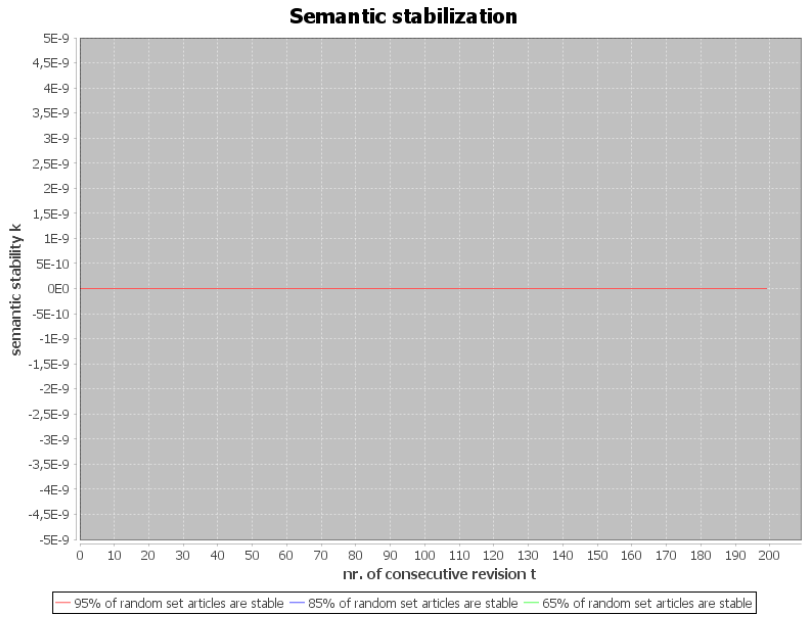


Figure 4.7: The architecture of the proposed software solution.

1st, 2004 to January 1st, 2014 as well as for the randomly generated timestamps for revisions. Figure 4.5 clearly shows that the cumulative number of *simulated* articles is at the maximum almost right away (article creation timestamp, first revision, is random, hence, not all of the articles created at the beginning) and it stays so until the end of the examination period. The exact opposite is true for the number of stable articles. There were no stable articles until the end of examination period. Actually, figure 4.6 shows that there are *no stable articles* observed over the period of time. The increase of stability in figure 4.5 is a consequence of the fact that there are no articles with randomly generated revision timestamps between the last two time-points of the examination. The stabilization algorithm assumes that all such articles are stable. Figure 4.7 shows that 65%, 85% and 95% of the articles never reached stability.

4.3 Small Wikipedia editions experiments

Czech Wikipedia experiment

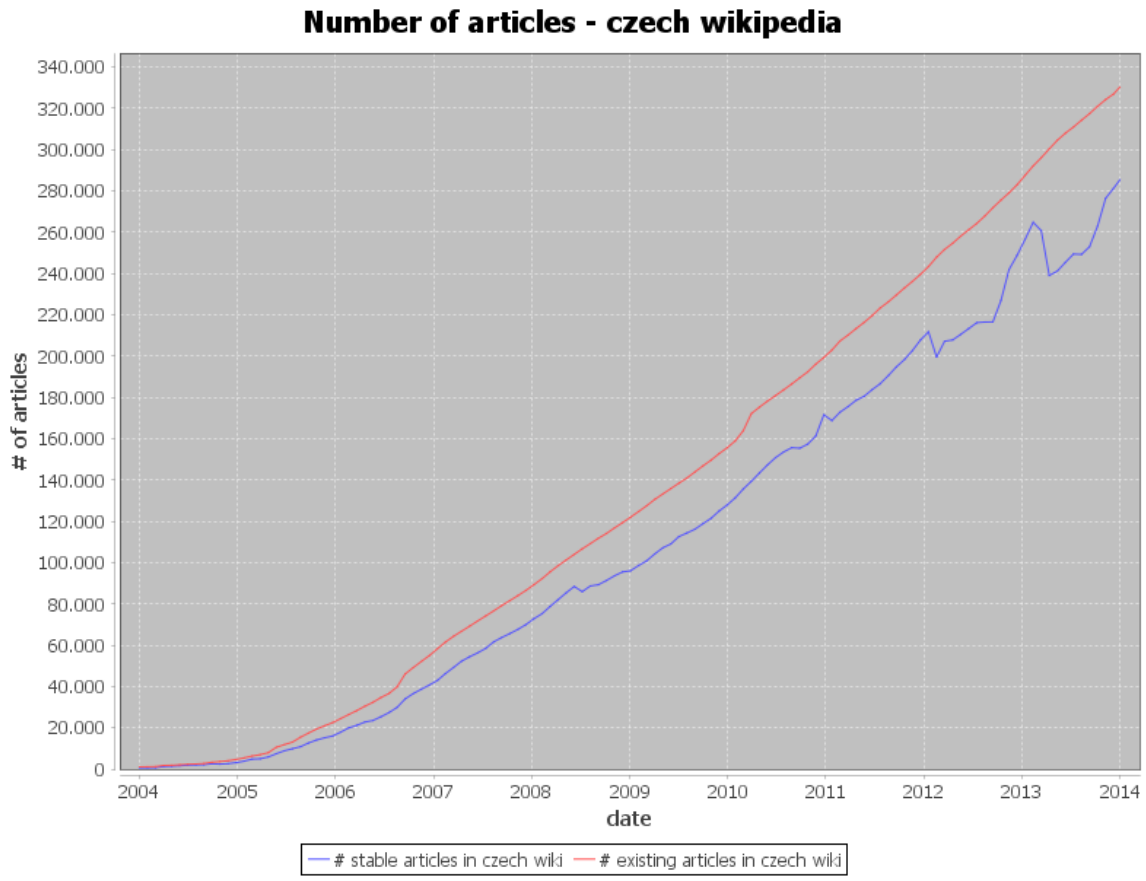


Figure 4.8: The stabilization process of the Czech Wikipedia edition.

The Czech Wikipedia XML dump file was analysed entirely for the purpose of this thesis, thus the sampling of the article corpus was not necessary. Almost all the other experiments in this thesis were performed on the reduced corpus of documents simply because of the fact that indexing a huge amount of data takes considerable time and, in order to demonstrate the possibilities of the software implemented for this master thesis, analysing the sampled data was sufficient to produce trustworthy results. The first articles for the Czech Wikipedia were created in the year 2002 and there are, currently, about 340 thousand articles. Figure 4.8 shows the process of stabilization. It is visible that up to year 2009, there were under 100 thousand articles available in this Wikipedia edition and that the most of them were semantically stable. In the period afterwards, almost a constant increase in number of articles is noticeable and the stability is preserved at the same time, meaning there were no major semantic changes in articles.

Danish Wikipedia experiment

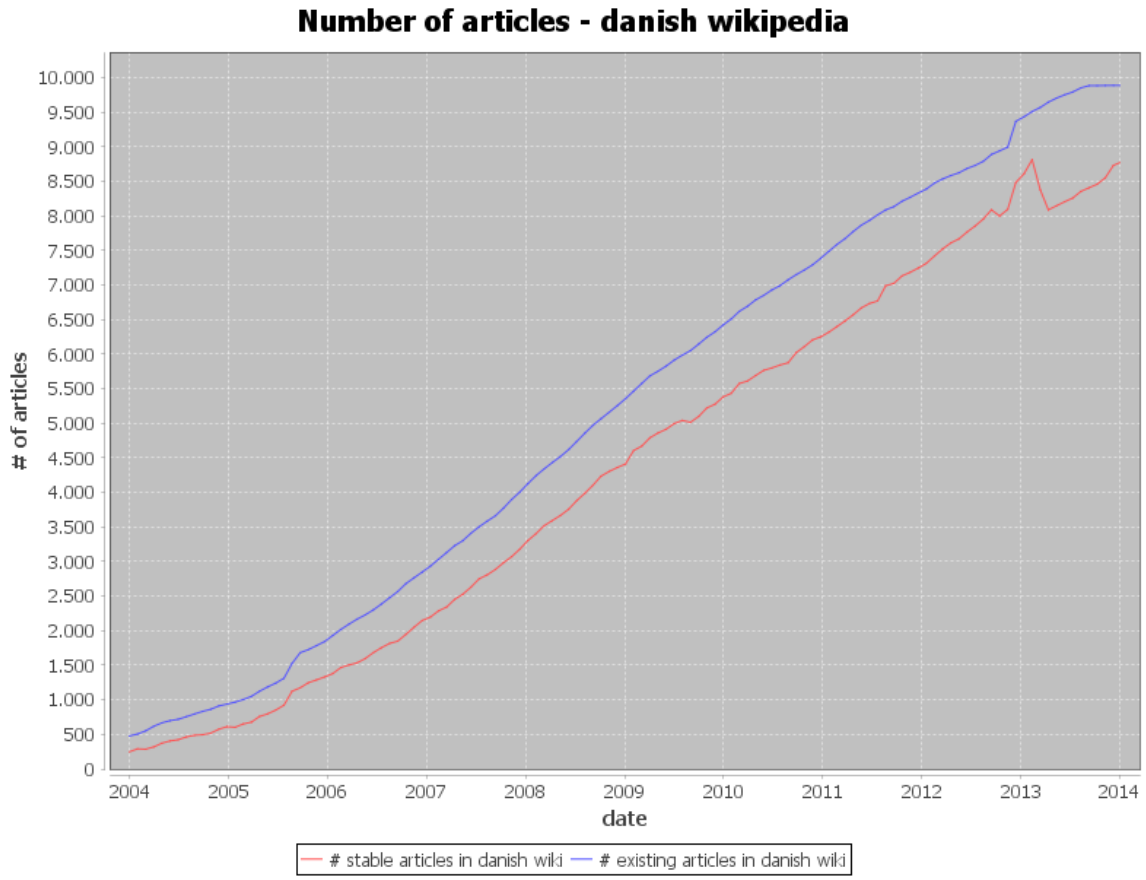


Figure 4.9: The stabilization process of the Danish Wikipedia edition.

Figure 4.9 shows the stabilisation process of the Danish Wikipedia. This Wikipedia edition counts about 215 thousand articles and it is one of the smallest editions examined in this paper. For the purpose of this experiment, not the complete corpus of Danish Wikipedia was used. Just as it is the case with other sampled Wikipedia editions, a sample consisting of 10 thousand articles was examined. The two lines in the graph, representing the cumulative number of articles and the number of semantically stable articles from Danish Wikipedia, have very similar characteristics to those from section describing the Czech Wikipedia experiment. A small increase in stability is noticeable at the beginning of year 2013.

Finnish Wikipedia experiment

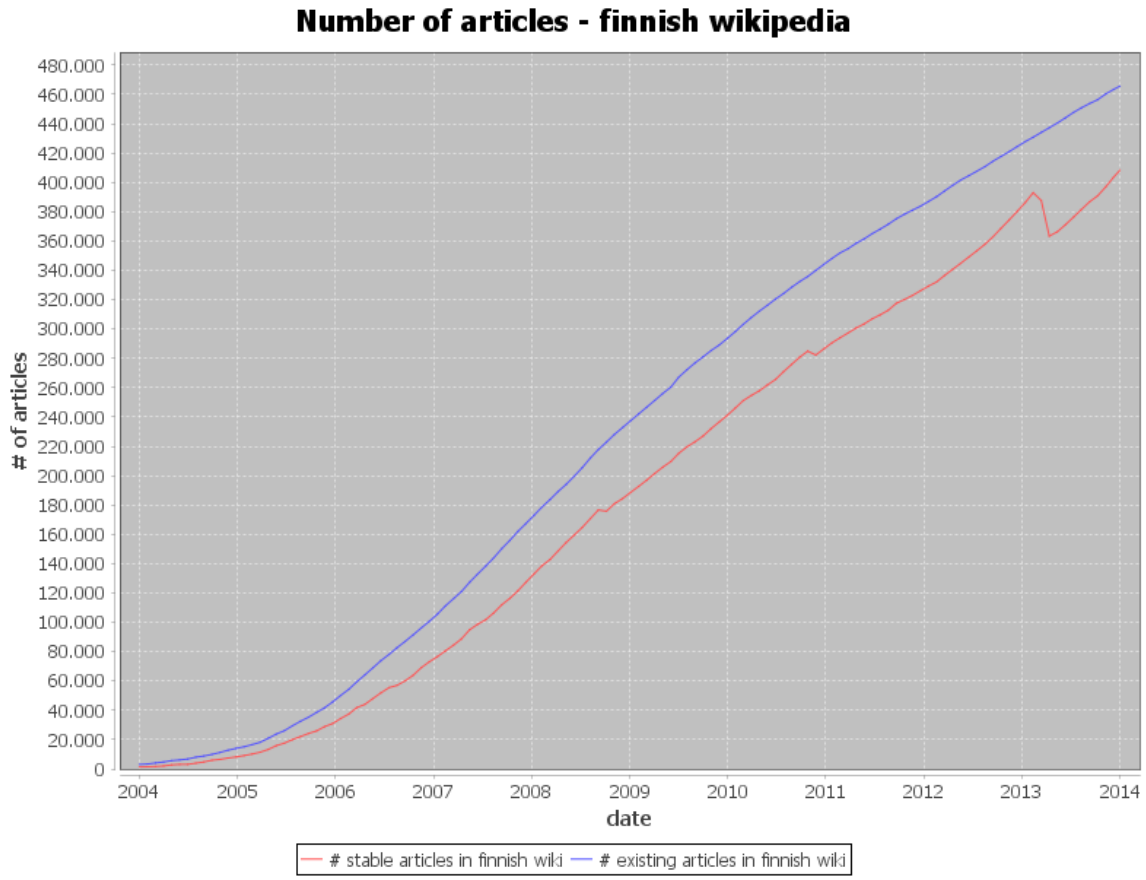


Figure 4.10: The stabilization process of the Finnish Wikipedia edition.

The process of semantic stabilisation of the Finnish Wikipedia edition is shown in figure 4.10. In this experiment, the complete corpus of the articles provided in Finnish Wikipedia XML dump file was used for the analysis, without sampling. Although the full analysis takes a lot of time, it was necessary so the results obtained could be compared with the stabilisation data acquired from the sampled versions of other Wikipedia editions. It is clear that the sampling procedure strongly reduces the time needed for analysis by reducing the input set of data, but this also means the loss of the result precision. When the data from this experiment is compared to the results of other experiments, the similarity is really noticeable. The characteristic form of the lines representing the number of available articles and the number of stable articles from the document corpus can be observed. Furthermore, both of the lines are almost constantly increasing , as in the majority of other experiments presented in this chapter. The Finnish Wikipedia has almost 385 thousand articles.

Greek Wikipedia experiment

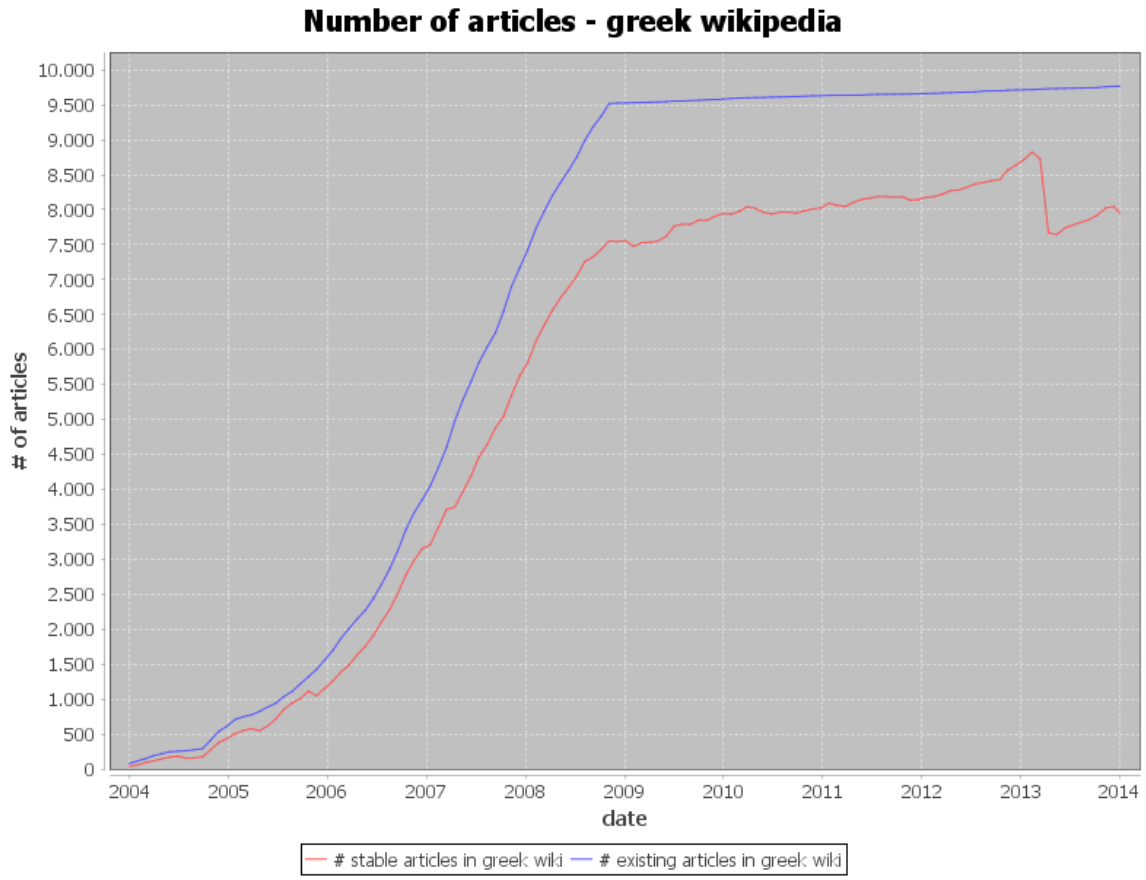


Figure 4.11: The stabilization process of the Greek Wikipedia edition.

The Greek Wikipedia XML dump file is the smallest one among the Wikipedia language edition being analysed in this thesis. With a bit under 115 thousand articles, it is even smaller than the Danish Wikipedia edition. Among the Wikipedia's used as input data for the experiments in this chapter, it is the only edition written in non Latin alphabet. For this experiment, the reduced size of articles is used. Figure 4.11 shows the semantic stabilization process of this Wikipedia edition. This time, compared to the previous three experiments, the results are a bit divergent to what has been seen until now. Although the used input data set consists of random ten thousand articles from the entire corpus of documents, figure 4.11 makes it obvious that the expansion of articles over time is not linear like as is the case with other analysed Wikipedia editions. Linear increase of the total number of articles is noticeable roughly until the end of 2008 and from that point on, the expansion is much slower but still present. Despite this observation, the ratio between the stable and non stable articles remains similar to the one observed in the previous tree experiments. Even the sudden increase in stability can be seen around the beginning of 2013.

Swedish Wikipedia experiment

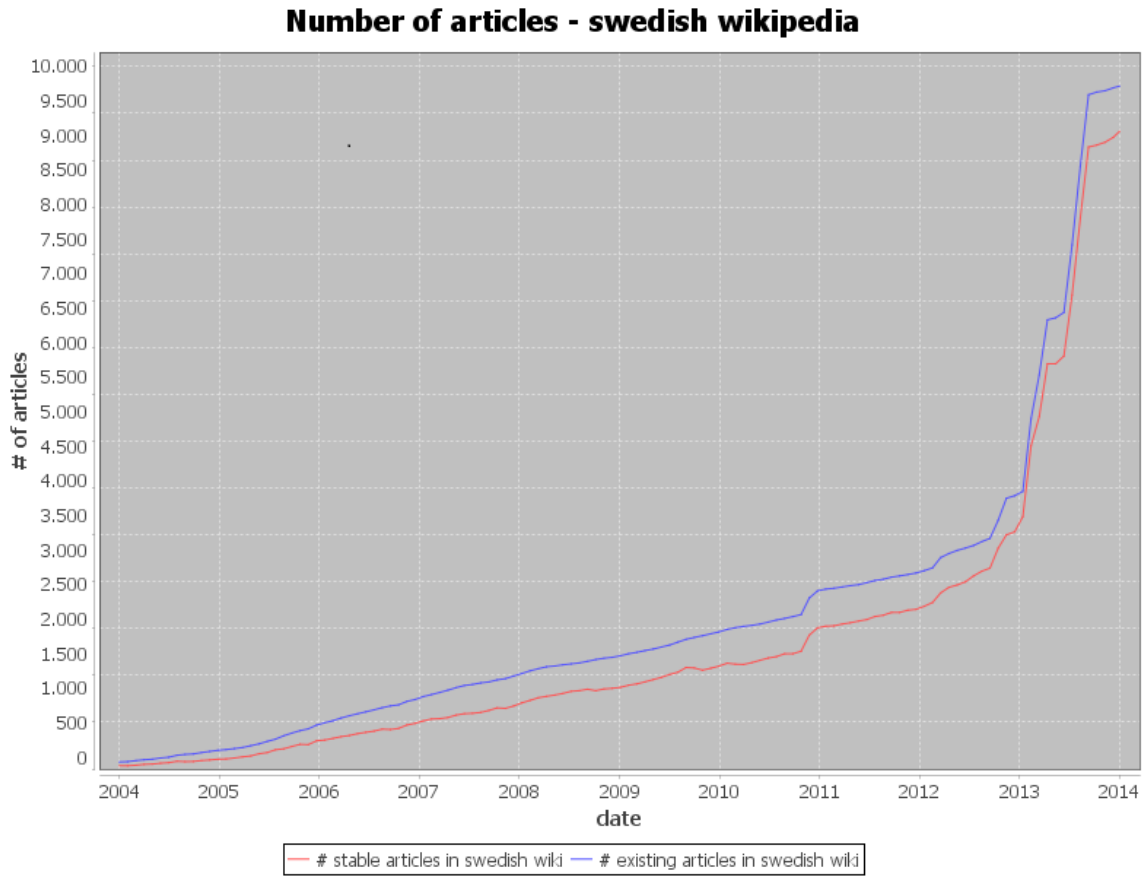


Figure 4.12: The stabilization process of the Swedish Wikipedia edition.

The last experiment on the input data from a group of small Wikipedia editions was performed on Swedish Wikipedia. Compared to the other Wikipedia's from this group, the Swedish edition is a lot larger having almost 2.3 million articles. The stabilization process observed over a sample consisting of 10 thousand articles is shown in figure 4.12. Just as was the case with Greek Wikipedia, the growth of this Wikipedia was not constant. A slow growth is noticeable in the period from 2004 to 2013, and then there is a huge increase in the total number of articles available in Swedish Wikipedia until the end of the observed period. What is strange about the results of this experiment in comparison to the other experiments is that the number of stable articles was always a lot lower than the total number of the articles examined in this experiment, which implies a lower semantic stability of the sample data, compared to other results. It is only possible if there was a large number of article revisions of the sampled articles. Although this Wikipedia edition is small compared to the largest editions, it is significantly larger than the other editions from this group, thus having much larger community working on this project and producing more revisions.

Stabilization process of small Wikipedia editions - Comparison

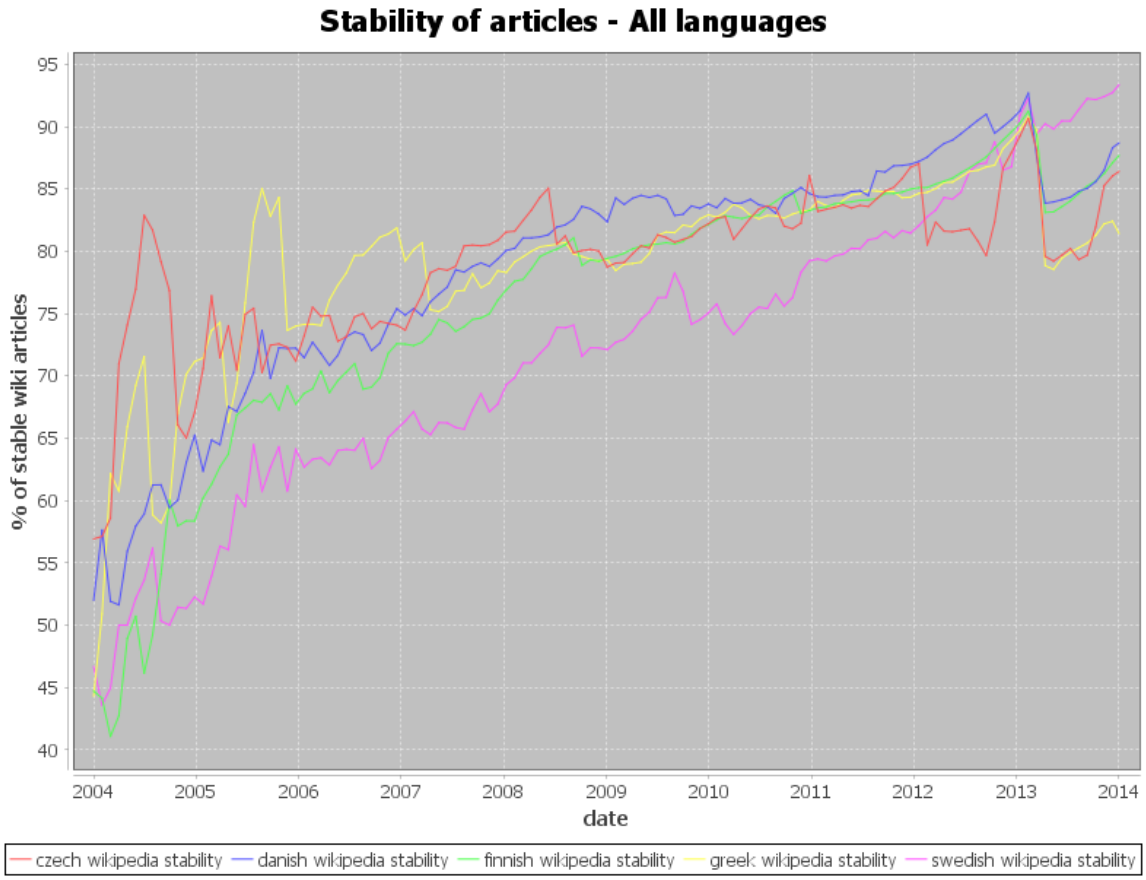


Figure 4.13: The percentage of the semantically stable articles in five different and relatively small Wikipedia editions.

The sole purpose of the graph presented in figure 4.13 is simply to compare the stabilisation process of the analysed Wikipedia language editions in the previous five sections of this paper. The difference to the previous figures is that, in this case, a portion of the stable articles (in percentages) instead of the total number is given, as well as the number of stable articles. This way, it is a lot easier to spot periods of increased stability or instability of an article corpus. When stabilisation data of the five Wikipedia editions is plotted, it is noticeable that all Wikipedia editions have the semantic stability variations in almost the same range, and it is normally $\pm 2\%$ from the average. The only exception to this is the case of Swedish Wikipedia. In figure 4.13 it can be seen that Swedish Wikipedia has the semantic stability well below the average semantic stability of the other four Wikipedia editions. The logical explanation for this is that the small Wikipedia editions consist mainly of articles which are the translated versions of the articles from the main Wikipedia editions (for example from English Wikipedia). Once translated and created, such articles are rarely edited a lot.

Stabilization process - 65%, 85% and 95% of articles from small Wikipedia corpus

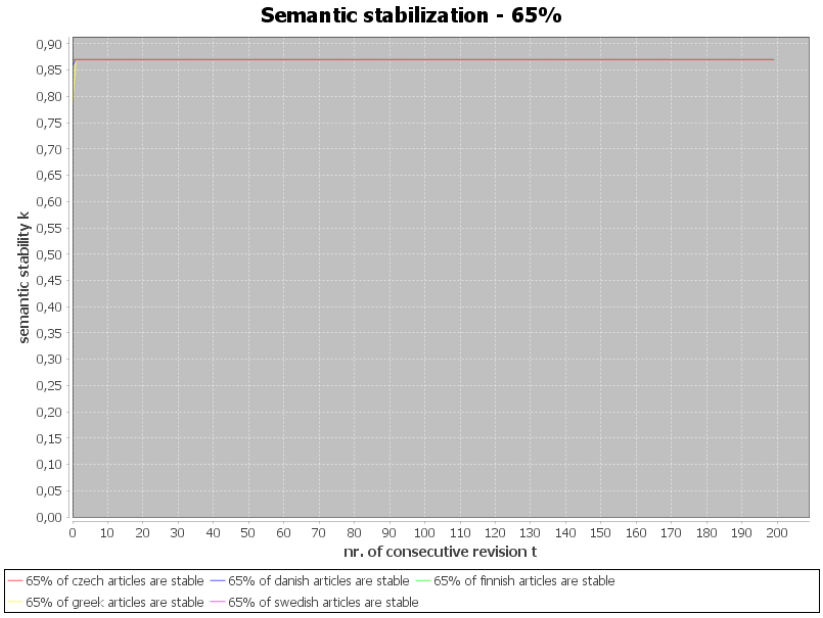


Figure 4.14: The number of revisions needed to achieve 65% semantic stability for stability threshold k.

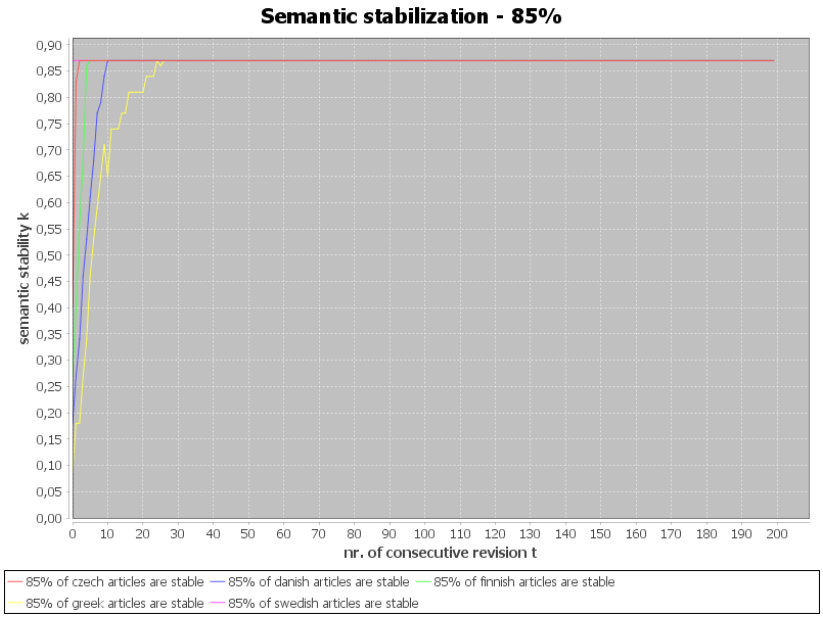


Figure 4.15: The number of revisions needed to achieve 85% semantic stability for stability threshold k.

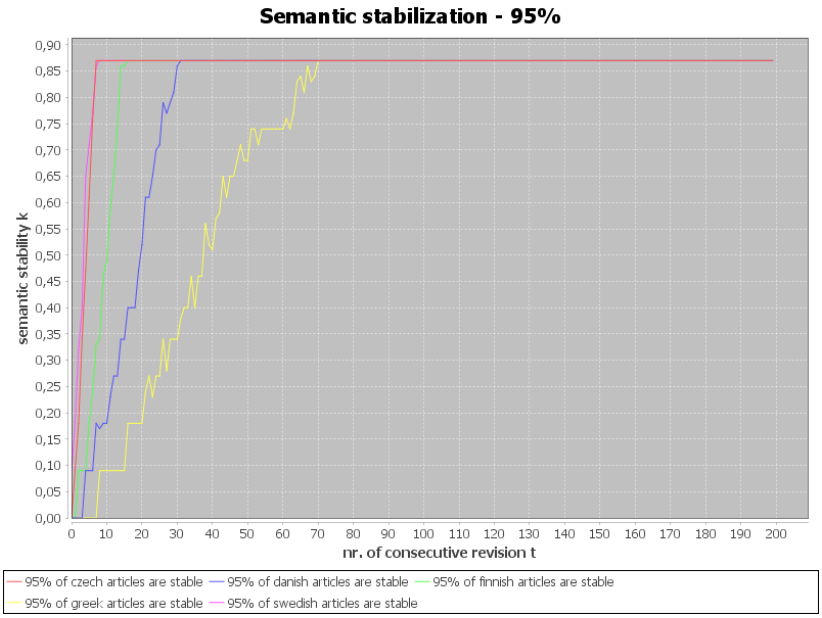


Figure 4.16: The number of revisions needed to achieve 95% semantic stability for stability threshold k.

Figures 4.14, 4.15 and 4.16 are used to visualize the number of consecutive revisions per article are needed to achieve the stability of 65%, 85% and 95% of the analysed articles. It can be seen from figure 4.14 that 65% of the existing articles in all five small Wikipedia editions get stable after only a 2-3 revisions even when the stability threshold k has a high value. When the percentage of the desired stable articles is higher, in this case 85%, all the analysed Wikipedia editions need less than ten revisions after which they get stable. The only exception is the Greek Wikipedia edition whose articles need as many as 30 revision before the stability is reached. In figure 4.16, 95% of stable articles is wanted and it is reached after 70 revisions for Greek Wikipedia and 30 or less revisions for all other small Wikipedia editions observed. From the last plot it can be seen that for the Greek Wikipedia edition, 95% of the article corpus has the stability of 0.5 after almost 40 revisions and $k = 0.5$ is only a medium stability. From this fact one can conclude that the Greek Wikipedia edition was the most frequently edited Wikipedia amongst the analysed, small Wikipedia editions. The Czech and Swedish editions are showing much more semantic stability. 95% of the article corpus has the semantic stability of 0.5 after only about 5 revisions.

4.4 Large Wikipedia editions experiments

English Wikipedia experiment

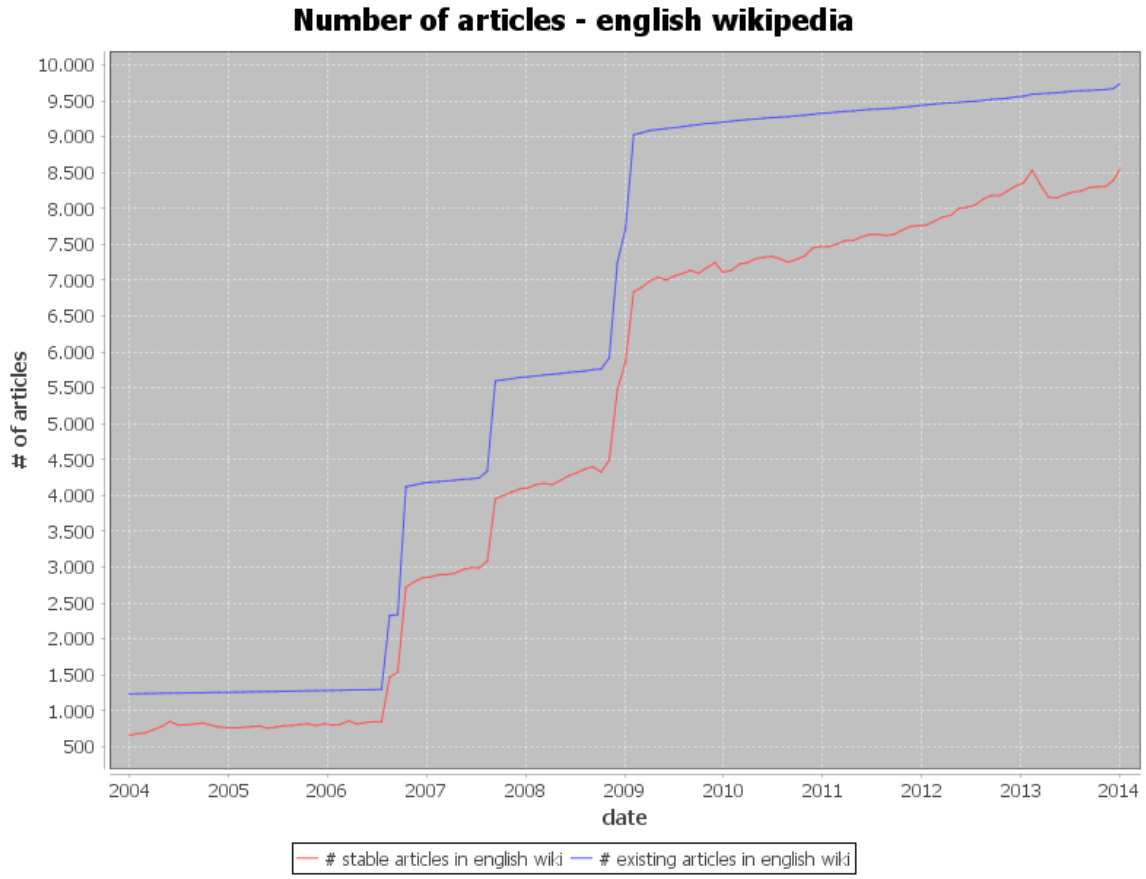


Figure 4.17: The stabilization process of the English Wikipedia edition.

The English Wikipedia edition is the largest one with more than 5 million articles and hundreds of millions revisions. The size of the XML dump file that is larger than a terabyte implies the necessity of the article corpus sampling. In this series of experiments over the group of five largest Wikipedia editions, the analysed data is always sample of 10000 randomly selected articles. Figure 4.17 shows the stabilization process of the English edition. This edition is different from the previous experiments because now it is noticeable that not the full corpus of articles has been analysed so both red and blue lines have the form of *stairs*. But as the analysed data consists of the randomly selected articles, the corresponding stabilization values should be representative for the complete corpus of articles. An important observation is the ratio of the total number of articles and the number of stable articles. Low semantic stability can be seen in the entire observed time period when compared to the experimental results of the editions from section 4.3.

French Wikipedia experiment

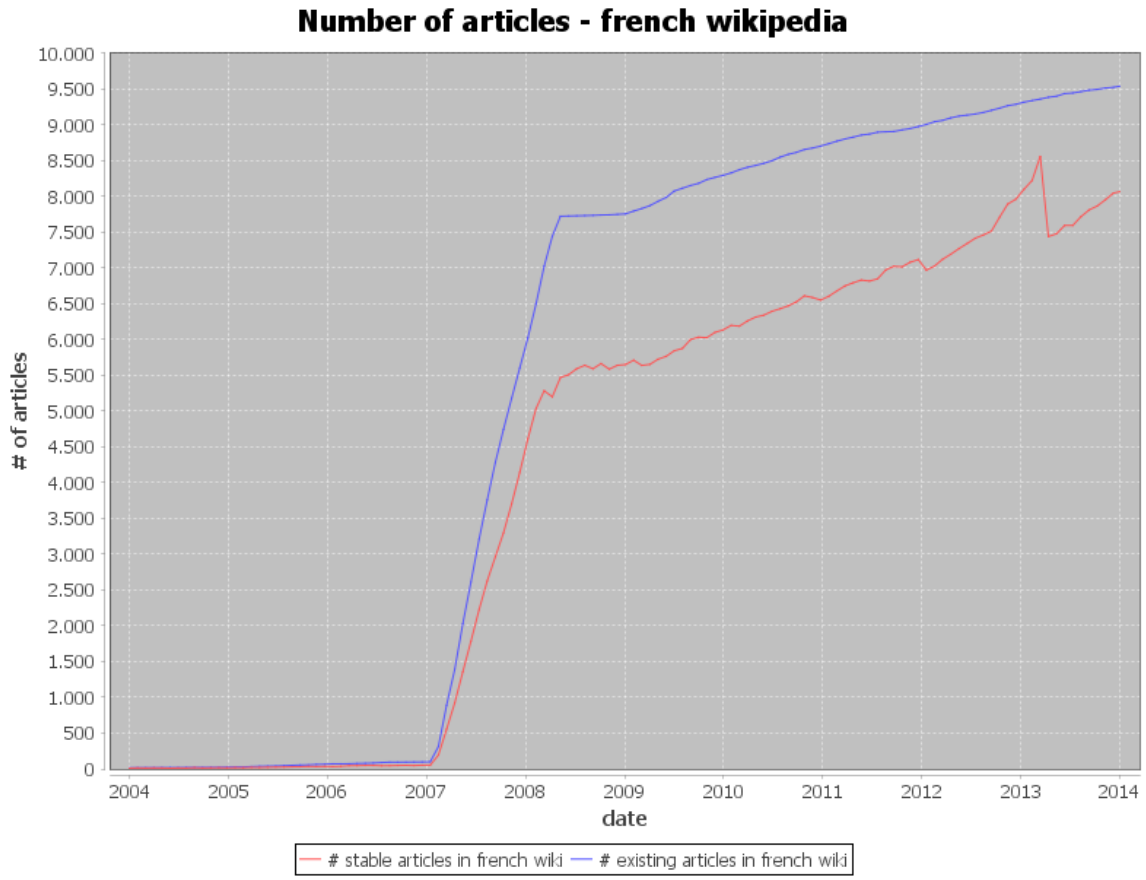


Figure 4.18: The stabilization process of the French Wikipedia edition.

Figure 4.4 illustrates the process of the semantic stabilisation of the French Wikipedia edition. The pattern is the same as in English Wikipedia observation - the blue and red lines are appearing in the form of *stairs* too. The common property of all experiments up to this point is the characteristic increase of stability around the beginning of the year 2014. Just as in the previous analysis, a significant instability of the articles available in the used sample is present. Another interesting fact is the time of the expansion in the total number of articles existing in this Wikipedia edition. The increase of the number of articles happens at the beginning of the year 2008; thus marking this time period as the period of the increased activity in this Wikipedia project and the booming of its community contribution. The French Wikipedia has about 1.7 million articles and is the third biggest in this group of experiments, and in general.

German Wikipedia experiment

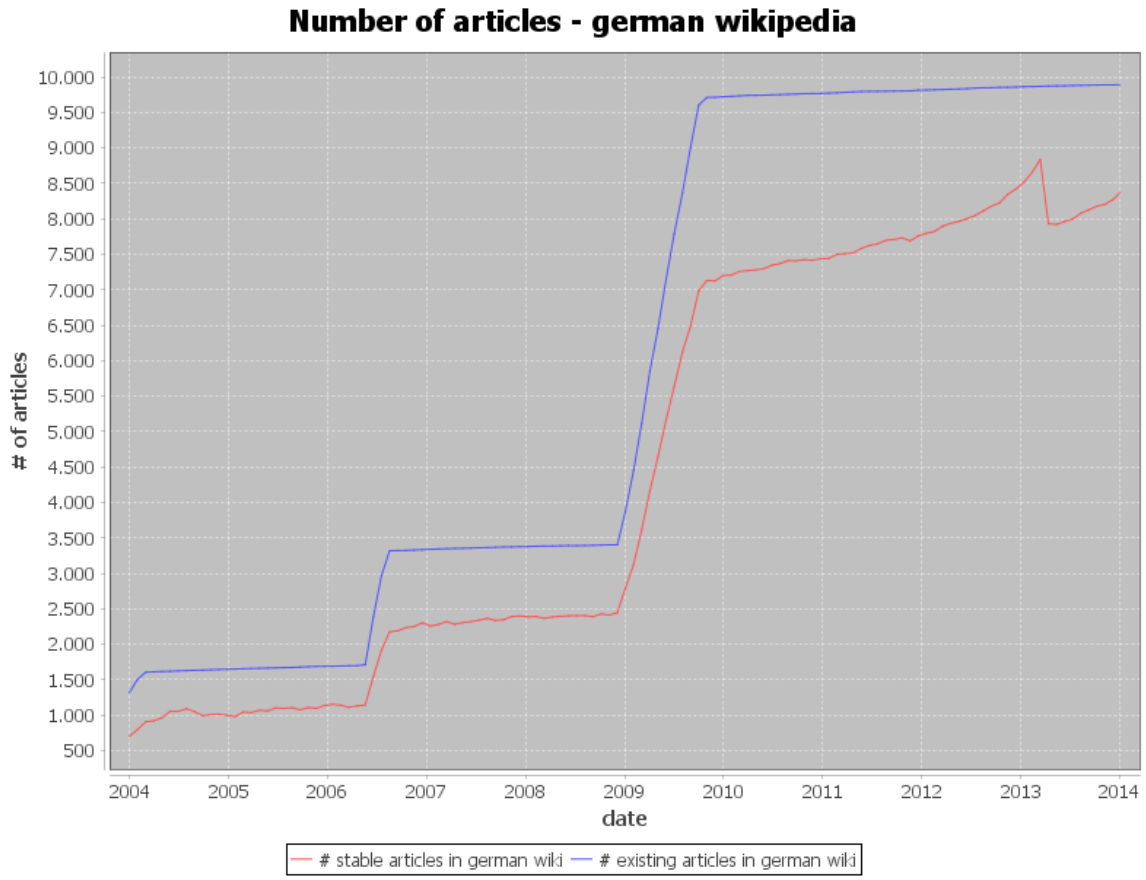


Figure 4.19: The stabilization process of the German Wikipedia edition.

The German Wikipedia edition is the second largest with the total of 1.8 million articles available. When one looks at figure 4.19 there is nothing that separates it from the previous two experiments from section 4.4 where the results of the experiments over the large Wikipedia editions. What is common to all of them is very low semantic stability value when compared to the stabilities shown in section 4.3.

Italian Wikipedia experiment

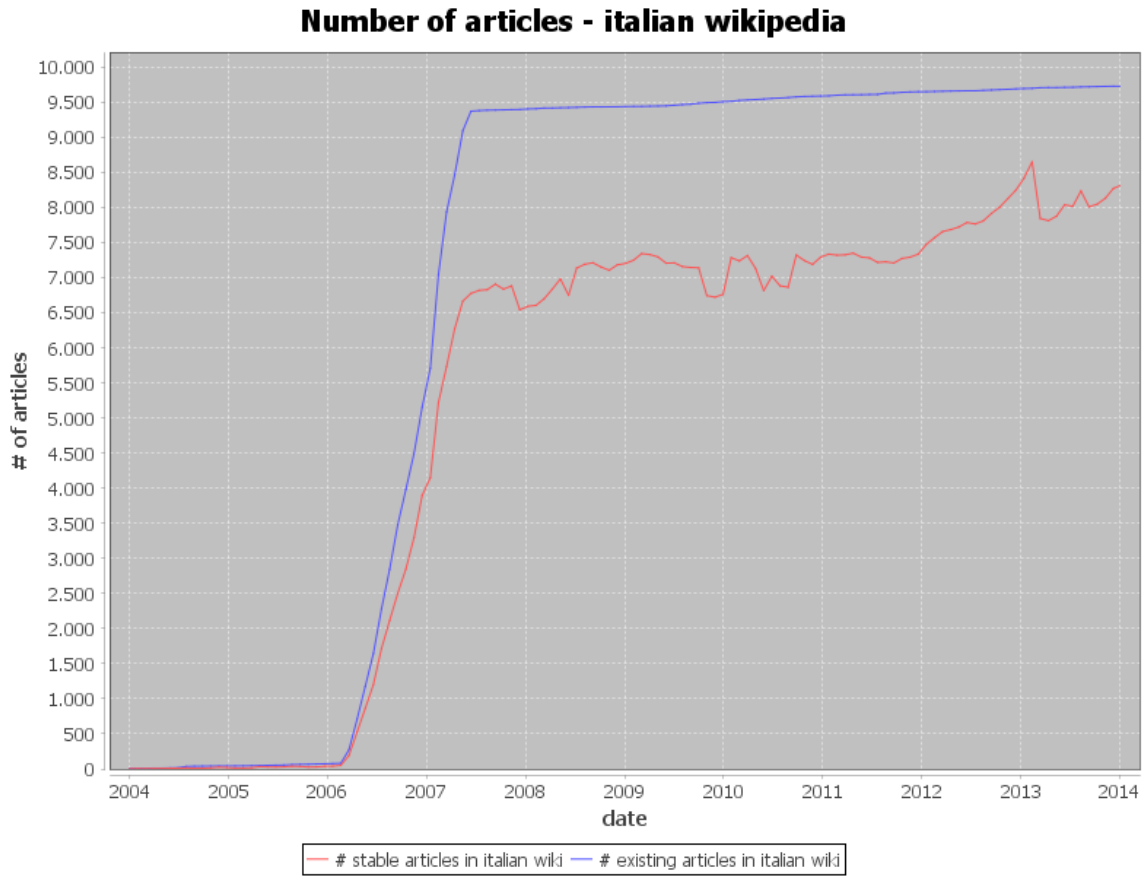


Figure 4.20: The stabilization process of the Italian Wikipedia edition.

Italian Wikipedia is the second to the smallest Wikipedia edition examined for the purposes of this thesis. It has almost 1.25 million articles. The expansion of the number of articles started in early 2005. The stability of the articles analysed from the sample corpus is relatively high until the end of the year 2007. With the development and the growth of the Italian community on Wikipedia, the level of semantic stability decreased significantly. The reason for this is a high surplus in newly created articles and a large number of revisions made to the existing articles from the corpus. All these observations can be identified in figure 4.20.

Spanish Wikipedia experiment

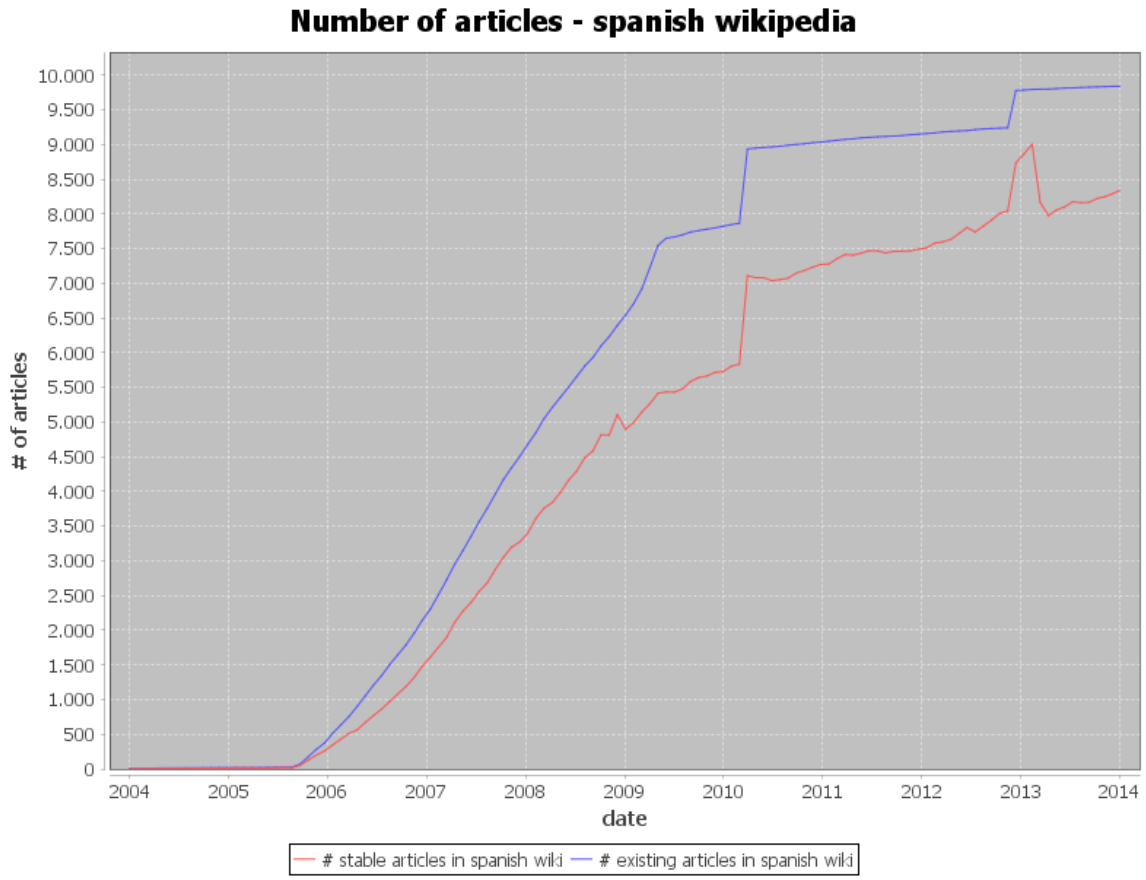


Figure 4.21: The stabilization process of the Spanish Wikipedia edition.

The smallest Wikipedia edition among the subset of the large Wikipedia editions analysed in this chapter is the Spanish Wikipedia. This edition's stability characteristics are similar to the results of the stabilisation analysis for other Wikipedia editions in the group of large Wikipedia editions and it can be observed in figure 4.21. The Spanish Wikipedia edition has 1.21 million articles. It is important to say that this is another experiment from the series where a sudden increase of stability can be noticed at the beginning of year 2013. In this experiment it is also a lot more visible than in English Wikipedia experiment, for example.

Stabilization process of large Wikipedia editions - Comparison

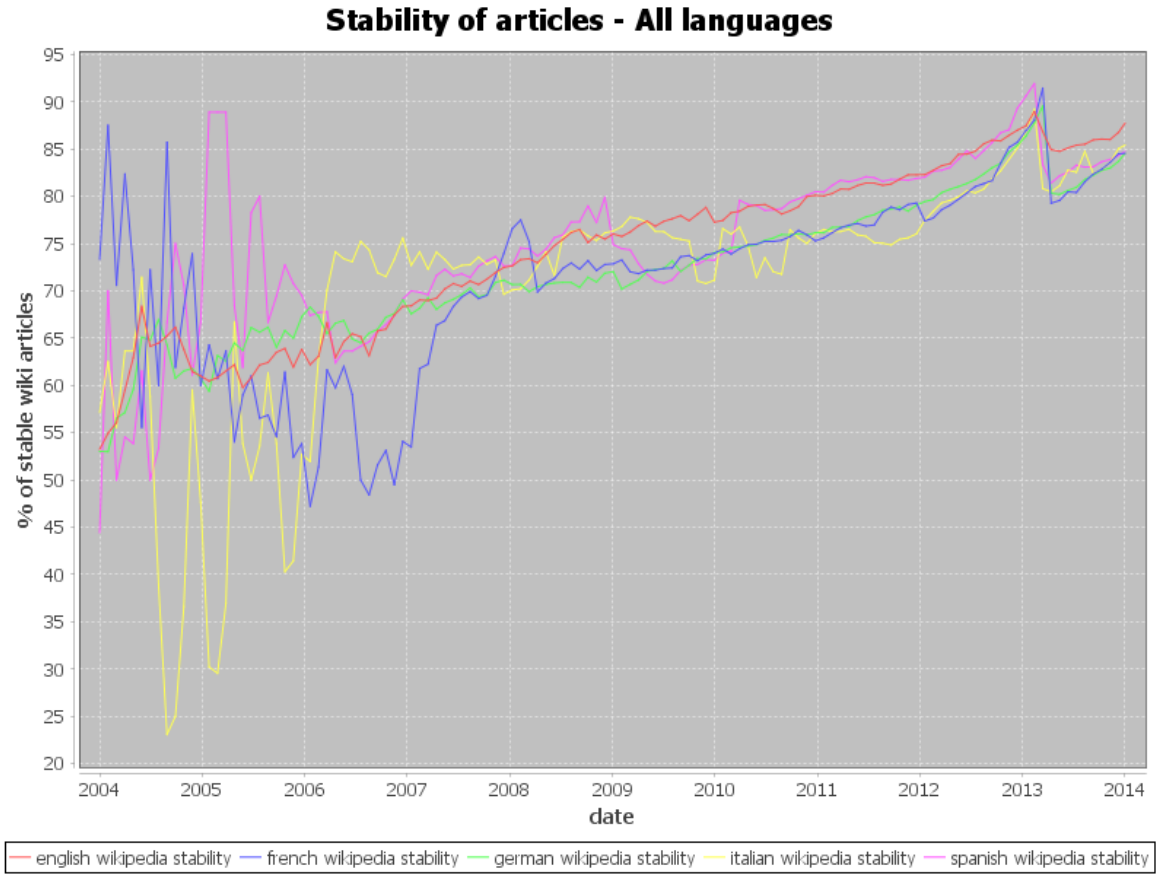


Figure 4.22: The percentage of the semantically stable articles in five largest Wikipedia editions.

Figure 4.22 shows the same situation as in section 4.4 with only one difference - now the semantic stabilities of the largest Wikipedia editions are compared. When compared to the values observed in 4.13 it is noticeable that the small Wikipedia editions are, on average 5%, more stable than the large Wikipedia editions. Although it is a fact that large Wikipedia editions have far more articles and article revisions than the small ones, it is also a fact that, in order to stay unstable, large Wikipedia editions also must maintain a high edit ratio of articles (a high percentage of articles have to be constantly and significantly changed). But it is also easy to understand how huge the communities working on the largest Wikipedia editions (like English, German or French) are compared to the communities improving the small Wikipedia editions written in languages used by a very small percent of the world population. Another important aspect is that the small Wikipedia editions contain large portions of articles simply translated from the English Wikipedia. Such articles are usually rarely changed substantially and they increase the overall stability discussed in section 4.3.

Stabilization process - 65%, 85% and 95% of articles from large Wikipedia corpus

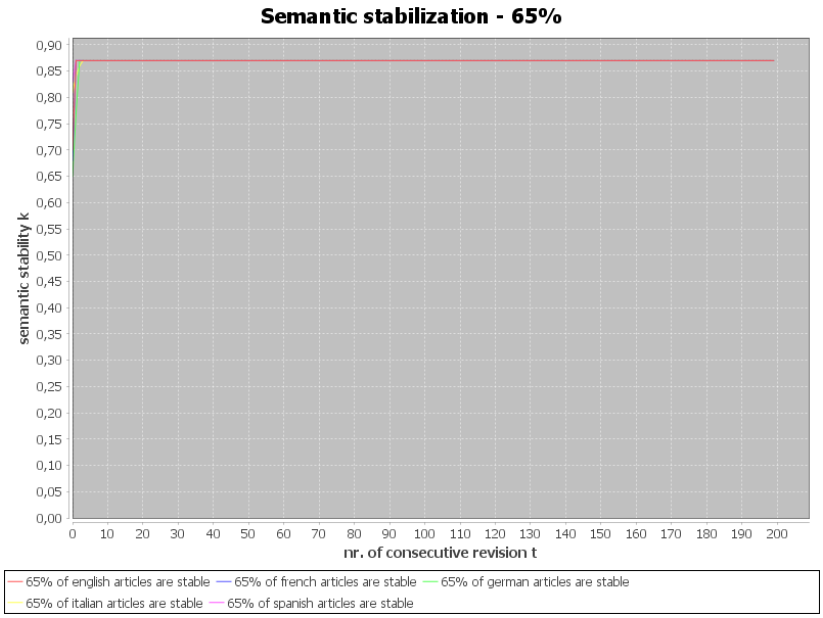


Figure 4.23: The number of revisions needed to achieve 65% semantic stability for stability threshold k.

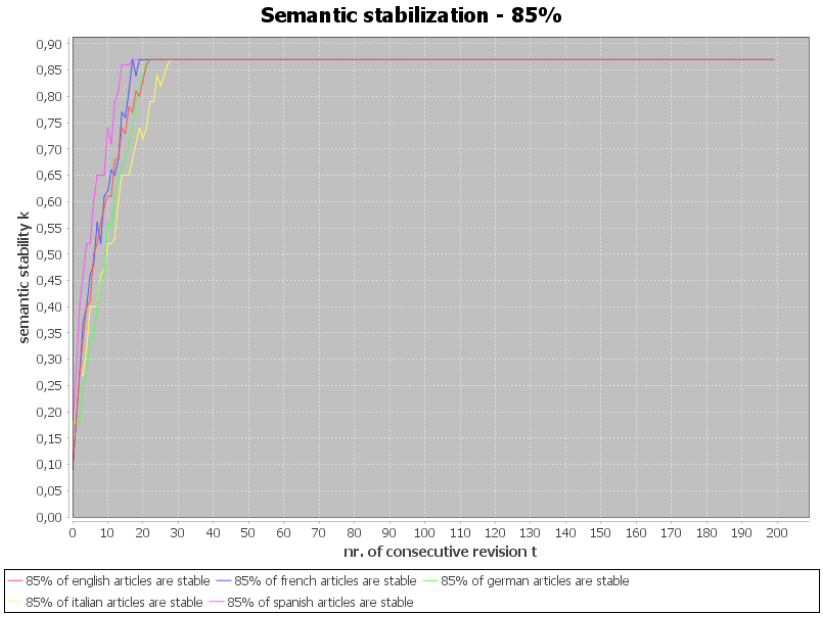


Figure 4.24: The number of revisions needed to achieve 85% semantic stability for stability threshold k.

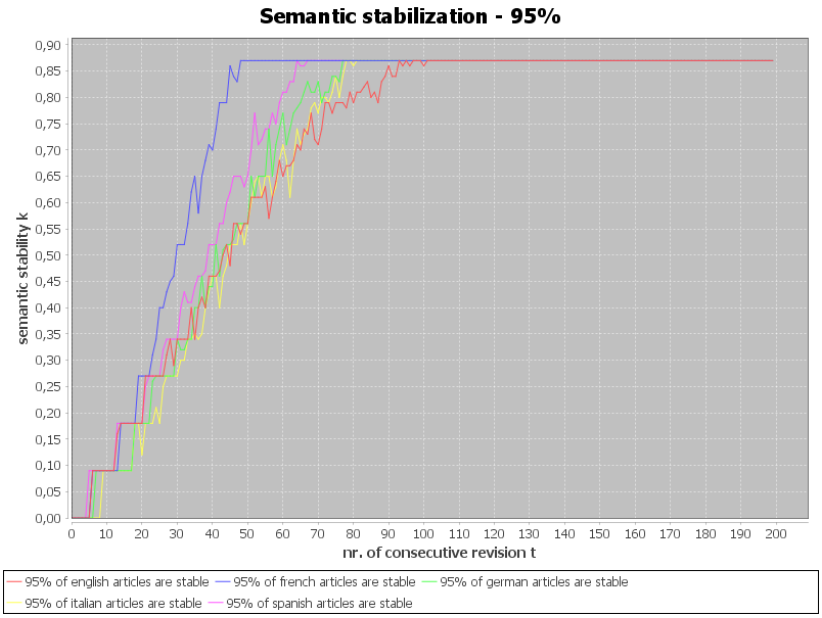


Figure 4.25: The number of revisions needed to achieve 95% semantic stability for stability threshold k.

In addition to the plots shown in section 4.4, which shows the stabilisation process of the large Wikipedia editions within a given time period, this section shows the comparison of the stabilisation process of large Wikipedia editions over the number of consecutive revisions. Figure 4.23 shows that each of the analysed Wikipedia editions have 65% of the stable articles in their corpus after as few as 3 revisions (or less) for the value of the parameter $k \geq 0.7$. This fact provides a better insight into the structure of the articles in Wikipedia editions. It can be concluded that a huge portion of the available articles in every single Wikipedia edition is stable with respect to the parameter k. Figure 4.24 depicts the results of the same experiment, but this time, 85% of the articles have to be stable. And now, it is possible to differentiate between the analysed Wikipedia editions stabilisation characteristics. The most semantically stable edition is the Spanish one, whereas the Italian Wikipedia edition needs the highest number of consecutive revisions until the stability of 85% of articles is achieved. Both of the editions need less than 30 revisions until the wanted stability is reached. The last figure in this section, figure 4.24 shows the stabilisation process of large Wikipedia editions where the achieved stability is 95%. This time, as expected, the English Wikipedia edition is the most unstable one. Almost the complete corpus of analysed articles becomes stable after almost 100 revisions of each article. The medium semantic stability of the corpus that is defined by the value of parameter $k = 0.6$ is, in the case of English Wikipedia, reached after about 60 revisions, and in the case of the French one (the most stable one) after about 35 revisions.

4.5 Stabilization process of English Wikipedia for different values of RBO parameter p

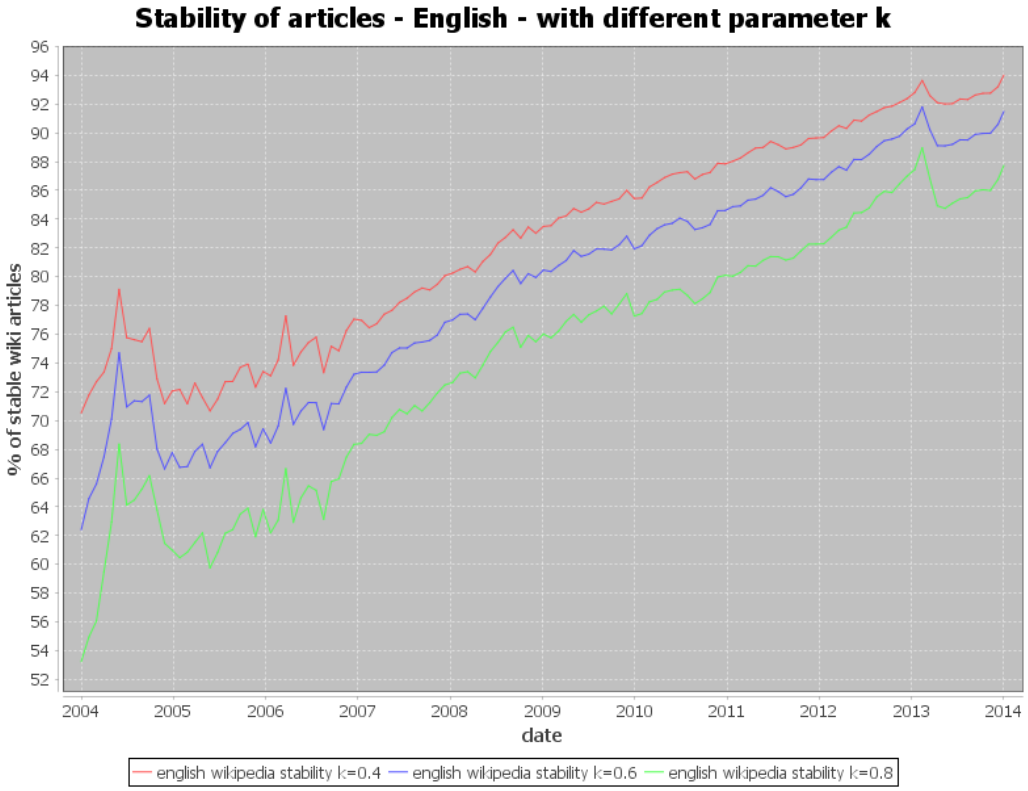


Figure 4.26: The stabilisation process of the English Wikipedia edition for different values of parameter k.

The last experiment in the series of experiments presented in chapter 4 is designed to show the role of parameter k in the stability calculation method proposed in [1]. Parameter k is the threshold value for similarity measure. Once the similarities of all revisions of a single Wikipedia article are calculated, the value representing the similarity in a given moment of time t is taken and compared to the arbitrary chosen value of the parameter k. The expectation of this experiment is to confirm the observation that, for a low value of parameter k, there are a lot of articles in the corpus of examined documents whose stability value in a given instant of time is higher than the chosen threshold. Thus, as the value of the threshold is increasing, the number of stable articles is decreasing. Figure 4.26 confirms this observation. The three lines depict the stabilisation process of English Wikipedia for the values of parameter k: 0.4, 0.6 and 0.8 respectively. The document corpus stability is inversely proportional to the value of parameter k.

4.6 Discussion of the experimental results

The goal of this chapter was to present the results attained from the proposed software tool when applied on a series of Wikipedia datasets. From the experimental results presented in this chapter, one can make two important observations:

1. all the analysed Wikipedia language editions show medium semantic stability
2. the mean semantic stability value of the Wikipedia language editions from the group of large Wikipedia editions is shown to be significantly lower than the mean semantic stability value of the small Wikipedia editions.

The authors of the paper [1] state that the natural languages are semantically stable in their nature. The first observation, stated in this section can be seen as proof for this statement. All the analysed datasets have at least medium semantic stability.

The second observation is the fact that large Wikipedia editions analysed for the purpose of this paper show less semantic stability than the small ones. This observation can be logically explained by the fact that large Wikipedia editions have much more contributors than the small ones. The sheer size of the community supporting and developing the English Wikipedia edition can not be compared to the size of community working on the Czech Wikipedia edition, which was also analysed for the purpose of this thesis. Having many more users contributing to the content means that higher semantic instability is brought to the system. The users of English Wikipedia are changing the content of the articles much more than the users of small Wikipedia editions. Another fact is that many articles available in small Wikipedia editions are simply translations of the articles found in English Wikipedia. Once translated, such articles are rarely significantly changed, which contributes to a higher semantic stability of the small Wikipedia editions. When figures 4.25 and 4.16 are compared, one can notice that, on average, a large Wikipedia dataset needs about 85 successive revisions until the high semantic stability level is reached. On the other side, the small datasets need only 35 revisions on average before they become highly stable.

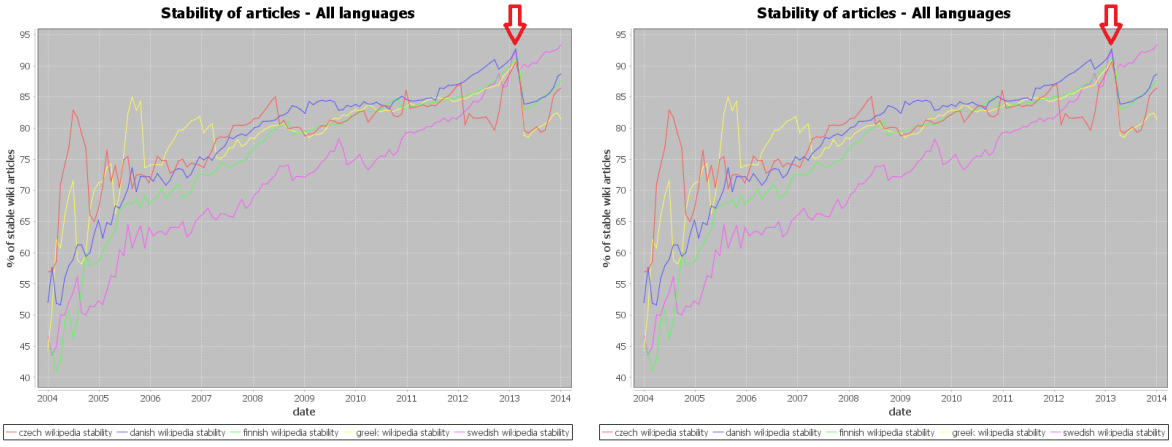
The experiment shown in section 4.5 was designed to investigate the influence of the semantic stability threshold parameter k used in the rank based stability method presented in [1] and implemented in the proposed software tool. As expected, the curve describing the semantic stabilisation process for different values of parameter k stays the same and it is only shifted upwards or downwards with respect to the value of the threshold parameter.

5 Discussion

The goal of this master thesis was twofold:

- to design and develop a software tool able to calculate and present the semantic stabilisation process of the natural language dataset by using *rank-biased overlap* method for the semantic similarity calculation and *rank-based stability* method proposed in [1] for the semantic stability calculation.
- to apply the proposed software tool on different Wikipedia language editions and to discuss the obtained experimental results. The correlation between the dataset size and the semantic stability of the dataset had to be investigated.

Both of the outlined goals are reached and the solution and the experimental results are presented in chapters 3 and 4. The theory behind it is discussed in chapter 2. As one of the



(a) The semantic stabilisation process of the small Wikipedia editions.

(b) The semantic stabilisation process of the large Wikipedia editions.

Figure 5.1: An interesting event at the beginning of the year 2013; a sudden increase in stability is noticeable in all Wikipedia editions and marked with rad arrows.

goals was to implement the software tool able to analyse Wikipedia datasets which are usually extremely large, the efficiency of the proposed software was really important. Although the indexing was done only on a relatively small sample of Wikipedia articles, the concurrency of Java had to be utilized for optimal results. When it is clear that the normal PC is nowhere

near the hardware requirements for this task and the same PC was the only hardware resource available, the need for optimized solution is even greater. The parallelism of the proposed software tool is also explained in detail in previous sections.

When all the available datasets were analysed and the curves representing the process of the semantic stabilisation were plotted, an interesting feature was revealed. Figure 5.1 shows the semantic stabilisation process of both groups of examined Wikipedia editions (small and large). Red arrows in both sub-figures (a) and (b) mark the event of interest. The sudden increase of the semantic stability could not be explained. The author of this thesis wrote several posts in the *Wikimedia.org*[1] mailing list, but no plausible answer was given. Some of the assumptions are that:

- some of the Wikipedia servers were down for a short maintenance
- some of the Wikipedia maintenance bots were active and editing Wikipedia contents was shortly blocked
- malfunctioning of Wikipedia servers was induced by malicious software or hacker attacks and so on.

Still, no hard evidence was brought into light.

Furthermore the limitations of the proposed software tool are clear. If one wants to analyse the complete dataset of any of the larger Wikipedia editions, some modifications of the software are required and that will be explained in chapter 6. One more thing worth mentioning is the shape of all the curves representing the semantic stabilization process. All the curves have more or less the same steepness (elevation), meaning that the content of all Wikipedia editions is modified in a continuous and relatively uniform manner.

[1] `https://lists.wikimedia.org/mailman/listinfo/wiki-research-l`, 07.01.2016

6 Conclusion and future work

The author of this thesis proposed a software solution which is able to calculate the semantic stability of the corpus of documents written in natural language. All the modules of the proposed software are explained in detail and the concepts used to develop this piece of software are easy to understand and can be further developed to achieve some other goals that will be mentioned in this chapter.

The proposed software was applied on 10 different Wikipedia editions (different languages) and the attained results are also presented and discussed in detail.

The two aforementioned statements imply that the biggest scientific contribution made by this thesis is twofold:

- a contribution through the practical implementation of the needed software tool and
- an empirical contribution made by carrying out the experiments and providing and discussing the results of these experiments.

It can be said that this thesis can help anyone looking for an efficient way to analyse the semantics of natural language documents contained in the Wikipedia XML dump files. The proposed software is highly modular, configurable and flexible as shown in previous chapters of this thesis.

Should the tool be used to index and analyse the full English or any other Wikipedia edition from the group of large Wikipedia editions, the proposed software might require some additional work. First of all, the hardware requirements will be much higher, so in the case of full indexing of English Wikipedia, a standard PC would have to be replaced with a moderate-size computational cluster with much more storage space which is available on drives able to provide a parallel I/O to its content.

When the new hardware requirements are met, module 1 of the proposed software tool 3.1, which is responsible for reading and parsing the Wikipedia XML dump files, has to be rewritten. Currently, when several partial XML files of a single Wikipedia dump are given, proposed software utilizes Java concurrency capabilities to read those files in parallel. If a cluster was available, a much better solution would be to use the Hadoop® capabilities. Namely, Hadoop® provides its own distributed file system. If one takes the XML dump files and copies them to the Hadoop® distributed file system, Hadoop® will implicitly use the concurrent data I/O, which it is designated and optimized for. This task would not be

trivial, as the configuration and running of Hadoop® is a complex task. Hence, further literature research would be needed, but it would not be in vain as it is probably the only way possible to accomplish full indexing of English Wikipedia in a reasonable time with reasonable resources. All the other modules declared in 3.1 are already optimized as they use the Mahout® library which is very scalable and also optimized to work together with Hadoop®.

The second limitation would be the choice of the database used by module 4 of the proposed software; the one responsible for storing the article data into a database. The English Wikipedia currently has over 5 million articles and the average number of revisions per article for English Wikipedia is almost 22 at the moment[1]. . This means that the database table will have over 110 million revision entries and the performances of the currently used MySQL® will be under serious pressure at that point. Under the assumption that the cluster will be used to host the database server, a possible solution, would be to use some of the distributed databases as Apache Cassandra, for example. Anyway, further research into the topic of distributed computing will be needed in order to solve such complex task like the semantic analysis of the full English Wikipedia index.

[1] `https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia`, 07.01.2016

List of Symbols

$\cos\theta$	the angle between two vectors
A, B	vectors containing term weights
$\|\|A\|\|$	magnitude of the vector
A_i	i-th element of the vector A
$tf(t,d)$	the term frequency of term t in document d
$f_{t,d}$	the number of occurrences of term t in document d
$idf(t,D)$	the inverse document frequency of term t in the document corpus D
N_D	the total number of documents in document corpus
n_i	the number of documents from corpus that contain the needed term
S, T	two indefinite rankings
S_i, T_i	i-th elements of the ranked lists S and T
$S_{c:d}$	a set of elements from the indefinite list S beginning from index c up to index d
$S_{:c}$	a set of elements from the indefinite list S beginning from index 1 up to index d
$S_{d:}$	a set of elements from the indefinite list S beginning from index d up to index ∞
$I_{S,T,d}$	an intersection of lists S and T at depth d
$\cap$	an intersection operator
$X_{S,T,d}$	an overlap - the length of the intersection of the lists
$\|A\|$	the length of the vector A
$A_{S,T,d}$	the *agreement* of vectors S and T at the depth d
$AO(S,T,k)$	an average overlap of the lists S and T for maximal depth k
$SIM(S,T,k)$	similarity
$RBO(S,T,k)$	rank-biased overlap
σ	different notation for an indefinite ranking

List of Abbreviations

API	Application Programming Interface
TF-IDF	Term Frequency - Inverse Document Frequency
XML	Extensible Markup Language
ORM	Object-relational mapping
RBO	Rank Biased Overlap
TF	Term Frequency
IDF	Inverse Document Frequency
VSM	Vector Space Model
TDM	Term Document Matrix
DVD	Digital Video Disc
7z	7zip archive format extension
bz2	bz2 archive format extension
JDK	Java Development Kit
JRE	Java Runtime Environment
NLP	Natural Language Processing
SSD	Solid State Drive
IDE	Integrated Development Environment
DOM	Document Object Model
SAX	Simple API for XML
UML	Unified Modelling Language
RegEx	Regular Expression
I/O	Input/Output
HDD	Hard Disc Drive

FJF	Fork Join Framework
DWPT	Document Writer Per Thread
JPA	Java Persistence API
ORM	Object-Relational Mapping
EJB	Enterprise Java Beans
PC	Personal Computer

Bibliography

[1] Markus Strohmeier Claudia Wagner, Philip Singer and Bernardo A. Huberman. Semantic stability in social tagging streams. In *WWW'14 Proceedings of the 23rd international conference on World wide web*, pages 735–746, Seoul, Korea, April 2014.

[2] Franca Debole and Fabrizio Sebastiani. Supervised term weighting for automated text categorization. In *SAC '03 Proceedings of the 2003 ACM symposium on Applied computing*, pages 784–788, New York, USA, 2003.

[3] Ralph Johnson Erich Gamma, Richard Helm and John Vlissides. *Design Patterns: Elements of reusable object-oriented software*. Addison-Wesley publishing, 1994.

[4] Amir Gandomi and Murtaza Haider. Beyond the hype: Big data concepts, methods and analytics. *International Journal of Information Management*, 35:5137 – 144, April 2015.

[5] Javier Fernandez Gonzalez. *Java 7 Concurrency Cookbook*. Packt Publishing, 2012.

[6] Chunyu Kit Jonathan J. Webster. Tokenization as the initial phase in nlp. In *COLING '92 Proceedings of the 14th conference on Computational linguistics - Volume 4*, pages 1106–1110, 1992.

[7] Mike Keith and Merrick Schincariol. *Pro JPA 2: Mastering the Java Persistence API*. Apress, 2013.

[8] Manu Konchady. *Building Search Applications: Lucene, Lingpipe, and Gate*. Mustru Pub, May 2008.

[9] Magnus Sahlgren. An introduction to random indexing. In *Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering, TKE 2005*, Copenhagen, Denmark, August 2005.

[10] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management: an International Journal*, 24:513 – 523, 1988.

[11] Ted Dunning Sean Owen, Robin Anil and Ellen Driedman. *Mahout in Action*. Manning Publications, 2011.

[12] WikipediaProject. Wikipedia:About. `https://en.wikipedia.org/wiki/Wikipedia:About`, 2015. [Online; accessed 24.12.2015].

[13] Justin Zobel William Webber, Alister Moffat. A similarity measure for indefinite rankings. *ACM Transactions on Information Systems (TOIS)*, 28, November 2010.