Radboud University Nijmegen

Faculty of Science
Institute of Computing and Information Science

Master Thesis

A tool for click data preprocessing

by

Marc Pieterse

mpieterse@gmail.com

Supervisor: Prof. Dr. Ir. Th. P. van der Weide
Second Reader: Dr. P. van Bommel

Nijmegen, Version date: 21 June, 2010
Graduation number : 124

In memory of Hans Pieterse

Preface

This process for me, graduating, especially in the beginning was like the great journey of a rocket leaving the Earth's atmosphere. Uncertain whether the launch would succeed. Uncertain about its chosen trajectory and uncertain about reaching its destiny: space. After a bumpy launch, the project finally kicked off.

The further the rocket got, the more obstacles it had to face, so during the journey, there were times when I thought: "Houston, we have a problem!!". Luckily for me, I had a very stable mission control that stayed very calm and flexible. This person for me was Prof. Dr. Ir. Theo van der Weide and I hereby would like to express my gratitude for his patience, advice and support during the last two years. Not only for my supervision, but also for all his work regarding international cooperation that allowed so many students (myself included) to experience ICT in different cultures.

Eventually the rocket left the Earth's atmosphere. Thereby escaping from its gravitational field: finding itself free to choose a new destination.

- Marc Pieterse

Special thanks go out to:

- Prof. Dr. Ir. Th.P. van der Weide
- Dr. P. van Bommel

- M. Hermsen
- M. Hinne
- T. Leijten
- J. Schepens
- I. Schreurs
- P. Tenbült

- Family and Friends

Abstract

Keywords: *IR, click data, Singular Value Decomposition, preprocessing, filtering, large matrices, sparse matrices, numerical stability, SVDLIBC*

This Master Thesis for Information Science describes the design and implementation of a tool that performs Information Retrieval (IR) techniques on click data. Click data can be seen as a log file from a search engine that provides information about the search behavior of its users. By interpreting the log file, it's contents can be used as a source of relevance feedback by which search engines can improve the search results given a certain query.

The goal was to expose hidden underlying structures (concepts) in the 'query terms – URL' relation represented in the click data, by creating an association matrix and eventually computing its Singular Value Decomposition (SVD). Computing the SVD of a matrix itself is nothing new. However since the matrix is of enormous proportions with dimensions in the order of millions, it still poses a problem for modern computers to compute it's SVD. The tool proposed in this thesis especially aims at preprocessing large click data sets, so they become more manageable by third party applications.

The program is modeled around a generic data analysis process. It's architecture consists of a modular object oriented approach, so that expanding functionality or cooperative development becomes more easy. The idea is that it could combine all kinds of IR techniques in the future, creating a workbench especially aimed at preprocessing (click) data sets. However the current implemented features are aimed at completing the specific task of analyzing the 'query terms – URL' relation. The tool was written in C# and compiled to run on the MS Windows 64 bit platform as a console application. That itself is quite unusual since most IR tools are designed for the UNIX platform. Therefore during implementation, the chosen language and platform caused several obstacles, like memory allocation and the portability of a third party SVD package. The implementation also focused on numerical stability, since rounding errors in the application could cause the results to become invalid.

Eventually, several filters have been implemented in the tool and their performance has been tested. Using a combination of URL stemming, navigational noise filtering, lowercase conversion, and special character removal, an association matrix with dimensions of 1.8M times 4.8 million was reduced to $\frac{1}{5}$th of its original size, while trying to maintain the semantic relations as much as possible. Also the density of the matrix was increased three times. The tool's performance shows a linear consistency in terms of execution time versus the amount of data it has to handle increases.

Future work could be to explore the possibility of reusing data that was removed using the URL stemming filter, to improve domain specific navigation. A major improvement for the tool, would be to have an SVD algorithm incorporated aimed at large and sparse matrices. To speed up the performance of calculating the SVD, this algorithm should be able to run on the computer's graphics card (GPU) for example using the CUDA Framework.

Contents

1 Introduction **3**
1.1 Problem Description . . . 4
1.1.1 Choices and assumptions . . . 4
1.2 Method . . . 5
1.3 Overview . . . 6

2 Background **7**
2.1 Click data . . . 7
2.1.1 Elements of a click data set . . . 8
2.1.2 Relations within a click data set . . . 8
2.1.3 Selecting a relation for further study . . . 10
2.2 The vector space model . . . 11
2.3 Singular Value Decomposition . . . 12
2.4 Memory Management . . . 13
2.5 Numerical stability . . . 14
2.5.1 Numerical stability in C Sharp . . . 14
2.5.2 Numerical stability in SVD . . . 15
2.6 State of the Art . . . 16
2.7 Summary . . . 17

3 Introduction of an IR Data preprocessing tool **19**
3.1 Requirements . . . 19
3.2 Data Analysis Process and Information Retrieval . . . 20
3.3 From plain text to an Association Matrix . . . 20
3.4 Domain specific approach . . . 24
3.5 From Association Matrix to concepts . . . 25

4 Implementation **27**
4.1 Design decisions . . . 27
4.2 Segmentation using three-tier . . . 29

4.3 Design patterns . . . 31
4.4 Data structures . . . 32
4.4.1 Hybrid Arrays . . . 32
4.4.2 Sparse Matrix Structure . . . 33
4.5 Filtering the data . . . 35
4.5.1 Focusing on individual query terms . . . 36
4.5.2 Focusing on individual URLs . . . 36
4.5.3 Focusing on the relationship between Terms and URLs . . . 37
4.6 Implementation of an SVD algorithm . . . 38
4.6.1 Testing the accuracy . . . 39
4.6.2 Specifications of several test matrices . . . 40
4.6.3 Verification of implementation . . . 42
4.7 Validation of the tool . . . 43
4.8 Evaluation . . . 44
4.8.1 Future work . . . 44

5 Experiments 45
5.1 The experimental setup . . . 45
5.1.1 Data Set . . . 45
5.1.2 Environment . . . 46
5.1.3 Variables . . . 46
5.2 Experiment 1: Data Cleaning Module VS. the size of the association matrix . . . 47
5.3 Experiment 2: Data Cleaning Module VS. the environment variables 48
5.3.1 Execution time . . . 48
5.3.2 RAM usage . . . 49
5.4 Experiment 3: Execution time VS. the number of records . . . 50
5.5 Results . . . 50

6 Conclusions 53
6.1 Answering the research questions . . . 53
6.2 Future work . . . 54

Bibliography 55

A Data tables 61
A.1 Data from the experiments . . . 61
A.2 Parameters of SVDLIBC . . . 64
A.3 Example Log file . . . 65

List of Figures 67

List of Tables 68

Chapter 1

Introduction

Search engines try to satisfy the users information requirements based on a given search string (query). A query generally consists of several words or terms, that the user provides as input to the search engine. The user then is presented with a list of URLs of which the system believes that they meet the requirements of the user. The scientific area that is concerned with finding optimal matches between what a user requests and the system provides, is called Information Retrieval (IR) . To generate a simple result list, lots of different IR techniques are required. In the background, search engines have to learn what users need and therefore what they mean, or what their understanding of the world is. Many machine learning techniques involve analyzing data and user behavior to formulate so called concepts. The search engine can then use these learned concepts to better serve the users information need, since it can relate better to the question.

Jung et al. use a special tool called SERF to collect data on the user's search behavior [15]. This is done by logging the user's actions while searching the World Wide Web with the tool. They describe how it's collected 'click data' can be used to improve the quality of search results by using it as implicit relevance feedback. Also tools such as WebMate interact with the user and try to apply relevance feedback on the fly [18]. However, their approach only allows the current user to benefit from it while Jung et al. are specially interested in improving the relevance for all users.

Another form of click data that can be used as relevance feedback are log files from search engines. Hinne et al. use the 'query term – URL' relation inside the click data to improve the document descriptions for Wikipedia pages [13]. What is interesting is to look deeper into the semantic relations within click data and examine what kind of other information could be extracted from them. For their experiments they also apply several IR techniques like TF–IDF weighting to their data, before further analysis can take place.

1.1 Problem Description

Usually association matrices derived form such log files have enormous dimensions while they only contain very little of actual data. Depending on the type of analysis, the matrix dimensions could have an enormous impact on the execution time. Especially in the case of Singular Value Decomposition (SVD) which traditionally is of order $O(M \cdot N^2)$. [19, pp. 135] So shrinking the association matrix in a responsible way, would go go a long way in improving this.

Currently there are some scientific tools available that allow to apply IR techniques in a simple to use way on different data sets, like Lemur*. However, most tools are single scripts, based on UNIX and therefore do not care for any form of GUI. For experts this is fine but for people who are less familiar with IR and do need to apply the techniques (students); a tool that incorporates different IR techniques combined with a process that allows to apply these techniques in an easy way would be helpful. For this reason and also to familiarize the author with IR techniques, a custom data analysis tool will be developed. This tool will be based on the analysis process defined in [6] and will mainly focus on preprocessing of the data. To create flexibility, the tool will be separated into several modules that each fulfills a specific role. This will also allow for easy implementation of new IR techniques and communication with third party applications.

1.1.1 Choices and assumptions

This is an Master Thesis for Information Science. Even though the research has a strong focus in the areas of Informatics and Mathematics, Information Science is the viewpoint from which both areas are observed.

In this work, the gap between theory and practice is bridged. In the authors opinion, this is a strong aspect of his work, because the entire process from determining the problem to formulating a solution and implementing it has been walked though. The familiar phrase: *"In theory there's no difference between theory and practice, in practice there is."* best describes some of the most challenging aspects that where faced in writing this thesis.

*http://www.lemurproject.org/

1.2 Method

In this section the research method is further explained. Based on the problem description, a research question has been formulated:

What information is hidden inside click data?

Since this question alone is to global to answer, it has been divided into several sub questions. These have helped to steer the research in the direction that it went.

1. In what ways can click data relations be perceived?
2. What kind of techniques are required to extract the necessary information from the data set?
3. How can these techniques be applied to the data set?

Below, each of the sub questions is explained in more detail.

1. **In what ways can click data relations be perceived?** Answering this question can give insight in how this data could help to make computers understand a bit (or a few bits and bytes) about our universe. This will be done by studying the different attributes in the click data and its underlying relations. After that one of the relations will be selected for further study.

2. **What kind of techniques are required to extract the necessary information from the data set?** To be able to able to use the information within one of those relations, scientifically accepted methods should be used to extract it. For that relevant literature has to be studied.

3. **How can these techniques be applied to the data set?** To extract the required information from the relation selected in sub question one, a special tool is required. This general-purpose tool should implement the methods that are the result of answering sub question 2. Based on this outcome, a tool should be formulated and implemented to retrieve the required information.

1.3 Overview

This thesis will be divided in several chapters. Chapter 2 outlines the background material used in this thesis. The first part will discuss the relations inside the click data. The second part will focus on IR techniques that are used to extract the underlying semantics within one of those relations. These IR techniques suffer in practice from several drawbacks which are also discussed in chapter two. This information is later used to coop with those problems. The last part of chapter two describes some recent developments, concerning the computation of the SVD in practice.

Chapter 3, describes the tool's requirements and behavior. The first part describes several requirements that account for the usability of the tool. The second part takes a global process of data analysis and specifies how this can be perceived in terms of IR. The third part is concerned with the preprocessing steps of the analysis process up to the creation of the association matrix. After that, the research briefly steps out in a small side area and describes an idea that can be explored in future research. In the last part, several linear algebra packages for C# are briefly evaluated and decisions are made whether or not to implement them in the tool, to make up for the last part of the analysis process.

Chapter 4 describes the actual architecture of the tool. It formulates design decisions that were made in the selection process of the programming language and operating environment. It also describes the specific data structures that are used to handle the large amount of data. Also the implementation of several preprocessing filters is described in more detail and suggestions are made for future filters. A big part of the work that has been put into this thesis went to the finding and having the tool work with a SVD algorithm. That is described in the last part of this chapter.

Chapter 5 describes several small experiments that have been performed to test the preprocessing filters and the overall performance of the tool. First the experimental setup is explained, then three experiments are performed and its results are discussed.

In the last chapter, the overall results of this research are discussed and possible future work is brought to attention.

Chapter 2

Background

This chapter provides background information about the context of the research. The first part explains what click data is, what kind of information it contains and how it can be useful. The second part highlights some recent developments in the area of Information Retrieval (IR), to put the research into context. The third part explains what kind of technologies are available to analyze the click data and the last part of this chapter is concerned with several important aspects of computer's limitations.

2.1 Click data

When the user enters a search query into a search engine, it retrieves a list of URLs of which the system believes that they meet his requirements. The user makes his choice and clicks on the link that seems the most relevant. This action is then stored by the search engine into a log file. A collection containing this kind of information engine is referred to as click data. So, click data is a collection of facts about users search behavior, logged into a file. To have an idea what such a log file looks like, a small sample of click data is shown in table 2.1.

Table 2.1: A sample of click data from the RFP data set

Hash	**Terms**	**Date & time**	**URL**	**Rank**
00003dae94c849ed	global star phones	2006-05-26 11:10:38	http://www.globalstar.com/	5
000026b651464092	www.certmail.com	2006-05-15 18:08:06	http://www.certmail.com/	1
00003e133e6c46a4	plastic culvert	2006-05-25 06:57:00	http://www.daltonrock.com/	1

2.1.1 Elements of a click data set

Before we go into details about the different relations, it is important to have an understanding of the elements that make up this click data. In this subsection each of the elements will be discussed briefly. In this case, the available data file consist of the following kind of elements: a session hash, query terms, a date & time, the related URL and the rank of the page in the search engine's result list. Each log entry (record) in the data file represents the action of one search event, which can be formalized as follows: $l =< h, q, t, u, r >$.

Session hash - The session hash is a unique identifier to a certain session from a certain user. The particular data set has been made anonymous due to privacy aspects. But to have some idea of what kind of searches originated from the same source (within a certain time frame) a hash has been added. Unfortunately the hash only applies to the same terms within the same session, so in this particular case its usage is rather limited.

Query terms - A query is a collection of terms entered in the search engine. These terms are individual words that together describe the type of information the user is looking for. It could consist of one word, several words or even complete sentences. Each query can be seen as a collection of terms separated by spaces so that: $Q = \{t_1, t_2..t_n\}$.

Date & time - This field contains the date and time notation at which the user performed the search action.

URL - This is the URL that was clicked on by the user after being presented with a result list. The URL itself can also be perceived as a collection of terms (or qualifiers) that could potentially describe some of the content it contains.

Rank - This is the position of the URL on the search engine's result list. The result list was displayed in such a way that the most promising results (according to the search engine) were displayed on top of the page in descending order. So the lower the rank attribute, the higher the URL's probable relevance.

2.1.2 Relations within a click data set

The attributes mentioned in the previous section combined, form different kind of relationships between objects (see figure 2.1). For example: the relation between query terms and the URL could describe which kind of information is concerned with a certain URL. In this subsection the various relations between two different attributes are described.

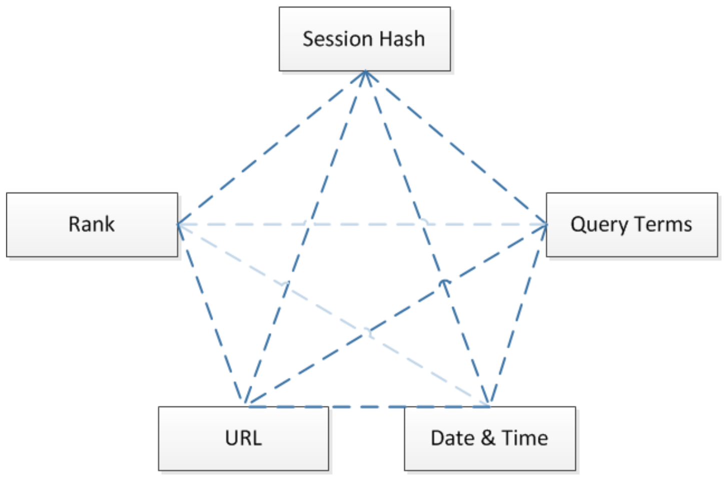

Figure 2.1: Possible relations inside click data

Query Terms – Date & time: This relation could contain information about whether some topics are searched for at certain times more than others. In that case it may proof interesting for advertising campaigns. Companies could use this information to show certain advertisements more often during specific times. This could increase the number of people viewing their site and potentially generate more revenue. This can be formalized as:

$$q(t) = \begin{cases} \mathit{True} & \text{if the click data contains } q \text{ within a certain time } t \\ \mathit{False} & \text{otherwise} \end{cases}$$

Query terms – URL: The relations between query terms and URLs describe an important relation within click data. It shows which keywords where used, that led the user to visiting this specific URL. Interesting questions would be whether those keywords are relevant for the URL or which concepts they form together.

Date & time – URL: Just like the Query Terms – Date & time relation, this relation could also indicate if certain sites are being visited within a specific time. However it would require the sites to first be classified into a certain category.

URL – Rank: The URLs frequency can be used together with the Rank attribute to describe the search engine's most popular URLs.

Session Hash – Query Terms: This relation could normally give insight in the kind of information the user was looking for at a certain time in different searches. Combined with the date & time attribute, it could even tell about the user's determination process.

The following relations that contain the hash attribute could promise to be very useful. However, with the current data set, this relationship cannot be exploited to its full potential since the hash does not cover one complete browsing session.

Session Hash – Date & Time: This says something about the search behavior of the user. What amount of time did he approximately spend on researching the subject?

Session Hash – URL: This data could be used to identify user patterns. Only if the session time is too short, the profiles won't be accurate. It could help to determine interest groups.

Session Hash – Rank: This relation can tell us something about user behavior. Do they constantly click on the top URLs or do they also try other results? It can tell something about whether other sites even get a chance when they are not in the top ten.

Some other relations could prove to be interesting, but they cannot be easily represented as a 2d matrix:

Query Terms – URL – Rank: When the rank of a URL clicked on deviates a lot from the first x positions, it could mean the search engine has a wrong view on the terms that are associated with the more popular URLs. Examining this relation could be used to correct wrongly associated terms.

2.1.3 Selecting a relation for further study

When studying most relations, the main issue remains: what query terms are related to the same topic? It is still hard for a computer to determine whether words relate to the common topic or not. That's why this research continues with the focus on the relation that can give meaning to concepts and look for methods to extract these concepts from one of those relations. Assumed is that the keywords that lead to a certain website, are somehow relevant for that website and therefore they are part of concepts that are described by that website. That is why the 'query terms – URL' relation seems the most interesting one and will be studied further in this thesis.

2.2 The vector space model

One of the first models developed for Information Retrieval was the Boolean model. It is based on Boolean algebra and set theory. It perceives each query and document as a set of terms and uses Boolean algebra to determine if a set of query terms (combined with logical operators) match certain documents. But a document consists of many terms of which some are more relevant than others. Using Boolean expressions only gives a true (relevant) or false (irrelevant) result. However, this is very inflexible since it does not allow room for error. It would be more reasonable if a document's relevancy could be ranked somewhere in between relevant and irrelevant.

This is where the vector space model comes in. Relations between terms and documents are represented as vectors in an association matrix [21]. Usually term vectors are represented as columns and document vectors as rows. The vector space model tries to interpret the cohesion between terms and documents. This is done by assigning scores to a term / document relation with a process called term weighting. It produces a score for each term / document association expressed as a decimal number usually between 0 and 1. In the case of click data it can also be used, when the URL is perceived as the document. There are different weighting functions, but one of the most popular ones is the Term Frequency – Inverse Document Frequency (TF–IDF), described in [5, pp. 117–119]. Term Weighting assumes several rules [23]:

- Terms that occur more often in a document are likely to be more important.
- A term that occurs in a few documents is more important than a term that occurs in every document.
- Large documents should be treated equally to smaller documents.

The formal definition of TF–IDF is:

- TF: The Term Frequency expresses the relevance of a term t towards a certain document d. First its frequency is determined, then its value is then scaled against all terms in that document.

$$tf_{t,d} = \frac{n_{t,d}}{\Sigma_k n_{k,d}}$$

- IDF: The Inverse Document Frequency helps to scale down terms that occur too often in the collection. It is calculated by taking the log of the total number of documents that is divided by the number of documents that term t occurs in.

$$idf_t = log \frac{N}{DF_t}$$

- Combined, TF–IDF is a function that assigns a weight to a term within a certain document that complies with the rules formulated above.

$$(TF - IDF)_{i,j} = tf_{i,j} \cdot idf_i$$

The relation between terms and documents are represented as vectors in a term/document matrix. Its cohesion is represented a TF–IDF weight, usually expressed as score between zero and one. The higher the score means the stronger the association. This type of matrix is known as an association matrix and is usually very sparse. An example of an association matrix can be found in table 4.1. These type of matrices usually contain lots of noise: associations that are there, but that are unstable in a different context. If these dimensions where to be filtered out somehow, it would reduce the matrix drastically and thereby leaving only the strong concepts.

2.3 Singular Value Decomposition

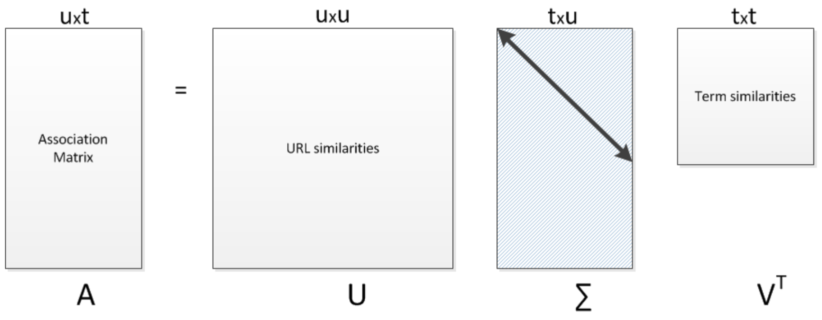

Figure 2.2: Singular Value Decomposition

The Singular Value Decomposition (SVD) (see figure 2.2) is a matrix decomposition technique that was introduced in 1965 by G. Golub and W. Kahan [9]. It can be used to transform the constructed association matrix into three matrices, so that:

$$A = U\Sigma V^T$$

The SVD is used to find a transformations for both input axes and output axes into a common system of variables. These variables will be called concepts. The SVD provides a transformation V of the conceptual space into the input space, and a transformation U from conceptual space into output space. In other words, a query is translated to a concept and that concept, is translated into URLs that most strongly represent that concept.

After the matrix is converted to the three matrices, the diagonal matrix Σ contains the concepts, represented as singular values (the square root of the non-negative Eigenvalues), which are sorted in decreasing order. This special Σ matrix can be used to filter out the meaningless concepts: the noise from the original matrix A [9, 24]. Usually at some point k in the diagonal there is a sharp drop of singular values. The small values can be interpreted as noise, and the corresponding concepts may be omitted (see figure 2.3).

This leaves a greatly reduced version of the three matrices. By multiplying them, the reduced matrix A_k is constructed. It still contains the most important aspects of the original association matrix A. This method can be used to uncover hidden similarity even if the query terms and URLs have no terms in common [24].

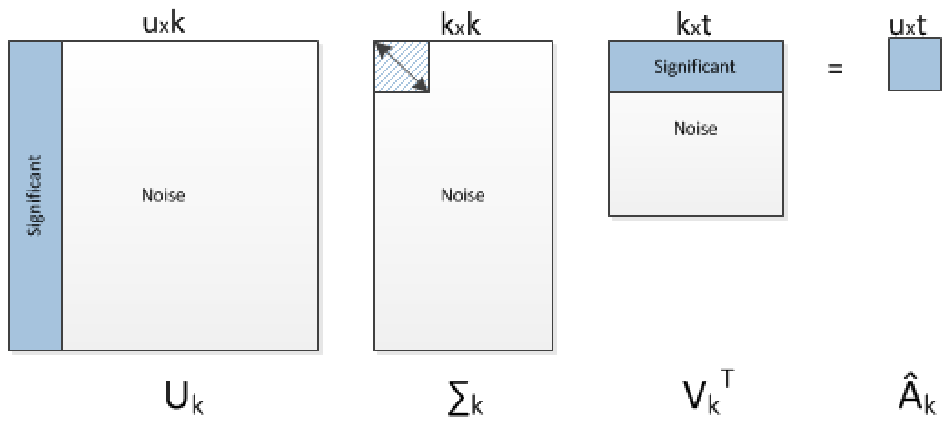

Figure 2.3: Reduced association matrix

Based on the type of matrix (size, density), a different approach in computing the SVD is chosen. For dense matrices, a straight forward approach is suggested while for large sparse matrices an iterative method is better suited, since it uses the sparseness to optimize the process [9]. Therefore an algorithm with an iterative approach should be chosen.

2.4 Memory Management

Desktop computers today are equipped with far more memory than when SVD algorithms like LAS2 were invented. It went from a maximum of 4 Megabyte in the early nineties, to around 16 Gigabyte in 2010. This means desktop computers are capable of processing way much larger data files. But, efficient memory management is still absolutely necessary. For example: the RFP data set in its completely raw unmodified form has 1,151,889 unique terms and 4,975,897 unique URLs. When trying to load its association matrix into a simple array of doubles (were each double is an 8 byte floating-point variable), these dimensions are multiplied times 8 byte, that is 45,853,448,155,464 bytes or 45,85 Terabyte of

RAM just to load it. This is obviously a huge waste of space, since most of the allocated memory is never used due to the sparseness.

So a much more efficient way to store the information is required. This could be accomplished by using a special data structure described by Horowitz et al. in [14, p. 123–127]. It's implementation will be discussed in more detail in chapter 4.4.2. Another way to reduce the needed memory is to decrease the size of the association matrix by reducing its dimensions before constructing it. To accomplish this, several methods will be proposed in chapter 3.

2.5 Numerical stability

Due to rounding errors in computer algorithms, the outcome of floating point operations could be seriously influenced. Higham describes these type of errors in his book [12]. Rounding errors occur when for instance working with fractions that have an infinite precision for example 2/3, which is 0.6666666 into infinity. Depending on the rules of the programming language and the compiler, the last digit could be rounded to a 7 to make the number fits into a specified variable. A couple of examples of what kind of real life consequences these type errors have caused, is shown by Erik Weisstein in [25]. Even though in this area of expertise numeric stability issues are not life threatening they should be prevented or at least minimized since they could render a research invalid. Because of that, special attention will be dedicated to see if and how numeric instability occurs and how it is prevented as much as possible.

2.5.1 Numerical stability in C Sharp

Today modern programming languages comply with several standards and have lots of stabilizing algorithms integrated in their core to minimize the numeric instability. However calculation errors can still occur easily. Looking ahead to the implementation of the tool in C#, a small test was conducted in both MS Excel 2010 and C# to determine whether these programs are still sensitive to numeric instability when working with floating point operations. This was done in Excel by entering the following formula into the formula bar: $= 1*(0.5-0.4-0.1)$. For C# , a small program was written to calculate the result of the same formula:

```
class Program
{
    static void Main(string[] args)
    {
        double d1 = (1 * (0.5 - 0.4 - 0.1));
        decimal d2 = (1 * (0.5m - 0.4m - 0.1m));
        System.Console.WriteLine(d1.ToString());
        System.Console.WriteLine(d2.ToString());
        System.Console.ReadLine();
    }
}
```

In the case of D1, double variables were used, in the case of D2, the decimal variable type was used. Doubles have a size of 64 bits, a precision of 15-16 digits and have a range of storing values between $5.0x10^{-324}$ to $1.7x10^{308}$. Decimals however, have a size of 128 bits and can contain 28-29 decimal places. Its range lies between $1.0x10^{-28}$ and $7.9x10^{28}$ *. The output of the experiment is described in table 2.2.

Table 2.2: numerical precision experiment

MS Excel	C# - D1	C# - D2
-2.77556E-17	-2.77555756156289E-17	0.0

This means both Excel and C# have an issue with numeric stability when working with (double) floating point variables. The C# type decimal however seems unaffected by this test. Unfortunately this type of variable takes up twice as much memory, thus using it for calculations requires more processing power. Some people also claim† that using decimals is about 20 times slower than using floats or doubles. So, performance wise using decimals instead of doubles is not an option given the number of computations that have to take place.

This typical behavior with the numeric stability of double variables is the result of the IEEE 754 standard that specifies how to store floating-point numbers in an easy to manipulate and compact way. The reason for this behavior is that in binary format, some numbers cannot be stored [12, pp. 41–43].

So what are the consequences of this behavior? The IEEE 754 standard, specifies that for the basic operations (+,-,* and /), the square root and the remainder that their values are first precisely calculated before rounding [8]. This means that when calculating the TF–IDF values, the round off errors are reduced to a minimum. However, the log function is not covered by the standard, although IEEE does recommend it. Information about whether C# confirms to this advice or not, wasn't found.

2.5.2 Numerical stability in SVD

Bai et al. describes several libraries and packages that contain either the Arnoldi or Lanczos algorithm. These two iterative algorithms are especially suited for handling large and sparse matrices [1]. A disadvantage however is that to perform well, they approximate the Singular values, rather than exactly calculating them.

Doug Rohde, the author of SVDLIBC states on his website‡ that the LAS2 algorithm (based on Lanczos) he implemented in his tool *"has the drawback that the low order singular values may be relatively imprecise"*. The reason why the

*http://en.wikibooks.org/wiki/C_Sharp_Programming/Variables

†http://gregs-blog.com/2007/12/10/dot-net-decimal-type-vs-float-type/

‡http://tedlab.mit.edu/d̃r/SVDLIBC/

the values become more imprecise has to do with the number of iterations that are necessary before the Eigen values convergence. This is explained in more detail by Parlett et al. in [19]. However, E. Cahill et al. describe that a new version of the Lanczos algorithm has been developed, that accounts for the precision of the Eigen values up to machine precision [3]. However, a suited implementation of this algorithm was not found. But, since the most important concepts are represented by the high order singular values and the low order values are thrown away, numeric instability this has no important influence on reducing the association matrix by using the SVD.

Since the number of iterations before convergence can be quite high, the SVDLIBC program that contains an implementation of the LAS2 (Lanczos) algorithm has several parameters that can be used to find a balance between performance and precision. These parameters are described in more detail in Appendix A.

For the straight forward approach (not applicable for large and sparse matrices), N. Higham, tests the numerical stability of several SVD subroutines (QR-factorization) in the second edition of his book [12]. It could be interesting to compare how the Singular Values of a matrix are influenced, by using stable and in-stable algorithms. A small experiment to observe this has been performed in chapter 4.6

2.6 State of the Art

Since the click data contains lots of noise, special methods need to be used to remove it. As already explained, the Singular Value Decomposition (SVD) is such a technique and has been widely used in different areas of science, like whether prediction, construction, and bio informatics [16].

Using the Graphics Processing Unit (GPU) , the processor of the graphics card, to solve linear algebra problems is becoming more popular. CUDA is the parallel computing architecture by NVIDIA, that allows programmers to use the GPU for computations trough the CUDA Framework. Since GPUs are optimized for floating point operations (measured in FLOPS or floating Point operations per second), they are better suited for the job than a regular processor, which is optimized for integer multiplications.

Recently S. Lahabar and P.J. Narayanan developed an SVD solver for dense matrices using CUDA [17]. Computing the SVD of an 14,000 x 14,000 matrix, took 4573.2 seconds, on a 4 Terra FLOP machine (the NVIDIA Tesla S1070). This would mean that on a AMD 64 X2 DualCore 4600 desktop processor, which is 1,200 mega FLOP, it would take 3,333.33 times longer, so about 176 days to complete.

In this particular case, the matrix was completely filled with values. According to Manning et al., there is currently no known experiment in which the SVD of a matrix with a dimension greater than 1 Million has been computed [5]. In the case of click data, matrices tend to be even larger but they are also very sparse. This means only a small fraction of the matrix is populated with non-zero values.

The intensity of a the straight forward approach is around: $O(M \cdot N^2)$ [19, pp. 135] and it doesn't try to take advantage of the sparseness. Usually for large and sparse matrices (matrices with both dimensions over 1K and a low density) it is more likely to compute a reduced version of SVD or to calculate the singular values by approximation.

This shows what a complex problem the SVD still poses for very large matrices. Especially when such a problem has to be solved on a desktop computer. To the author's knowledge, there are currently no tools available for the Windows platform that are capable of simply converting a given data set to an association matrix and then apply the SVD to that matrix.

2.7 Summary

Click data is an interesting source to gather relevance feedback from. After examining the different relations, the 'query terms – URL' relation seems the most interesting one and will be studied further.

Singular Value Decomposition is a suited technique to extract the concepts from the click data and to filter out the most important ones. But, given the huge amount of data that needs to be processed, memory management will become very important. This means special attention must be dedicated into shrinking the association matrix before processing it. However this has to be done in a responsible way, with respect to the semantics of the data. Also the speed / computing power required for computing the SVD is so immense that it requires a special algorithm that takes advantage of the sparseness to be able to complete the task within a reasonable amount of time. Algorithms best suited for this task are either based on the Arnoldi or Lanczos iteration. Unfortunately these algorithms have not been found for the CUDA platform. Even though those two algorithms calculate the singular values by an approximation, they provide the best alternative for calculating them, considering the intensity of the task given the size of the matrix. A stable version of the Lanczos algorithm has been developed. Yet, no freely available version has been found so alternatives have to be considered.

In terms of numerical stability, it is expected that the most instability could occur in the implementation of the Lanczos algorithm, depending on the version of implementation. Most of the basic operations that are needed to calculate the TF–IDF values fall under the IEEE 754 standard, meaning that the round off errors fall within the range of the machine epsilon. However the standard does not require this for other mathematical functions like Log, although it is recommended.

Chapter 3

Introduction of an IR Data preprocessing tool

This chapter describes a conceptual tool that supports the data analysis process for Information Retrieval. The first part determines the scope of the tool and specifies some practical requirements concerning usability that need to be kept in mind for implementation later on. The second part describes what steps are taken to go from a plain text data file, to a sparse association matrix and what kind of techniques could be used to accomplish this. In the third part, the research deviates a little from its original path and sketches an approach that could be interesting for future work. In the last part of this chapter is described how the tool should deal with transforming the association matrix to concepts, in other words computing its SVD.

3.1 Requirements

The scope of this work will focus on a modular framework that enables each step in the data analysis process and will implement a generic method to handle a given data collection. This will allow for multiple analysis methods and filters to be implemented. The tool will be systematically validated to make sure the produced outcome is valid. Also to be able to reproduce the experiments completely, log files will be created, showing all the taken steps, the applied features and the order they were applied in. Thus, the original data set is never altered. To prevent the system and the user from having to interrupt or interact with each other during the process, all options are entered in advance, so the system can analyze the data without further need of the user. The system should be able to store the results of each step separately so the researcher could examine them individually. Since many advanced analysis algorithms are freely available, this tool should also implement a generic interface, so that it can deal with different kinds of third party IR algorithms. Also in order to exchange data with other analysis tools, it is necessary that the tool can convert the data to the formats required by the third party tools.

3.2 Data Analysis Process and Information Retrieval

A general data analysis process is defined by P. Ender in [6]. He describes four phases: gathering, cleaning, modifying and analyzing the data. Below is described how this can be interpreted in terms of Information Retrieval (IR). Assumed is that the needed data has already been gathered and that it only has to be converted to an internal format instead. Also as a necessary step, data compression is added and after the experiment is completed, a log file should be created. This process will form the basis around which the analysis tool is further developed..

1. Conversion to a workable format for the tool
    - selecting a data file
    - determine the columns and fields
    - selecting the relations that need to be analyzed
2. Cleaning the data using filters
    - select available filters and their options
    - apply them in a specific order.
3. Compressing and transforming the data to make it workable
    - replace all unique terms and URLs with unique numbers
    - write an translation file to be able to relate to the data
4. Modify the data
    - determine the associations
    - create the association matrix
5. Analyze the data:
    - Select the IR techniques that need to be applied
    - Write the association matrix to disk in the selected format

3.3 From plain text to an Association Matrix

The scope of this tool, will primarily focus on transforming the plain text data file to an association matrix. To do that in an organized matter, an Object Oriented

(OO) approach is chosen and the tool will be split into several different modules as shown in figure 3.1. Each module will be responsible for its own distinctive task. This will make expanding the program's functionally easier, since objects can be reused or expanded in a more responsible way (More about OO in chapter 4).

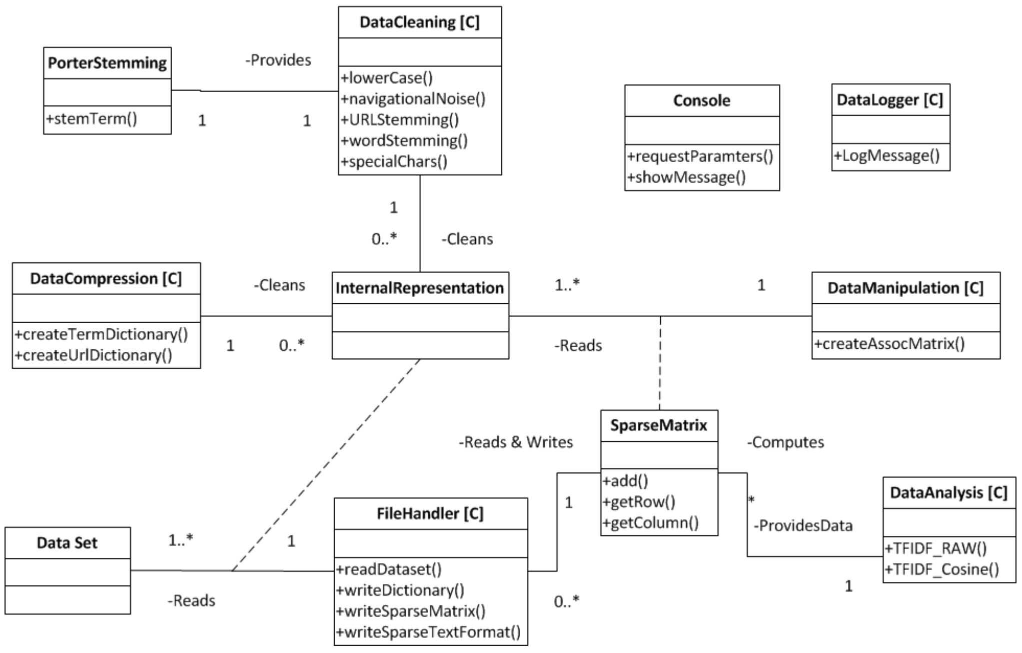

Figure 3.1: UML Domain Class Diagram

User interface The tool needs to interact with the user through a Graphical User Interface (GUI). Through the GUI the user can enter the parameters for the analysis. It will also be used to monitor the progress. A simple console based user interface has been chosen for this task. But, because of the tools modular structure, it can later on be replaced easily by a more enhanced one.

Log Module To document the analysis process, a logging module is created. Each time the program runs, it creates a log file containing the execution times of each step. Also it keeps track of the output files that were generated and all the parameters are logged when they are entered manually. This enables the researcher to collect the results and statistics easily. An example of the program's log file can be found in appendix A.3.

File handler The file handler is the part of the program that is used to read the selected data set from the external representation and transforms it into the internal representation. It will also be responsible for writing the logs files of the experiments and the results back to the hard disk. To be able to exchange data with other applications, this module is also responsible for conversion between different file types. As a safety measure a time stamp is added to the filenames of the output files in order to make sure no results are ever accidently overwritten. This also prevents mix-ups between output files from different experiments when they are located in the same directory.

Data Cleaning Module This module will be responsible for the cleaning of the data set and removes unwanted features by implementing the functions below.

- **Converting all characters to lower case**
  Differences in text formatting can lead to different unique terms while they actually represent the same word. That's why all terms should be converted to lowercase. Whether or not this feature will be applied during an experiment will be made optional. The impact of this basic feature on the number of unique terms and URL's is tested in 5.3.

- **Word Stemming (or suffix stripping)**
  Stemming is a commonly used technique in linguistic morphology to reduce a verb to its stem. For example, stemming would reduce the verb "working" to "work". Porter created such a stemming algorithm in 1997 for the English language [20]. *"The goal is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form."* - Z. Bai [1, pp. 32]. So implementing this function in the tool can group the meaning of different verbs by reducing it to one form, therefore also reducing the total number of unique terms. This filter is further described and tested in 4.5.1.

- **URL Stemming** The idea of stemming can also be applied to URLs. When reducing the URL to just the domain name, the unique number of URL's will quickly drop and the size of the association matrix will be reduced drastically. Thus it will increase the density just like with the word stemming. To prove this, a small experiment has been conducted, which is documented in 4.5.2. But the information that will be thrown away, could prove to be very useful in redirecting the user to a very specific part. One idea would be to create a small separate matrix based on the different subparts of the URL. Matrix multiplication could then be used to eventually redirect the user even more precise. This idea is further drawn up in 3.4. Another idea would be to also recompute the SVD in blocks, where the association matrix for u only contains different variations of one domain and for the query terms qt only terms that are directly associated with these URLs.

- **Navigational noise filter**
  After observing the raw data set, it often seems the query term field consists of a URL instead of query terms. This is odd considering the fact that the URL should be the result of a search and not the origin. It is obvious that this phenomenon occurs when a user inserts the URL in search engine's search field. The more important question is why? From my own experience it occurred that sometimes the URL was accidently typed in the search toolbar of the browser instead of in the address bar. It could also occur because the browser redirects the cursor to the search bar because of an auto focus event. In both cases it is not a real search query, when the URL someone visited equals the term that was typed in.

  These type of data entries in the data set will be referred to as *navigational noise*. One such entry could be seen as an extreme version of a navigational query as described by Manning et al. [5, pp. 432].

  The term weighting functions could filter out this noise eventually, but since memory and speed are important factors, it is faster to filter it out in an earlier stage. This feature has been made optionally applicable to the data set. Its implementation, formal description and results are further discussed in chapter 4.5

Data Compression Module To make the data set more manageable for intensive calculations, it has to be compressed. In this case, that is done by translating each unique term and URL in the data file into a unique integer number, because an integer number takes up less memory than a string. To be able to relate to the data after the analysis and for evaluation of the results, translation files are written to the hard disk (using the File Handler module) one for each dimension of the matrix. Important is that the compression on the data set should only be applied after manipulating the raw data with the Data Cleaning Module, otherwise the filters have no use.

Data Modification Module This module is concerned with transforming the data into a different structure. It determines the associations within the cleaned data set and transforms them into an association matrix from where the term weighting procedure can begin.

Data Analysis Module Before the calculation of the SVD can begin, the relations in the association matrix should to be weighted. This module provides several of those functions as explained in 2.2:

- TF–IDF Raw
- TF–IDF with Cosine Normalization

The idea is that this module will be extended later on with more weighting features such as the ones described in [23].

Sparse Matrix Structure To handle a large and sparse association matrix, a special data collection has to be used. It should provide efficient storage, fast access and it should be dynamic in size. This type of construction and its implementation is further described in 4.4.2.

3.4 Domain specific approach

The drawback of the URL stemming that was described in the previous section, prevents users from being redirected to a specific part of a website. This is somewhat a disaster for sites that covers lots of information about different topics, like Wikipedia. On the other hand, it does allows a major reduction of the association matrix. An idea that emerged to prevent data loss from the URL filter, was to create a small association matrix for each domain that links it's different qualifiers (keywords that are a part of the URL) to the specific domain.

$$example : http : //www.\underbrace{ru.nl/}_{\text{domain}}\underbrace{iii/onderwijs/informatiekunde/master/}_{\text{qualifiers}}$$

1. DQt is a relation that connects domains and query terms.
   $DQt[d,t]$ is the estimated relevance of t for d

2. DqD is a relation between domain qualifiers and domains.
   $DqD[s,d]$ is the estimated relevance of s for d

combined:

$$DqQt[s,t] = \Sigma_d DQt[d,t] \cdot DqD[s,d]$$

Also the position of the qualifiers for a certain URL could suggest a hierarchy in the structure, for example from global to specific. This could be used as weighting factor for the different qualifiers. Assuming Dq are the qualifiers for a specific URL and that the more relevant qualifiers are located at the end of the URL, a partial weighting function could look like:

$$\frac{1}{(|Dq| - pos(Dq)) + 1}$$

With this approach, it is suspected that the value of the qualifiers is not lost, while the $DqQt$ matrix is still relatively simply to calculate. If this works in practice is yet to be determined. However, it falls outside the scope of this research, but could be interesting for future studies.

3.5 From Association Matrix to concepts

Understanding the complex implementations of the mathematical algorithms exactly, requires an enormous amount of mathematical knowledge that falls outside the scope of this research. More important is the modular and structured approach for creating a stable research environment that is flexible and also easy to use. But from a technical point of view, the application has to be suited to cooperate with these algorithms. To be able to handle large sparse matrices and eventually calculate the SVD of a matrix, there was a choice to be made between incorporating a third party linear algebra package in the tool, write the necessary operations from scratch or a combination of both.

Several advanced mathematical libraries and frameworks are available for C#. Some of them focus especially on linear algebra and have been studied in more detail to consider implementation.

- **NMATH 2.0 Core**
  NMATH is a commercial off the shelf product that probably could have been useful. However, since it was only available as a 30 day trial version, it could only be used a short period of time. This would not allow for much time to implement it in and test it with the tool. Also the program should become expandable and open source, therefore this solution was discarded.

- **Math.NET Numerics**
  This program was formerly known as dnAnalytics and falls under the New BSD License. It is a very advanced open source linear algebra framework that contains both an SVD algorithm and a sparse matrix construction. But, after some research on the internet[*], it seemed the SVD algorithm they implement is a aimed at dense matrices thus not suited to handle the type of large and sparse matrices that need to be analyzed by the tool.

- **ALGLIB**
  ALGLIB[†] is a mathematical library that also contains a Singular Value Decomposition algorithm. ALGLIB is distributed under the GNU General Public License, so it is allowed to be freely redistributed. This algorithm works with two dimensional double arrays. Because of that, it is relatively fast, but hard to apply on sparse matrices. Therefore the author of the ALGLIB library, S. Bochkanov, was contacted, about changing the algorithm to make it compatible with the sparse matrix structure of the tool. It was then learned that their implementation was unfortunately aimed at very dense matrices. So, when it was applied on data set in mind, it

[*]http://alexreg.wordpress.com/2009/05/15/numerical-analysis-for-net/
[†]http://www.alglib.net/

would literally take centuries to complete calculating the SVD (assuming the computer wouldn't cash first).

After considering the different packages, it was decided to implement the necessary basic IR features from scratch. This way there was more control over the implementation and it also helped to better understand the different techniques. However, the different SVD algorithms are all such complex pieces of work, that this part would be used from a third party. Luckily S. Bochkanov (author of the ALGLIB library) was very helpful in pointing out several packages that contain such algorithms. These are the ones described by [1]. This implementation process will be described in the next chapter.

Chapter 4

Implementation

This chapter describes the actual implementation of the suggested tool. The first part describes the motivation for the programming language but also proposes a global architecture. Then some internal data structures are described to coop with the large amount of data the tool has to handle. Also several filters are described here in more detail about they are used to reduce the matrix dimensions. Furthermore a selection is made in how the chosen SVD algorithm is implemented and the validation process is described.

4.1 Design decisions

To provide modularity and flexibility in the tool, an Object Oriented (OO) approach has been chosen. Using the OO approach, allows programmers to manipulate objects through their own interface. This means that they don't have to know explicitly how the objects work inside. One important disadvantage of OO has been taken into consideration. Using OO, is expensive in terms of memory consumption, especially when working with flexible OO data structures. This is due to the object references that take up lots of memory (max 64 bytes each, depending whether the OS has a 32 bits or 64 bits architecture). This could mean a that an Object consumes more memory for its own meta data then for the data it has to store. Therefore in the implementation of the tool, OO data structures have been avoided as much as possible to coop with this problem.

As an operating environment, Microsoft Windows has been chosen. What has been learned throughout this process, is that it is not very common to develop these type of programs for the Windows platform. Scientist prefer the UNIX environment, since its strong command interface helps to transfer data from one script to another. However since this somehow is still an Information Science and there were already enough challenges, it was decided that using a different operating environment with its own programming languages shouldn't be one of them. Looking back at this decision it also meant that there were limited resources for existing scripts available. Therefore it probably required more effort to work around it, then if this decision was reversed.

For the implementation of this tool, C# (pronounced as C Sharp) has been chosen. It has a similar syntax as JAVA, thus it is also based on the OO approach that will support modular programming. An advantage of C# over JAVA, is that it allows the use of pointers. This will give the programmer more control over memory management, that is required to create the custom data collections. Even though C# is also not known for its speed, which still is an important factor for large amounts of data operations, it does provide flexibility, modularity and stability. Also additional libraries are available that contain optimized routines that can account partly for these differences in speed.

In the previous chapter, the data analysis process for click data was described. Eventually it has been implemented as shown in figure 4.1. On a technical level, each of these steps was implemented in a specific module. To put the responsibility of each module into context, they were divided into several layers using a three-tier structure.

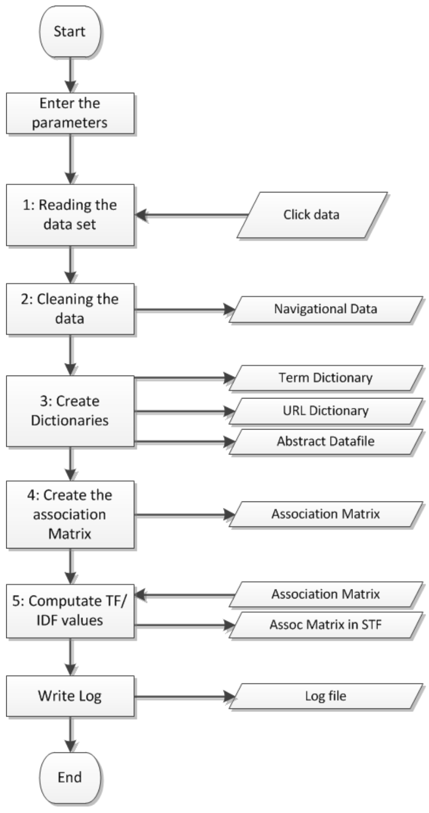

Figure 4.1: Data Analysis Process as implemented in the tool.

4.2 Segmentation using three-tier

For the tool's architecture a three-tier structure was kept in mind, as shown in figure 4.2. This means that the tool's representation, business logic and data access would be separated into different layers. It makes the tiers independent of each other, so that each layer could run on a separate system. This type of architecture has been used before in IR software by Grabs et al. [10]. It would allow a special optimized machine like the NVIDIA Tesla S1070 to compute the SVD, while another machine simply stores different data sets. Another advantage is that when one of the individual tiers is updated, it doesn't affect the functionality of the others. Below each of the different layer's responsibilities will be briefly described.

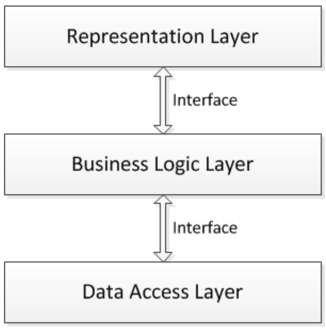

Figure 4.2: A three-tier structure.

Representation Layer The Representation Layer (RL) is also known as the Graphical User Interface (GUI) . It is used to communicate with the user by displaying information and providing input forms. This layer submits the input to its underlying layer, the Business Logic Layer. In the case of this tool, the GUI will be used to configure the settings for the experiment, showing the progress and afterwards to display its results.

Business Logic Layer The Business Logic Layer (BLL) is the middle layer. As its name states, this layer contains the business logic, meaning that it controls the working behavior of the application. If a user interacts with the GUI, for example to configure an experiment, the BLL will receive a request and triggers its responsible controller to take action. This controller will then gather the needed information using the Data Access Layer. After processing it sends back the results to the GUI.

Data Access Layer The Data Access Layer(DAL) is responsible for retrieving, storing and manipulating the data. It this particular case it consists of a file handler and different data collections. When the file handler, receives a request from the BLL, it either retrieves the requested data from disk and sends it back in a custom data collection or it writes a data collection from the BLL to the hard disk in a specific file format. Each of the data collections provide an easy interface towards the BLL. This way the BLL can work with its data without the need to know of how the data is stored in the underlying implementation. Because of the file handler's flexibility, the tool can work with the data of different data sets, that represent information in a system based on rows and columns.

In the implementation of three tier in this tool, the three layers are only logically separated into different packages, not in different systems. If the different layers where to run on different type of architectures, it could influence the numeric stability of the tool, due to different mathematical standards. So for now this unnecessary complexity of the application is avoided. However if in the future the program was to be expanded, it would require the implementation of a facade interface to accomplish this. An advantage of this approach without the facade is still that it makes the tool easier to understand for people who want to adjust or add functionality. Also when multiple people are working on the same project at once, it is easier to assign responsibilities, since each programmer can work on his assigned module with a predetermined interface.

When the Domain Class Diagram from the previous chapter (see fig 3.1) is combined with the separation in three layers, it results in the architecture[*] displayed in 4.3.

[*]Note that the main program, is also a controller that incorporates and manages all the other classes. It sends messages to the Console (GUI) and to the Data Logger. However, this is not visualized in the model. Therefore, it seems these the Console and Data Logger don't play a role in the system.

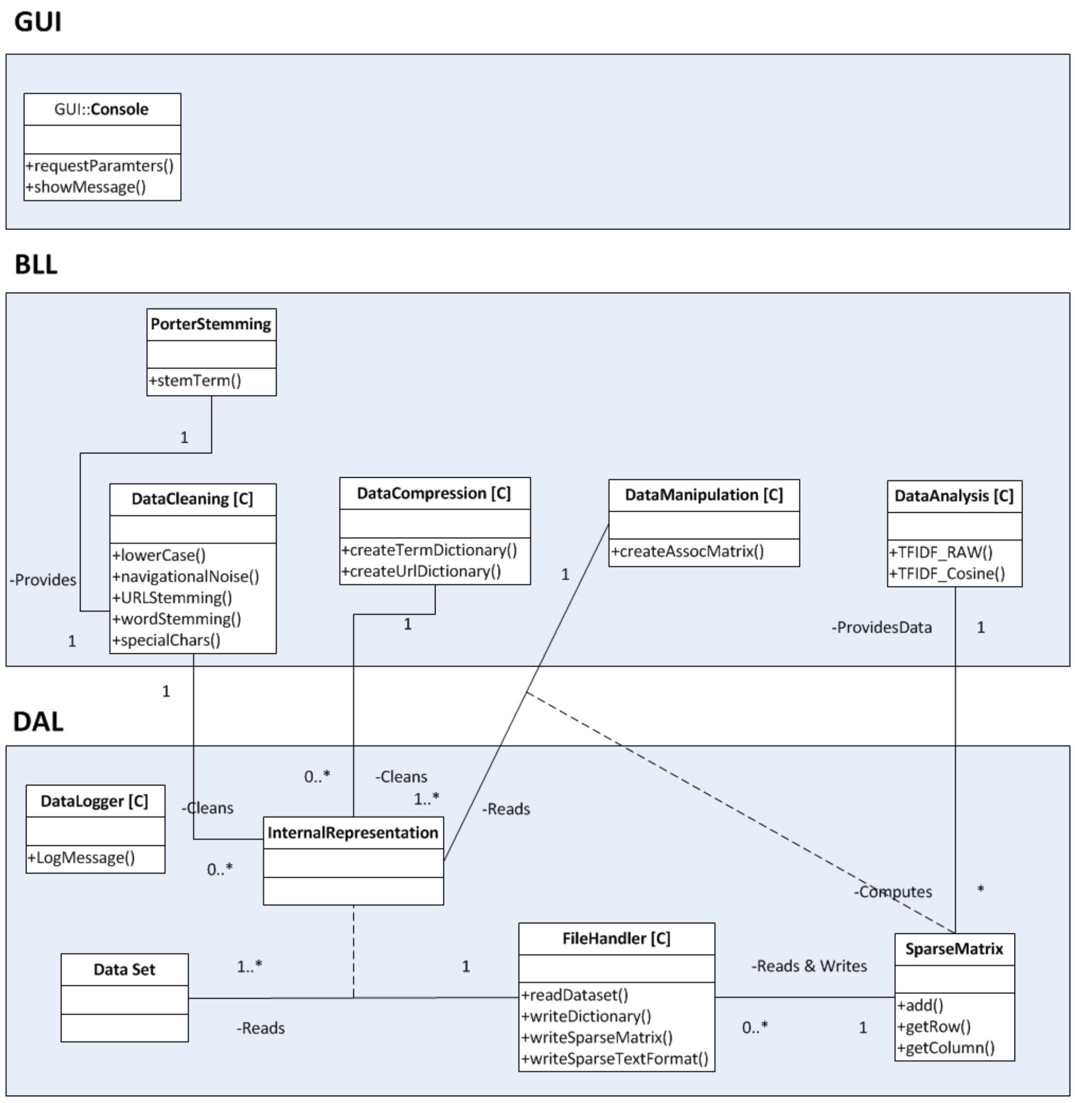

Figure 4.3: Architecture of the analysis tool.

4.3 Design patterns

Design patterns are standard solutions to solve standard situations or problems that occur when developing a program. They can contribute to create a solid foundation for the tool. E. Gamma et al. describe them in more detail in their book [7]. Several of them are suggested or have already been implemented in the architecture of the analysis tool, to provide for more flexibility and modularity.

The Singleton is one of the most commonly used patterns. It insures that there is only one instance of a certain class during the execution of the program. This way it prevents communication conflicts between different data controllers and could also save memory. An example of how this pattern is implemented in the tool can be found in its Log Module. It assumes only one experiment is running at the same time and that there should only be only one log file for each experiment. If one the controllers wants to log an event, it checks for a reference to the Log Module. If that reference doesn't exists, the Log Module is initiated and its reference returned. So the controller always receives a reference to the Log Module. The controller can then send messages to the Log Module which ultimately stores them in the log file.

For completely separating the three layers into different subsystems, a Facade pattern could be used. This pattern would provide the subsystems with a single interface. So instead of all individual classes communicate directly to each other, it passes through the facade interface. The benefits of this pattern are that it promotes a weak coupling between the several layers and is absolutely necessary if the tool was to run on multiple systems in the future.

Another patterns that was considered but has not yet been implemented was the command pattern. This allows to apply IR techniques to the sparse matrix structure in a more neat way, so other techniques could be applied easier.

4.4 Data structures

The development of the tool required the implementation of some special data structures. In order for it to be flexible, it has be able to work with data sets that are dynamic in size. A problem is to estimate the size of the required storage for a given data set in advance. However a static data structure could either claim lots of unneeded memory or it can cause memory overflows. An alternative is to use dynamic structures. But when working with very large data sets, dynamic structures are usually very inefficient in terms of speed and memory.

4.4.1 Hybrid Arrays

The most efficient solution would be to somehow keep the memory costs of static data structures but to gain the flexibility of dynamic structures. Therefore a hybrid form of a static and a dynamic structure has been created using recursion. With a custom created array wrapper class (which is normally only used to represent a primitive variable type, like an integer, as an actual object), this problem was partly overcome. The idea behind the custom wrapper class is that it provides read and write access to a primitive data type (an array of strings or doubles) and also protects it from overflowing. When the primitive type is full, the wrapper class initiates a new private instance of its class and redirect's all new data to that new instance using recurs. So from the outside it appears nothing has changed.

Say w is a custom array wrapper class and X is defined as function that gets the size of w. Assume d is data that needs to be stored in w. When $d > X(w)$, w creates an internal w_1 which is filled with $d - w$ until $X(d - w) = X(w_1)$. Now w_1 initializes an internal w_2, redirects the data towards it and the process continues. Using the induction hypotheses it can be concluded that eventually the remaining data $d - \Sigma(w_1..w_n)$ fits inside a certain w_m and the process stops.

What is important here, is the initial size of the new wrapper. This data collection is theoretically infinite if $X(w) > 0$ and doesn't change during the process. In practice the data set is not and the computer computer will eventually run out of memory. However each time the wrapper expands, the slower the structure becomes. This is because the data adding and retrieving the requests need to pass through all the previous wrappers, meaning an increase in access time. Therefore it would be smart to implement a certain decreasing factor for w. If it is assumed the person that initiates the experiment has a good estimate about the size of the data that needs to be stored, it could be that w_m is initiated with $X(w_n) - f$ where f could be a percentage of w_n. This does make the collection definite, but it gives the user some room for some error.

4.4.2 Sparse Matrix Structure

As already mentioned in 2.4, traditional data structures like double arrays are not capable of handling the enormous dimensions of an association matrix and would store information in an inefficient way. So a smart solution had to be implemented that only stores the non-zero values of the matrix. Therefore a custom sparse matrix construction was developed. It has much in common with a sparse matrix data structure that was described by Horowitz et al. in [14, pp. 123–127] which is also known as an orthogonal list.

A normal linked list consists of a chain of nodes that each contain actual data and also a link to their neighbor node. An orthogonal list (see figure 4.4) could be seen as a 2 dimensional linked list. This means each node, links to two of its neighbors; one neighbor in the same column and one in the same row. This was done to provide quicker access to both columns and rows thus to allow for faster matrix operations.

Each block in the figure represents a node, containing the coordinates in the matrix, its value (not displayed) and the two pointers. The pointers are represented by the arrows, while the X-dimension and Y-dimension represent two access arrays that contain the reference to the last inserted item.

In a first attempt to implement the orthogonal list, a full object oriented structure was used to create this data structure. It was easy to build and allowed for the matrix to be very dynamic in size. But, because of the many individual objects, the total program took up 3.8 Gigabytes of RAM just to store the association matrix, while it only contained about 550 Megabytes of actual data. This

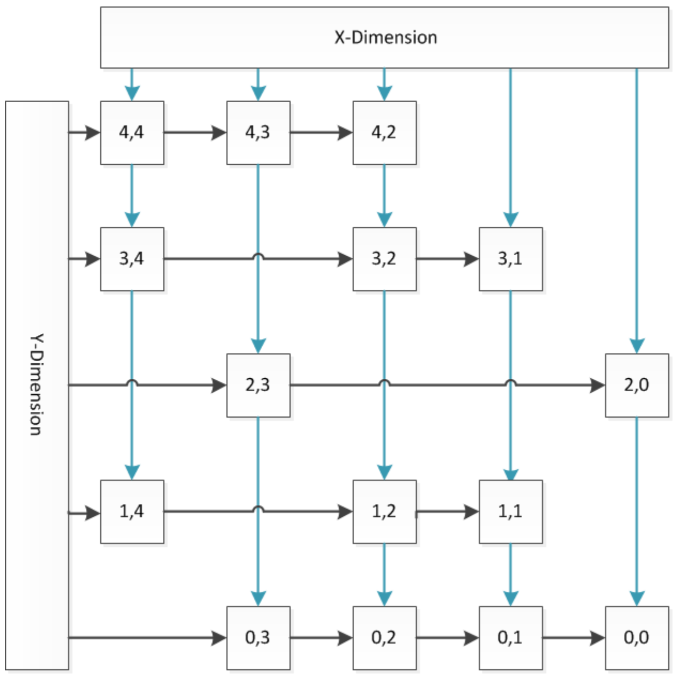

Figure 4.4: Orthogonal list

proved that way of storage was not efficient at all. The reason had to do with the way how C# stores its value in the memory[†]. Normally the number of objects (instances of a class) is relatively low so a few objects more do not matter. But in this case, there were 18 million objects for the data alone. Each containing multiple object references that (as already has been described in chapter 4.1) are quite expensive in terms of memory.

In the second attempt, a combination of a traditional approach and an object oriented approach was used. The class is still object oriented but its underlying data structure now use structs instead of objects. This change replaced the object pointers with address pointers, which are much smaller to store. However this required to implement manual memory management which is quite unusual in a language like C#, since the idea is that when the compiler addresses the memory it is safer. But the result of custom memory management was a RAM usage of only 650 megabytes for the loading of the association matrix: a reduction in size of almost six times. Also the execution time of the program to load the matrix was reduced from 5:49 minutes to less than one minute.

The class maintaining the orthogonal list is called 'Sparse Matrix' and stores the data in three hybrid arrays described earlier, where:

- one array contains all the nodes with the actual data.
- one array contains pointers to each last inserted node in the x-dimension.

[†]http://en.wikibooks.org/wiki/C_Sharp_Programming/Variables

- one array contains pointers to each last inserted node in the y-dimension.

Insertion in this collection is of order O(N), since all new nodes are inserted at the beginning of the list. Since the operations for TF–IDF are concerned in reading or manipulating an entire row or column, and the row remains sorted (only reversed) this doesn't pose a problem. It also explains why in the picture the coordinates are mirrored. The File Handler accounts for this phenomenon when it writes the matrix to hard disc.

Approximate storage requirements for the nodes of this collection:

- consider p the storage of one pointer in bytes, i for the size of one coordinate and d as the size of a double floating point.
- Assume $p = 8$ bytes, $i = 4$ bytes, $d = 8$ bytes and $2p + 2i + d$ meaning each node would require a memory of 32 bytes.
- So storing a matrix of 18 million nonzero values in the computers RAM, requires about 576 Megabytes.

At first sight it might seem that storing only one of coordinates in each node would save additional memory, since the array's index already represents one of them. However if that was the case, with the implemented structure, two node objects would have to be created for each matrix value to store the other coordinate, resulting in $2 * (1p + 1i + d) = 2 * (8 + 4 + 8) = 40$ bytes for each node.

4.5 Filtering the data

At first a personal goal in analyzing the click data, was to be able to keep the matrix as large possible since that meant meant manipulating the raw data as little as possible, since it would probably keep more of the semantics in tact. However a problem occurred when the Singular Value Decomposition (SVD) has to be computed. The size of the association matrix was enormous with dimensions of $4,971,990 \cdot 778,160$. As described in 2.6, currently there are no known experiments that have been successfully performed on a matrix of such a scale, especially not with a "simple" desktop computer. So the association matrix had to be reduced, but still with respect to the data. In this chapter, several filters are described that were applied on the data set to reduce the size of the association matrix. Three different areas were focused on when filtering data within the 'query term – URL' relation:

- the individual query terms
- the individual URLs
- the relationship between query terms and URLs

4.5.1 Focusing on individual query terms

The number of unique terms in the association matrix without any form of manipulation is 1,151,889. Several options are taken into consideration to reduce this number:

- stemming (or suffix striping)
- lemmatization
- removing query terms that are written in a different alphabet.

Stemming is a technique in which a word is reduced to its stem. For example, the verb 'working' is reduced to 'work'. To achieve word stemming, a version of the Porter stemming algorithm [20] was implemented in the Data Cleaning Module. However, what strongly should be kept in mind is that the data set does not only contains verbs. Therefore it can also accidently reduce non-verbs and mutilate the data. So whether or not it should be applied probably depends on the type of data set. The the effect on the size of the association matrix, is tested in 5.3. However, its implementation is not further used during the rest of the experiments.

Another approach that could be taken instead of stemming is lemmatization. Lemmatization is more advanced and reduces a verb to its lemma, the form of the word that is found in the dictionary. For example, the lemma of 'to be' is 'is'. Determining the lemma of a word is often done by using a special lemma lexicon, such as CELEX[‡]. However, this falls in the research area of linguistic morphology and therefore its implementation is left for future work.

Query terms that are formulated in a different alphabet like Chinese, Arab or Greek could also be filtered out, since it is hard to determine their relevance. However, computing the SVD, does not require words to be in the same language to find overlap in concepts [24]. Therefore these query terms are currently not being excluded. However, a filter could be implemented that uses a Boolean expression to filter them out.

4.5.2 Focusing on individual URLs

With a total of almost 5 million unique URLs, this was a dimension that had be reduced drastically. To do this, a special filter was implemented that reduces the URL to a basic form: e.g.: 'www.something.com/'. Throwing away the rest of the URL seems harsh, since it could contain potential information about the content of the site and helps to direct the user to a more specific part of the site. However, two approaches have been suggested to still be able to use this information. One of them is explained in more detail in 3.4.

[‡]http://celex.mpi.nl/

The implementation of the URL stemming filter takes the URL and looks for the first dot and after that for the first slash. This because the first slash could be part of 'http://' or another protocol. Then the part from the start of the URL up to the first slash after the first dot is subtracted as the clean domain (possibly with a sub domain).

$$example : \underbrace{http : //www.ru.nl/}_{\text{part that is subtracted}} iii/onderwijs/informatiekunde/master/$$

If for some reason the URL clicked on, didn't contain a slash or a dot, the filter passes the entire URL, to prevent data loss. In 5.3 the effect of this filter on the data set is further discussed.

4.5.3 Focusing on the relationship between Terms and URLs

In 3.3 a phenomenon in this thesis called 'navigational noise' was introduced. This means that the set of query terms used to describe the URL is equal to the URL clicked on, or: $qt = u$.

For this filter it is assumed, that when a user (almost) knows the exact URL of a website that he or she wants to visit is not actually searching, but simply navigating with a little detour.

The navigational noise filter's first implementation applied the following simple process on each data entry:

1. first convert the contents of the query term field and the URL field to lower case
2. perform a simple string comparison between the query term and URL field
3. separating the navigational data into a separate file.

In this first attempt, only separated 2,699 navigational links exactly matched $qt = u$ on a total of 12.2 million records. This meant it would clearly not cause a significant improvement in decreasing the size of the association matrix.

Then a slightly different approach was chosen. It was assumed all records, where the query looked like a URLs were navigational. A Boolean expression that was used to classify URLs in [13] was applied to the query field to filter out the navigational noise. This function is defined as follows:

$$SimURL(qt, u) = \begin{cases} True & \text{if } qt \text{ equals } u \vee qt \text{ is sufficiently similar to URL syntax} \\ False & \text{otherwise} \end{cases}$$

This will clear data set D of navigational noise, so that

$$D \to \{qt\epsilon D \mid \neg SimURL(qt, u)\}$$

This resulted in a collection of 538,923 navigational noise records, or 4,4% in terms of the original number of records. Out of the 538,923 records that supposedly contained navigational data, the first 200 records of supposed navigational data where assessed manually to determine the precision. Those 200 records were divided into three categories: green, orange and red.

- 165 Green: the record is correctly classified as navigational data
- 12 Orange: the record is correctly classified but one of the following minor errors occurred
  - 2 The URL had a missing or different extension (like .com instead of .org) than the site clicked on.
  - 2 The domains with a small typing error of (a comma instead of a dot or an accidental space within the URL).
  - 8 The terms lead to a sub domain or a subdirectory within the intended URL.
- 23 Red: (false positives) the record is not correctly classified since the query terms it contains do not represent a likely intended URL.

Based on these results, the filter has a precision of 82,5% when just considering the green ones. If the orange URL's where also taken into consideration, the precision would be 88.5%. It also means that about 0.5% of the total amount of records would be lost by filtering the navigational data. However in this setting, the number of unique terms in the association matrix, was also reduced by 15.6% from 778,160 to 657,076. The effect on filtering the navigational noise on the data set and the optimal combination of filters is further described in 5.3.

4.6 Implementation of an SVD algorithm

Z. Bai et al. describes several libraries and packages that contain an implementation of either the Arnoldi or Lanczos algorithm, suited for this specific problem [1]. But they were only available in other programming languages: Fortran, C and C++. Furthermore they were also based on a UNIX environment, using UNIX libraries while C# is aimed at the Windows platform. So just plugging in a library was not possible. Rewriting one the algorithms for C# was also no option, because of the algorithm's size and complexity. Therefore the idea was to compile a DLL from one of those libraries that could later be integrated in the tool.

An UNIX application with a console interface that implemented the Lanczos algorithm, was SVDLIBC. This program was derived from the SVDPACKC library [2], which implements four algorithms to decompose the matrix. Based on the tests performed in [2], the author of SVDLIBC took the most promising algorithm of that time LAS2 (Lanczos), revised its code and added a console based interface.

The SVDLIBC package has been used in several studies that also involve the analysis of large term document matrices [16, 4]. However, since it's implementation is relatively old, it was developed for 32 bit processors and does not take advantage of modern features like parallel processing [4].

Since the SVDLIBC package was written for UNIX, it first had to be tested on the Windows platform. This was achieved by compiling the program using a tool called MinGW. MinGW is a port of the GNU compiler to the Windows platform. It was compiled in a UNIX shell called MSYS together with the MSYSDVLPR plug-in that ran from inside Windows. It allowed the compiled program to run, but only from the MSYS shell.

4.6.1 Testing the accuracy

To test the algorithm's accuracy and to see if it was working accordingly, a small experiment was conducted. This was done by using two SVD algorithms: LAS2, within SVDLIBC and ALGLIB that the xBDSQR and xGESVD subroutines derived from LAPACK. Both algorithms used an example matrix (see figure 4.1)from the Information Retrieval course [23] containing TF–IDF values. The sample matrix had a dimension of 10 x 6 and with 37 non-zero values, it had a density of 61.7%. Even though this is not really a sparse matrix, the most important goal of the experiment was to compare the results of both algorithms.

Table 4.1: Sample matrix after TF–IDF weighting and cosine normalization.

TF	t1	t2	t3	t4	t5	t6
d01	0.16	0.93	0.29	0	0	0.13
d02	0.41	0.85	0.31	0	0.13	0
d03	0.11	0.97	0.22	0	0	0
d04	0.13	0.97	0.20	0	0	0
d05	0.19	0.89	0.42	0	0.04	0
d06	0.01	0	0	0.74	0.15	0.66
d07	0	0	0.02	0.98	0.19	0
d08	0.02	0	0	0.99	0.09	0.09
d09	0.00	0	0	0.77	0.31	0.56
d10	0.03	0	0	0.59	0.07	0.80

First, ALGLIB was put to the test by loading the matrix file into a two dimensional array using C#. Second SVDLIBC was first ran (using MSYS under Windows) with its default parameters: "svdlib -o c:/matrixoutput h:/Matrix.txt", were 'c:/matrixoutput-*' would become its output matrices (Σ, U^T, V^T) and 'h:/Matrix.txt' was the source matrix. The diagonal matrix Σ, that contains the singular values is shown for both algorithms in table 4.2.

In order for SVDLIBC to work with the matrix from the tool, a conversion function was implemented in the File Handler module, to generate the required "Sparse Text Format" [*]. During this conversion, there was no loss in the precision of the values. Also the SVDLIBC tool works with double variables, so it should share the same precision as C#, but again a difference between theory and practice. Looking at the results it is clear the ALGLIB algorithm has a much higher precision then the SVDLIBC output. According to Chulkov this is due to different levels of serialization formats [4]. This can also be seen in the results when you look at the rounding of SVDLIBC between # 3 and #4. Notice that in both rows the values have been rounded, but on a different level of precision.

Table 4.2: Singular Values calculated by ALGLIB and SVDLIBC

#	**ALGLIB**	**SVDLIBC**
0	2.21200216584117550	2.212
1	2.08029429390015700	2.08029
2	0.79406042048884640	0.79406
3	0.30536062903431732	0.305361
4	0.17770961866406934	0.17771
5	0.15512670744751916	0.155127

Küffner et al. have been able to use SVDLIBC to process a Matrix of $516,000 \cdot 128,000$ on a standard workstation with 2 GB of RAM, which sounds promising [16]. However according to Chulkov, the number of dimensions that can be used for the matrix is limited by the 32 bit architecture and the reach of signed integers [4]. Keeping these drawbacks in mind, the SVDLIBC implementation should first be tested on a small part of the association matrix. After that, steps could be taken in trying to extend the bounds of the algorithm or to decrease the dimensions of the matrix even further.

4.6.2 Specifications of several test matrices

To estimate how long it will take to compute the SVD of the entire matrix with the current hardware, the tool has implemented a function that can create reduced versions of the association matrix.

At some point an association matrix was created from the RFP data set. After applying all filters it had 1,818,875 unique URLs, 654,236 terms and contained 13,417,242 non-zero values. Each of the matrices was constructed by using the first x rows and y columns based on a percentage in terms of their dimension size (see figure 4.5). Their specifications can be found in table 4.3.
Something interesting that can be noticed from observing the results in figure 4.6, is the drop in density of the smaller matrices compared to the higher ones.

[*] http://tedlab.mit.edu/˜dr/SVDLIBC/SVD_F_ST.html

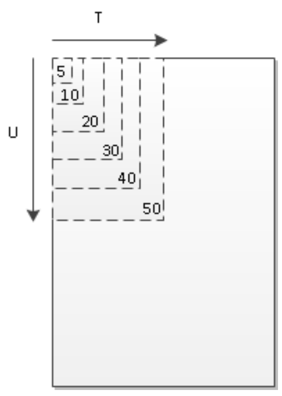

Figure 4.5: Density of the partial matrices

Table 4.3: Specifications of several sub matrices

%	Query Terms	URLs	Non-Zero	Density %
5	32,711	90,943	2,733,765	0.091896312
10	65,423	181,887	4,235,247	0.035591528
20	130,847	363,775	6,211,966	0.013050659
30	196,271	545,662	7,627,296	0.007121816
40	261,694	727,550	8,774,849	0.004608749
50	327,118	909,437	9,758,789	0.003280340
100	654,236	1,818,875	13,417,242	0.001127524

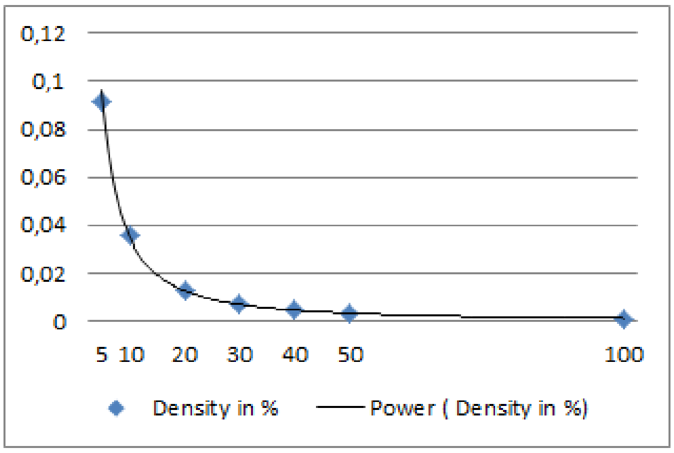

Figure 4.6: Density of the sub matrices

At first it drops quickly in the beginning, but then stabilizes. This could be explained by the fact that the data set has a sequential build up (namely over time). Since popular terms are more often used, they are also likely to appear early on in the log file. Therefore in the process where the unique query terms and URLs are replaced with numbers, the more popular terms will be likely to be labeled sooner and therefore have a lower index number.

Formalized
Assume:
$P(t_1) > P(t_2)$
Let:
$\underline{\mathrm{N}}_1$ = first time t_1
$\underline{\mathrm{N}}_2$ = first time t_2
So it becomes clear that:
$\Sigma(\underline{\mathrm{N}}_1) < \Sigma(\underline{\mathrm{N}}_2)A$

This phenomenon could be related to Zipf's Law that states the 2nd most popular term occurs half the time of the most popular one, the third half the time of the second, and so on. To create a more evenly distribution of the non-zero values, the numbering process should be adjusted, so the performance of the algorithm could be better predicted. However, as will be discussed in the next section, the testing of the SVDLIBC didn't come to that.

4.6.3 Verification of implementation

After the test matrices were created, they were fed to the SVDLIBC program from the UNIX shell MSYS. However loading the matrix failed, since it turned out that MSYS as a console application, could not use more than 256 MB's of the systems 6 GB of RAM due to security restrictions in the Windows operating system.

In a second attempt SVDLIBC was therefore recompiled on a UNIX machine at the University. To make sure the package itself was functioning accordingly, first a small test was run, repeating the experiment from 4.6.1. This test completed in seconds, with the same output as on the Windows platform, so this seemed promising. However, the sample matrices described above seemed to pose much of a problem. At first it was believed that the matrix was just large and computing the SVD would just take a couple of days (since the program showed process activity). But, after patiently waiting it became suspicious when also in a separate thread, matrices with dimensions in the order of a thousand would not complete within several hours.

In a third attempt, to be able to test and debug the program, SVDLIBC was manually ported to the Windows platform. In this process several basic UNIX libraries were replaced by their Windows variant. Most changes made to the program had to do with requesting the system time for registration of the execution time and not with the mathematical subroutines. Also a special preprocessor directive was switched of. It was used to fix a bug on an architecture that did not apply and prevented the program from doing anything. After some trial and error, a version of SVDLIBC was compiled under Windows. Here also SVDLIBC also gave the same results as in 4.6.1, so it was believed that the subroutines of the algorithm were still intact.

Unfortunately this version also lacked to process the larger matrices, after which a debug session was started. The debug session revealed that after several iterations in one of the subroutines, the algorithms values became 'QNaN' or 'Not a Number' and because of that, the program got stuck in an infinite loop. The reason for this problem is still unclear. The SVDLIBC package also has a build in syntax checker to test the matrix source file, so this type of problem was excluded. However the errors did not appear on all test matrices. Several tests with even smaller matrices have been performed. Calculating the SVD of two sub matrices, the resp. the 5% and 50%, based on the first 1000 records of the original data set completed within a second. The largest of them having dimensions of $328 \cdot 207$ with 526 non-zero values, which was nowhere near the results from [16]. With these two matrices, there seemed no problem at all in calculating the SVD. However, the other generated sub matrices from the same run, with sizes in between: the 10%, 20%, 30% and 40%, all caused 'heap corruption' errors.

Eventually a hard decision had to be made: calculating the SVD was put on hold, since it would take up too much time and delay the rest of the research even further. A partial explanation for the errors might be found in [4] where he describes several drawback of SVDLIBC's implementation.

4.7 Validation of the tool

During the development process of the tool, each step in building the association matrix was validated. This was done by using an association matrix from the Information Retrieval Course [23] as input and comparing its output to the results in the material. First, the matrix from the example was manually converted to a simple sparse matrix format in which each record would consist of their respective x and y coordinates and their value so that $r =< x, y, v >$. Second the matrix was loaded into the custom developed sparse matrix construction. With a simple visualization function, this relatively small matrix was displayed on the screen and checked for inconsistencies. After validation of the input, the TF–IDF values where computed using the tool and compared against the outcomes from the course material. Since the TF–IDF values had been pre calculated in several steps, it provided a good foundation to check whether the techniques were implemented correctly. In the same way, cosine normalization was also applied. Using the induction hypothesis, it was assumed that when the tool provided the right output for this small matrix, it would also provide the right output for larger matrices. Eventually each of the weighting functions implemented in the tool, has been successfully validated this way.

4.8 Evaluation

The result of the implementation is a tool that can analyze large data sets in a Windows environment on a desktop computer. The extendible architecture provides basic IR functionality on custom developed data structures. It's flexible file handler allows to analyze two different dimensions of a random large data set. The Data Cleaning module provides several filters to manipulate the click data in a responsible way. However using them is still left to the user. The logging module allows the researcher to view all parameters and different steps that were taken in a certain experiment, so experiments can be easily reproduced. For computing the SVD, the tool depends on a third party application that when even if it would functioning properly is still known to have several drawbacks. Hopefully in the future a better suited SVD package becomes available, so it can be properly integrated and the concepts extracted.

4.8.1 Future work

During the implementation of the tool, some concessions were made in terms of usability since functionally had a higher priority. Because of that, the Graphical User Interface (GUI) is currently console based and perhaps not as user friendly as it could have been. Also improvements should be made so that the parameters of the filters can be changed more easily. In section 4.5, several suggestions are left unimplemented, like lemmatization. Also extra functionality could be added to the hybrid data collections and sparse matrix structure, so that they become even more self-regulating.

To reach out to other developers, integrating the .NET Managed Extensibility Framework (MEF) in the would allow for even more modularity, because it would allow new modules (containing different weighting functions or other IR techniques) to be added without needing to recompile the entire solution.

Chapter 5

Experiments

In this chapter the performance of the tool is tested. First the experimental setup is described: the data set, hardware and variables. Next three experiments are described. Its goals are to measure:

- the performance of the tool itself in light of the amount of data it has to handle.
- the influence of the data cleaning module on the dimensions of the association matrix.
- the influence of the data cleaning module on the execution time.

5.1 The experimental setup

5.1.1 Data Set

The experiments are based around on the Microsoft RFP data set, which contains click data gathered from the English website of the Microsoft Live Search Engine. This flat text file is a little over one gigabyte in size and contains over 12.2 million log entries of search requests. The data has been collected during the month may in 2006. A sample of this data set has already been shown in 2.1.

The data set was introduced during the Workshop on Web Search Click Data. Therefore it is sometimes also referred to as the WSCD09 data set. The workshop[*] was held in conjunction with WSDM 2009 and the data set has already been used in other IR related studies: [11, 13, 22].

The obtained version of the data set initially contained some minor errors. In total, about five records were corrupt, since some of the data fragments occurred outside of the document's structure. Because of that, they caused some problems

[*]http://research.microsoft.com/en-us/um/people/nickcr/WSCD09/

with automated reading and have therefore been removed. But the loss of five records on a total of over 12.2 million records hardly compromises the validity and integrity of the entire data set.

5.1.2 Environment

The experiments have been conducted on a desktop computer (see table 5.1), since in the requirements it was determined that the tool should be a desktop solution. Because the total amount of RAM that can be assigned to one process is limited to 1.2 GB under a 32 bit Windows Operating System (OS), a 64 bit version was used since processing large data sets could quickly require this amount of memory. To make sure the interaction between the OS and the primary hard drive doesn't influence the performance of the tool, the data set was located on a separate one.

Table 5.1: Used hardware

Processor	AMD X2 Dual Core 4600
RAM	6 GB
Hard disk (with data set)	SATA 3Gb/s 320GB 7200 Rpm
Operating System	Windows 7 (64 Bit)

5.1.3 Variables

During the experiments, different variables have been monitored. These have been divided into two groups: variables that relate to the performance of the tool on a given PC and variables that give insight in how several steps of the tool influence the size of the data set. These results can be used later on to evaluate and if necessary sub optimize each step of the process. The variables below are used to measure the performance of the tool.

Processor usage Since the tool was not optimized for parallel processing, only one processor core could be used. This core was dedicated to the application, using the Windows Task Manager to make sure other processes had little influences on this task. The load on that core was almost 100% during the experiments, during which other applications had been closed as much as possible to minimize system activity.

Memory usage Another variable that relates to performance was the tools RAM usage that was measured during every step of every experiment. Unfortunately since the Visual Studio is not very transparent when it comes to memory allocation of non-system variables, the RAM usage had to be monitored with the help of the Windows task manager and a C# system class called 'PerformanceCounter'. The performance counter was used to logged the total amount of available RAM A for the entire system at time t. The Windows Task manager was able to draw a graph of its RAM usage, but not with absolute values. So for usage to be relevant, all background programs were closed and a baseline b of RAM usage was logged before starting the analysis. The systems memory s was 6 Gigabyte, so the program's RAM usage was calculated using: $(s - A(t)) - b$.

Execution Time At the start of the program and after the execution of each step of the analysis process, the system time was logged. This information was used to determine the execution time of the different stages. Also each event that is logged, will contain a date and time notation.

5.2 Experiment 1: Data Cleaning Module VS. the size of the association matrix

The goal of the data cleaning module is that applying the filters, will cause the number of unique query terms and unique URLs to be smaller. Thereby shrinking the association matrix and increasing its density. The density of the matrix is defined here as:

$$d = \frac{100}{x * y} * n$$

To measure the influence of the tool on the dimensions of the association matrices, several experiments have been conducted. First, each of the Data Cleaning Module's functions were tested separately to determine its effect on the dimensions of the matrix. Second, the different cleaning features were combined in order to maximize the effect. Note that when the combined features were tested, there was no stemming feature applied on the terms, due to unwanted results on non-verbs described earlier. The order in which the cleaning features were applied mattered. The most logical and optimal sequence was to go from global to more specific filters. This resulted in the following sequence:

1. apply the "to lowercase" feature
2. apply the "URL stemming" feature
3. check for and remove "navigational noise"
4. remove "special characters" from the term field

In table 5.2, the results of applying the filters are described with absolute values. Table 5.3 describes the reduction in terms of the original association matrix.

This experiment showed, the implemented filters reduce the matrix to almost $\frac{1}{5}$th of its original size. Also the density increases 3.3 times. But what is maybe

Table 5.2: Absolute reduction of the association matrix

Feature	**Applied On**		**Dimension stats**			
	Terms	**URLs**	**Terms**	**URLs**	**Non-zero**	**Density**
-	-	-	1,151,889	4,975,897	19,571,086	0.000341455
Stemming	x	x	1,080,616	1,858,000	14,888,177	0.000741523
Navigational Noise	x	x	1,027,855	4,886,305	19,363,844	0.000385549
Special Chars	x		1,001,433	4,975,897	19,438,150	0.000390087
To lowercase	x	x	902,249	4,971,990	18,508,191	0.000412579
Combined features			654,236	1,818,875	13,417,242	0.001127524

Table 5.3: Relative reduction of the association matrix

Feature	**Difference with the initial values in %**			
	Terms	**URLs**	**Non-zero**	**Density**
Stemming	-6.19	-62.66	-23.93	217.17
Navigational	-10.77	-1.80	-1.06	112.91
Special Chars	-13.06	0	-0.68	114.24
Lowercase	-21.67	0.08	-5.43	120.83
Combined	-43.20	-63.45	-31.44	330.21

even more importantly, it is done with respect towards the data, trying to keep the semantics intact as much as possible.

5.3 Experiment 2: Data Cleaning Module VS. the environment variables

In this experiment, the focus on lies on the influence of the Data Cleaning Module on the execution time and RAM usage.

5.3.1 Execution time

Table 5.4 shows the measured execution times (h:mm:ss) of the entire process against the applied filters.

Interesting to see is that the filters that have the biggest impact on the size of the association matrix (see table 5.3), also reduce the execution time the most. Replacing the special characters seems the most inefficient filter when comparing the execution time and the filters reduction. However, it still contributes to dimension reduction. This filter's implementation could probably be optimized, however that is left for future work. Most importantly this experiment has shown that cleaning the data first using the cleaning module, speeds up the entire process even though the cleaning itself also costs time.

Table 5.4: Execution time vs Filtering

	Without Filters	**Stemming**	**Nav. Noise**	**Special Chars**	**Lower Case**	**Combined Filters**
Step 1:	0:01:00	0:00:58	0:00:58	0:00:58	0:00:58	0:00:59
Step 2:	0:00:05	0:01:28	0:00:09	0:02:33	0:00:48	0:03:23
Step 3:	0:03:43	0:02:18	0:03:32	0:03:35	0:03:11	0:01:45
Step 4:	0:06:46	0:05:04	0:06:35	0:06:25	0:06:25	0:04:48
Step 5:	0:02:56	0:02:47	0:02:39	0:02:42	0:02:33	0:02:22
Total:	0:14:30	0:12:36	0:13:55	0:16:16	0:13:57	0:13:14

5.3.2 RAM usage

The RAM usage was monitored in a different run, but only for the combined filters. Figure 5.1 shows the total amount of RAM used by the tool after the execution of each step. What can be seen is the tool consumes a almost 4 GB of RAM to process the complete RFP set (see table A.1 for more details).

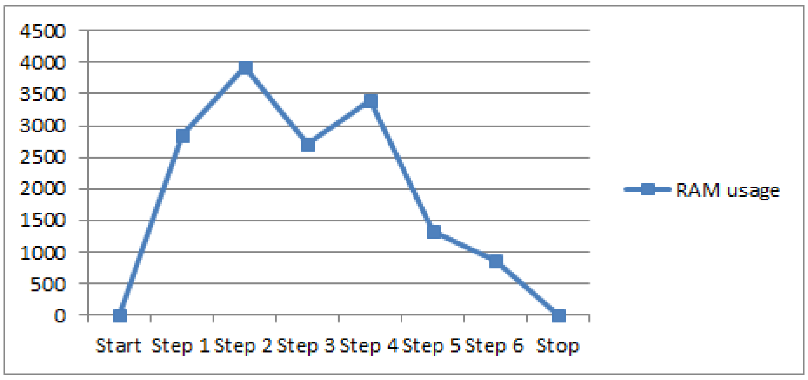

Figure 5.1: RAM usage of the tool in MB

Another graph was created by the Windows Task Manager (see figure A.1) and shows the systems RAM over time. However, no exact values can be deducted from this graph. At first, it was assumed the nice curves and sudden drops in the graph occurred exactly after the completion of one step and used memory was deallocated. To test this, another run of the same experiment was conducted (see figure A.2), with small sleep intervals, so that after each a drop in processor activity was printed in the graph. This showed the peeks and drops could not be related exactly 1:1 to the completion of one step. The most logical explanation is the garbage collector of Visual Studio cleans up the heap memory when it approaches the system's limit. Optimizing the memory usage even further, was left for future work, since the current objective could be completed with the proposed hardware.

5.4 Experiment 3: Execution time VS. the number of records

In the previous experiments, each time the full data set was used. However to see what kind of influence the number of records has on the execution time of the several steps in the process, an experiment was conducted to measure the performance with different quantities of data. The program was used 12 times to perform the conversion from data set to association matrix. Each time the only changing variable was the number of records which was consistently increased with one million. For the Data Cleaning Module, the optimal combination of filters from experiment 1 was used. The results of this experiment are shown in figure 5.2. The raw data that was collected from the log files. The actual numbers can be found in appendix A.2.

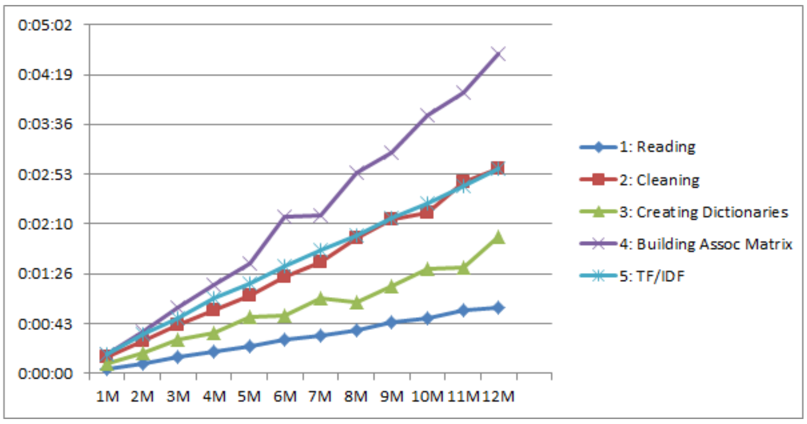

Figure 5.2: The influence of the matrix size on the execution time of the program

The graph shows the overall performance of the tool seems to be quite consistent as the number of data it has to process increases. Here step one and five behave the most consistent. This is probably due to the fact that these steps are not so much influenced by differences in data. Step 4 seems to be the slowest step in the process, so trying to optimize this step would benefit the overall performance the most. When the execution times in the graph are grouped together as a total, they almost form a perfect straight line, showing a linear consistency when the amount of data it has to handle increases. On average, the difference between each step of one million records, is 1 minute and 8 seconds. This makes it easier to predict how the tool will behave when it has to process more data.

5.5 Results

Unfortunately, as already explained in 4.6, using an SVD package from a third party was unsuccessful. Therefore no further experiments could be conducted to test its performance. However the experiments performed, show that the Data Cleaning module has a positive impact in terms of execution time and reduction of the association matrix. Further more, it gives insight in what filters or steps could be optimized to gain performance. Also it is noticed the tool performs consistent in terms of speed.

Chapter 6

Conclusions

6.1 Answering the research questions

In what ways can click data relations be perceived? In chapter 2.1, the possible relationships between two different click data elements have been studied. From the ones that seemed promising, an interpretation has been documented. From there it was determined to further investigate the 'query terms – URL' relation. This relation was found to be the most interesting one in terms of cohesion.

What kind of techniques are required to extract the information from the data set? Several Information Retrieval (IR) techniques have been studied in chapter 2. The Singular Value Decomposition (SVD) has been found to be useful (especially in theory) to extract underlying hidden semantic relations (concepts) from the data. To be able to apply the SVD on the data in practice, the dimensions of the association matrix first needed to be reduced drastically, mainly because of hardware limitations. Therefore several preprocessing filters have been suggested. They do not only decrease the dimensions, but also improve the quality of the data.

How can these techniques be applied to the data set? To be able to apply these techniques on the data set, a tool has been proposed in chapter 3 and its implementation was described in chapter 4. Special attention has been dedicated to account for the numerical stability. Using this tool, the association matrix was eventually reduced to $\frac{1}{5}$th of its original size and it became 3,3 times denser, while keeping the semantics intact as much as possible. The computation of its SVD however, was left to a third party tool, in this case SVDLIBC.

What kind of information is hidden in click data? Answering the main research question however is still rather difficult since the tool is still in a stage where the SVD of the 'query terms – URL' relation for a large matrix is yet to be computed. This was the result of problems with the implementation of SVDLIBC on a more modern computer architecture. However, because preprocessing the data, was a primary condition to be able to compute the SVD of the entire matrix, extra effort has been put into this part of the tool. Eventually a smaller matrix was generated from the data set using the tool and its SVD could be computed. Even though its decomposition might be accurate, the greater goal was to compute the SVD of a much larger matrix.

However, the result of this thesis is the implementation of a tool that can preprocess and (up to a certain level) analyze large data sets in a Windows environment and on a desktop computer. The extendible architecture provides basic IR functionality on custom developed data structures. It's flexible file handler allows to analyze two different dimensions of a random large data set. The Data Cleaning module provides several filters to manipulate the click data in a responsible way. The logging module allows the researcher to view all parameters and different steps that were taken in a certain experiment, so they can be easily reproduced. For computing the SVD, the tool depends on a third party application that when even if it would functioning properly is still known to have several drawbacks. Hopefully in the future a better suited SVD package becomes available, so it can be properly integrated.

6.2 Future work

In the chapter 2.1 the 'query term – URL' relation was chosen for further investigation. However, there are still other relations within click data that are interesting to be studied further and could be analyzed by using the tool. Other future work could be to explore the possibility of reusing data that was removed using the URL stemming filter, to improve domain specific navigation. During the implementation of the tool, some concessions were made in terms of usability since functionally had a higher priority. Because of that, the Graphical User Interface (GUI) is currently console based and thus not very user friendly. Improvements should be made so that the parameters of the filters can be changed more easily. In section 4.5, several suggestions are left unimplemented, like lemmatization. As already mentioned, another major improvement for the tool would be to have a stable implementation of the SVD algorithm incorporated. To speed up the performance of calculating the SVD, this algorithm should be suited for parallel processing an should be able to run on the computer's graphics card (GPU) for example using the CUDA Framework. Currently SVD solvers exist for the CUDA framework however they cannot yet deal with large and sparse matrices.

Bibliography

[1] Zhaojun Bai, James Demmel, Jack Dongarra, Axel Ruhe, and Henk van der Vorst, editors. *Templates for the Solution of Algebraic Eigenvalue Problems: a Practical Guide.* SIAM, Philadelphia, 2000. [cited at p. 15, 22, 26, 38]

[2] M. Berry, T. Do, G. O'Brien, V. Krishna, and S. Varad-han. *SVDPACK (version 1.0) user's guide. Technical Report CS-93-194*, April 1993. [cited at p. 38]

[3] Eamonn Cahill, Alan Irving, Christopher Johnston, and James Sexton. Numerical stability of lanczos methods. *Nuclear Physics B - Proceedings Supplements*, 83-84:825 – 827, 2000. Proceedings of the XVIIth International Symposium on Lattice Field Theory. [cited at p. 16]

[4] Georgi Chulkov. Buglook: A search engine for bug reports. Master's thesis, Jacobs University Bremen, august 2008. [cited at p. 39, 40, 43]

[5] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *An Introduction to Information Retrieval - Online edition.* Cambridge University Press, April 2009. [cited at p. 11, 17, 23]

[6] Philip Ender. Applied categorical and nonnormal data analysis - the process of analyzing data. http://www.gseis.ucla.edu/courses/ed231c/notes2/analyzing.html. last visited on 06-05-2010. [cited at p. 4, 20]

[7] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Professional, 1995. [cited at p. 31]

[8] David Goldberg. What every computer scientist should know about floating point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991. this version is an edited reprint found on: http://www.validlab.com/goldberg/paper.pdf. [cited at p. 15]

[9] G. H. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *JSIAMB*, 2, 1965. [cited at p. 12, 13]

[10] Torsten Grabs, Klemens Böhm, and Hans-Jörg Schek. Powerdb-ir: information retrieval on top of a database cluster. In *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*, pages 411–418, New York, NY, USA, 2001. ACM. [cited at p. 29]

[11] Maarten van der Heijden, Max Hinne, Wessel Kraaij, Suzan Verberne, and Theo van der Weide. Using query logs and click data to create improved document descriptions. In *WSCD '09: Proceedings of the 2009 workshop on Web Search Click Data*, pages 64–67, New York, NY, USA, 2009. ACM. [cited at p. 45]

[12] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms - second edition.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002. [cited at p. 14, 15, 16]

[13] Max Hinne, Wessel Kraaij, Stephan Raaijmakers, Suzan Verberne, Theo van der Weide, and Maarten van der Heijden. Annotation of urls: more than the sum of parts. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 632–633, New York, NY, USA, 2009. ACM. [cited at p. 3, 37, 45]

[14] Ellis Horowitz and Sartaj Sahni. *Fundamentals of data structures in PASCAL.* Computer Science Press, Inc., New York, NY, USA, 1984. [cited at p. 14, 33]

[15] Seikyung Jung, Jonathan L. Herlocker, and Janet Webster. Click data as implicit relevance feedback in web search. *Inf. Process. Manage.*, 43(3):791–807, 2007. [cited at p. 3]

[16] Robert Küffner, Katrin Fundel, and Ralf Zimmer. Expert knowledge without the expert: integrated analysis of gene expression and literature to derive active functional contexts. *Bioinformatics*, 21(2):259–267, 2005. [cited at p. 16, 39, 40, 43]

[17] Sheetal Lahabar and P. J. Narayanan. Singular value decomposition on gpu using cuda. *Parallel and Distributed Processing Symposium, International*, 0:1–10, 2009. [cited at p. 16]

[18] Innes Martin and Joemon M. Jose. A personalised information retrieval tool. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 423–424, New York, NY, USA, 2003. ACM. [cited at p. 3]

[19] B. N. Parlett and J. K. Reid. Tracking the progress of the lanczos algorithm for large symmetric eigenproblems. *IMA Journal of Numerical Analysis*, 1(2):135–155, 1981. [cited at p. 4, 16, 17]

[20] M. F. Porter. An algorithm for suffix stripping. *Readings in information retrieval*, pages 313–316, 1997. ISBN: 1-558,60-454-5. [cited at p. 22, 36]

[21] Vijay V. Raghavan and S. K. M. Wong. A critical analysis of vector space model for information retrieval. *Journal of the American Society for Information Science*, 37(5):279–287, January 1999. [cited at p. 11]

[22] Suzan Verberne, Max Hinne, Maarten van der Heijden, Wessel Kraaij, Eva D'hondt, and Theo van der Weide. Annotating urls with query terms: What factors predict reliable annotations? In *Proceedings of the Understanding the User (UIIR) workshop at SIGIR 2009*, 2009. [cited at p. 45]

[23] Theo van der Weide and Wessel Kraaij. Block a, lecture 4: Vector model. Course on Information Retrieval, februari 2009. [cited at p. 11, 24, 39, 43]

[24] Theo van der Weide and Wessel Kraaij. Block b, lecture 2: Knowledge extraction. Course on Information Retrieval, februari 2009. [cited at p. 13, 36]

[25] Eric W. Weisstein. Roundoff error. From MathWorld–A Wolfram Web Resource http://mathworld.wolfram.com/RoundoffError.html. last visited on 11-03-2010. [cited at p. 14]

Appendices

Appendix A

Data tables

A.1 Data from the experiments

Table A.1: RAM usage of the tool in MB during the execution of the tool

	Total system RAM:	6000	

	Available RAM	**Used System RAM**	**RAM Usage Tool**
Start	4663	1337	0
Step 1	1821	4179	2842
Step 2	738	5262	3925
Step 3	1954	4046	2709
Step 4	1271	4729	3392
Step 5	3335	2665	1328
Stop	4663	1337	0

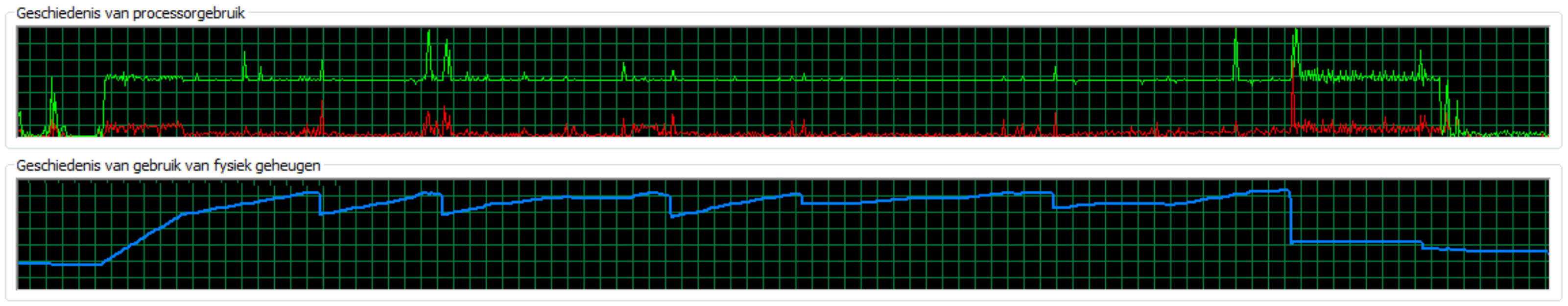
Geschiedenis van processorgebruik
Geschiedenis van gebruik van fysiek geheugen

Figure A.1: System RAM usage

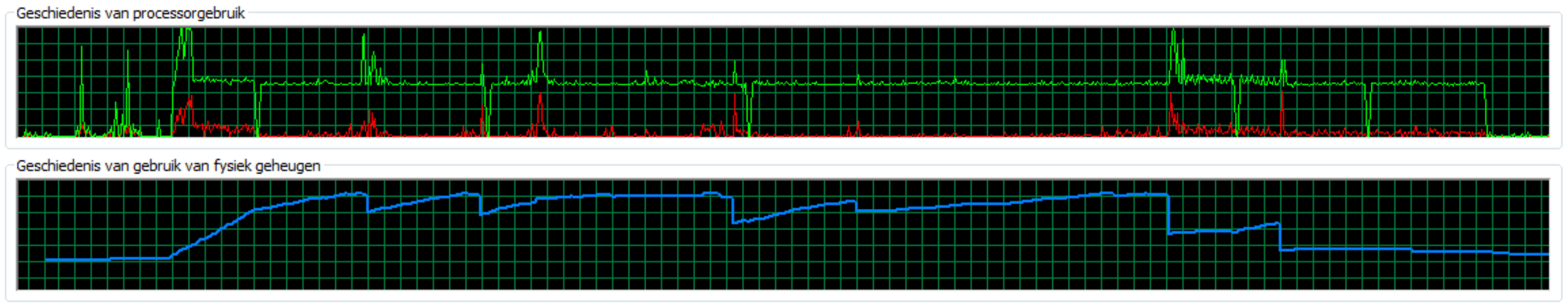
Geschiedenis van processorgebruik
Geschiedenis van gebruik van fysiek geheugen

Figure A.2: System RAM usage with interruptions in the process to mark the different steps in the graph

The green line represents the processor activity of both CPU cores, the red line is the systems kernel time. In figure A.2, there are several drops on the green line* that are caused by a Sleep() function. They each mark the separation of the steps.

*Note that a sixth step is marked, it was used to write the sub matrices to the hard disc

Table A.2: Execution time V.S. Assoc Matrix size

	1M	2M	3M	4M	5M	6M	7M	8M	9M	10M	11M	12M
Step 1:	0:00:04	0:00:09	0:00:14	0:00:19	0:00:24	0:00:29	0:00:33	0:00:38	0:00:44	0:00:48	0:00:55	0:00:57
Step 2:	0:00:14	0:00:28	0:00:42	0:00:55	0:01:07	0:01:24	0:01:37	0:01:57	0:02:13	0:02:19	0:02:46	0:02:58
Step 3:	0:00:09	0:00:18	0:00:29	0:00:35	0:00:49	0:00:50	0:01:05	0:01:02	0:01:16	0:01:31	0:01:32	0:01:58
Step 4:	0:00:16	0:00:36	0:00:57	0:01:17	0:01:35	0:02:16	0:02:17	0:02:54	0:03:12	0:03:44	0:04:03	0:04:37
Step 5:	0:00:16	0:00:34	0:00:48	0:01:05	0:01:18	0:01:33	0:01:47	0:02:00	0:02:15	0:02:28	0:02:42	0:02:57

A.2 Parameters of SVDLIBC

Table A.3: Parameters of SVDLIBC

Usage			
svd options matrix file			
-a	algorithm	Set the algorithm to use. They include:	
		las2	Single-Vector Lanczos Method (default)
-c	infile outfile	Converts a matrix file to a new format (using -r and -w to specify the old and new formats). Then exits immediately.	
-d	dimensions	Desired number of SVD triples or dimensions (default is all)	
-e	bound	Minimum magnitude of wanted eigenvalues for las2 (1e-30)	
-k	kappa	Accuracy parameter for las2 (1e-6)	
-i	iterations	Maximum algorithm settling iterations. By default it iterates as many times as necessary to obtain the desired number of dimensions, and most users will not want to adjust this. But you can set this to a lower value to speed things up, with the possible loss of some dimensions.	
-o	file_root	Root of files in which to store resulting U', S, and V'	
-r	format	Input matrix file format (see below for format specifications)	
		st	Sparse text (default)
		sth	SVDPACK Harwell-Boeing text format
		dt	Dense text
		sb	Sparse binary
		db	Dense binary
-t		Transposes the input matrix. Can be used when computing the SVD or converting the format with -c.	
-v	verbosity	Default is 1. Use 0 for no feedback, 2 to list singular values, and 3 for the vectors too.	
-w	format	Output matrix file format. Options	

A.3 Example Log file

Below is displayed one of the log files created by the tool to get an impression:

```
27-5-2010 17:51:55 ******* Parameters: *******
27-5-2010 17:52:03 data file : h:/-clickdata/datafiles/clickdata.txt
27-5-2010 17:52:05 cols: 1,3
27-5-2010 17:52:07 Separation char: \t
27-5-2010 17:52:09 sample begin: 0
27-5-2010 17:52:12 sample size: 12000000
27-5-2010 17:52:13 separate nav data?: True
27-5-2010 17:52:14 dictionary files to be written to disk?: True
27-5-2010 17:52:14 abstract matrix to be written to disk?: True
27-5-2010 17:52:14 write the matrix in Sparse Text Format to disk?: True
27-5-2010 17:52:14 ******* Begin Analysis: 27-5-2010 17:52:14 *******
27-5-2010 17:52:14 Step 1: Reading dataset to internal format
27-5-2010 17:53:12 Done reading
27-5-2010 17:53:12 Step 2: Started cleaning the data
27-5-2010 17:56:10 The data has been cleaned
27-5-2010 17:56:10 Writing navigational data to disk:
                   H:/-ClickData/Datafiles/27-5-2010 17.51.45_navdata.txt
27-5-2010 17:56:11 Step 3: Creating dictionaries
27-5-2010 17:57:44 Dictionaries created with: 587801 unique terms and 1797799 URLS
27-5-2010 17:57:44 Writing term dictionary to disk:
                   H:/-ClickData/Datafiles/27-5-2010 17.51.45_td.txt
27-5-2010 17:57:47 Writing URL dictionary to disk:
                   H:/-ClickData/Datafiles/27-5-2010 17.51.45_ud.txt
27-5-2010 17:58:00 Writing abstract data file to disk:
                   H:/-ClickData/Datafiles/27-5-2010 17.51.45_asdf.txt
27-5-2010 17:58:09 Step 4: Creating the association matrix
27-5-2010 18:01:53 Writing association matrix to file
                   H:/-ClickData/Datafiles/27-5-2010 17.51.45_assocm.txt
27-5-2010 18:03:19 Matrix Dimensions: 1797799 x 587801 with 12892808 nonzero values
27-5-2010 18:03:19 Step 5: Determining TF/IDF values for the association matrix
27-5-2010 18:03:26 Performing Cosine normalisation
27-5-2010 18:03:28 Ready with TF/IDF
27-5-2010 18:03:28 Writing matrix to disk in Sparse Text Format
27-5-2010 18:04:02 Step 6: Writing Submatrices to disk
27-5-2010 18:04:02 Cols: 29390 Rows: 89889 Non Zero Values: 2660012
27-5-2010 18:04:11 Cols: 58780 Rows: 179779 Non Zero Values: 4063828
27-5-2010 18:04:23 Cols: 117560 Rows: 359559 Non Zero Values: 5930871
27-5-2010 18:04:39 Cols: 176340 Rows: 539339 Non Zero Values: 7282543
27-5-2010 18:04:59 Cols: 235120 Rows: 719119 Non Zero Values: 8385327
27-5-2010 18:05:22 Cols: 293900 Rows: 898899 Non Zero Values: 9333736
27-5-2010 18:05:45 Done writing Submatrices
27-5-2010 18:05:45 **************
27-5-2010 18:05:45 Done!
27-5-2010 18:05:45 **************
27-5-2010 18:05:45 Step1 took: 1-1-0001 0:00:57
27-5-2010 18:05:45 Step2 took: 1-1-0001 0:02:58
27-5-2010 18:05:45 Step3 took: 1-1-0001 0:01:58
27-5-2010 18:05:45 Step4 took: 1-1-0001 0:04:37
27-5-2010 18:05:45 Step5 took: 1-1-0001 0:02:57
27-5-2010 18:05:45 **************
27-5-2010 18:05:45 Total time: 1-1-0001 0:13:30
27-5-2010 18:05:45 **************
```

List of Figures

2.1 Possible relations inside click data 9
2.2 Singular Value Decomposition 12
2.3 Reduced association matrix 13

3.1 UML Domain Class Diagram 21

4.1 Data Analysis Process as implemented in the tool. 28
4.2 A three-tier structure. 29
4.3 Architecture of the analysis tool. 31
4.4 Orthogonal list 34
4.5 Density of the partial matrices 41
4.6 Density of the sub matrices 41

5.1 RAM usage of the tool in MB 49
5.2 The influence of the matrix size on the execution time of the program 50

A.1 System RAM usage 62
A.2 System RAM usage with interruptions in the process to mark the different steps in the graph 62

List of Tables

2.1 A sample of click data from the RFP data set 7
2.2 numerical precision experiment 15

4.1 Sample matrix after TF–IDF weighting and cosine normalization. 39
4.2 Singular Values calculated by ALGLIB and SVDLIBC 40
4.3 Specifications of several sub matrices 41

5.1 Used hardware 46
5.2 Absolute reduction of the association matrix 48
5.3 Relative reduction of the association matrix 48
5.4 Execution time vs Filtering 49

A.1 RAM usage of the tool in MB during the execution of the tool 61
A.2 Execution time V.S. Assoc Matrix size 63
A.3 Parameters of SVDLIBC 64