Examensarbete

Graph Similarity, Parallel Texts, and Automatic Bilingual Lexicon Acquisition

Tobias Törnfeldt

LiTH - MAT - EX - - 08 / 03 - - SE
Graph Similarity, Parallel Texts, and Automatic Bilingual Lexicon Acquisition

Department of Mathematics, Linköpings Universitet

Tobias Törnfelt

LiTH - MAT - EX - - 08 / 03 - - SE

Examensarbete: 45 hp

Level: D

Supervisor: Lars Eldén,
Department of Mathematics, Linköpings Universitet

Co-supervisor: Lars Ahrenberg,
Department of Computer and Information Science, Linköpings Universitet

Examiner: Lars Eldén,
Department of Mathematics, Linköpings Universitet

Linköping: April 2008
In this masters’ thesis report we present a graph theoretical method used for automatic bilingual lexicon acquisition with parallel texts. We analyze the concept of graph similarity and give an interpretation, of the parallel texts, connected to the vector space model. We represent the parallel texts by a directed, tripartite graph and from here use the corresponding adjacency matrix, $A$, to compute the similarity of the graph. By solving the eigenvalue problem $\rho S = ASA^T + A^TSA$ we obtain the self-similarity matrix $S$ and the Perron root $\rho$. A rank $k$ approximation of the self-similarity matrix is computed by implementations of the singular value decomposition and the non-negative matrix factorization algorithm GD-CLS. We construct an algorithm in order to extract the bilingual lexicon from the self-similarity matrix and apply a statistical model to estimate the precision, the correctness, of the translations in the bilingual lexicon. The best result is achieved with an application of the vector space model with a precision of about 80%. This is a good result and can be compared with the precision of about 60% found in the literature.
Abstract

In this masters’ thesis report we present a graph theoretical method used for automatic bilingual lexicon acquisition with parallel texts. We analyze the concept of graph similarity and give an interpretation, of the parallel texts, connected to the vector space model. We represent the parallel texts by a directed, tripartite graph and from here use the corresponding adjacency matrix, \( A \), to compute the similarity of the graph. By solving the eigenvalue problem \( \rho S = ASA^T + A^TSA \) we obtain the self-similarity matrix \( S \) and the Perron root \( \rho \). A rank \( k \) approximation of the self-similarity matrix is computed by implementations of the singular value decomposition and the non-negative matrix factorization algorithm GD-CLS. We construct an algorithm in order to extract the bilingual lexicon from the self-similarity matrix and apply a statistical model to estimate the precision, the correctness, of the translations in the bilingual lexicon. The best result is achieved with an application of the vector space model with a precision of about 80%. This is a good result and can be compared with the precision of about 60% found in the literature.

**Keywords:** Parallel texts, graph similarity, bilingual lexicon, SVD, ARPACK, NMF, OpenMP, text mining.
Acknowledgements

First of all, I would like to thank my supervisor professor Lars Eldén for presenting this project to me and for his perceptivity in the area of data mining and numerical linear algebra and his tremendous support throughout the project.

I would also want to thank my co-supervisor professor Lars Ahrenberg for providing me with the parallel texts and for providing his knowledge about linguistics and semantics which has been a great contribution to the project.

The National Supercomputer Centre at Linköping, NSC gave me the opportunity to perform my tests at Mozart and Monolith. For this I am very grateful, without NSC many of the computations would not have been possible to carry out.

My opponent Johan Gyllensten deserves a great thank for his careful reading of the report and for giving me suggestions of how to improve the work.

Lastly I would like to thank my family and my beloved girlfriend Sofia for her understanding and support.
Notation

Most of the recurring abbreviations, operators, functions and symbols are described here.

Symbols

\[ \mathbf{A} \] is a matrix denoted with bold face upper-case letters
\[ \mathcal{G} \] is a graph denoted with Computer Modern calligraphic letters
\[ \mathcal{M} \] is an operator denoted with Computer Modern calligraphic letters
\[ v_i \text{ or } v_j \] is the row vector \( i \) or the column vector \( j \) in a matrix
\[ \mathbb{R}^{m \times n} \] is the real \( m \times n \) vector space
\[ \mathbb{C}^{m \times n} \] is the complex \( m \times n \) vector space
\[ \mathbf{A}^{m \times n} \] is a real or complex matrix with dimension \( m \times n \)

Operators and functions

\[ \text{vec} \] Vectorization of a matrix
\[ \text{rank} \] The rank of a matrix
\[ \text{diag} \] The diagonal of a matrix
\[ \text{span} \] The subspace spanned by a matrix
\[ \otimes \] The Kronecker or tensor product

Abbreviations

\[ \text{TMG} \] Text to Matrix Generator
\[ \text{GTP} \] General Text Parser
\[ \text{SVD} \] Singular Value Decomposition
\[ \text{CCS} \] Compressed Column Storage
\[ \text{ARPACK} \] ARnoldi PACKage
\[ \text{NMF} \] Non-negative Matrix Factorization
\[ \text{GD-CLS} \] Gradient Descent with Constrained Least Squares
\[ \text{LAPACK} \] Linear Algebra PACKage
\[ \text{IRAM} \] Implicit Restarted Arnoldi Method
\[ \text{VSM} \] Vector Space Model
\[ \text{SS-SVD} \] Self-Similarity computations by SVD
\[ \text{VSM-SVD} \] SVD of the Vector Space Model
\[ \text{VSM-NMF} \] NMF of the Vector Space Model

Törnfeldt, 2008.
Contents

1 Introduction 1
   1.1 Background ........................................... 1
   1.2 Outline of the report ................................. 1

2 Preprocessing and Text Parsing 3
   2.1 Preprocessing of Parallel Texts ................... 3
   2.2 Text Parsing and the Vector Space Model .......... 4
       2.2.1 The Vector Space Model ......................... 5

3 Graph Theory and Graph Similarity 7
   3.1 A Graph Theoretical Approach ....................... 7
   3.2 Hub and Authority Scores ........................... 10
   3.3 Graph Similarity .................................... 12
       3.3.1 The Similarity Matrix Operator .............. 16
   3.4 Self-similarity ..................................... 17
   3.5 Self-similarity of Parallel Texts .................. 18
   3.6 Subgraphs and Rank $k$ Approximations ............ 23
   3.7 The Automatic Bilingual Lexicon Acquisition from $S_{12}$ ................ 29
       3.7.1 Verifying the Translations, Semantics of Swedish and English Words .................... 30
       3.7.2 An Interface with MATLAB, the Bilingual Lexicon and a Statistical Model ............. 31

4 The Computation by Singular Value Decomposition 33
   4.1 The Data Set and some Preprocessing issues ......... 33
   4.2 Sparse Matrices and Data Matrix Formats ............ 34
       4.2.1 The Harwell–Boeing Format .................. 34
       4.2.2 The Compressed Column Storage ............... 35
   4.3 A Krylov Decomposition Method ..................... 37
       4.3.1 Reverse Communication ........................ 39
   4.4 The SVD Implementation with ARPACK ............... 40
       4.4.1 The Implementation of the Automatic Bilingual Lexicon Acquisition Algorithm ........ 42
   4.5 Tests and Results .................................. 42
       4.5.1 Test 1 with SS-SVD ............................. 42
       4.5.2 Test 2 with VSM-SVD ............................ 44
       4.5.3 Test 3 with VSM-SVD ............................ 46
   4.6 Discussion of the Results .......................... 50

Törnfeldt, 2008. xiii
5 The Computation by Non-negative Matrix Factorization 51
  5.1 The Non-negative Matrix Factorization Problem .............. 51
  5.2 The GD-CLS Algorithm .................................... 53
  5.3 The Implementation of the GD-CLS Algorithm .............. 54
    5.3.1 Parallelization with OpenMP ......................... 56
  5.4 Test and Result with VSM-NMF ............................ 59
  5.5 Discussion of the Results ............................... 60

6 Comparison of the Results, Summary and Conclusions 61
  6.1 Comparison of the Results ............................... 61
  6.2 Summary and Conclusions ................................ 62
  6.3 Future Work and Improvements ........................... 64

A Perron-Frobenius Theory .................................. 67

B Theorems and Proofs ........................................ 69

C Details of the Preprocessing and the Text Parsing of the Text Collections 77

D The Euclidean Vector Norm, the Frobenius Matrix Norm and the Trace Operator 79

E Text Parsing Tools: TMG and GTP .......................... 83
  E.1 TMG: Text to Matrix Generator .......................... 83
  E.2 GTP: General Text Parser ............................... 83

F A Random Sample from the Bilingual Lexicon ................ 85
Chapter 1

Introduction

1.1 Background

In the area of text mining there are several methods for extracting useful information from large text collections. These text collections, or corpora, could for example be a collection of medical papers or an electronic mail collection. The methods used for extracting information from these corpora depend on the problem, and the structure of the text. One application, among others developed in a study by M. Sahlgren and J. Karlgren [1], is automatic bilingual lexicon acquisition.

The need for a good functioning multilingual lexicon is big particularly in the EU and the European Commission, where translations of text documents are done on a daily basis. There are, of course, manually constructed multilingual lexicon which give high precision but these are hard to maintain. Therefore one wishes to develop automatic techniques to generate a multilingual lexicon.

In the study of Sahlgren and Karlgren they present the concept of parallel texts and describe the method of random indexing to generate an automatic bilingual lexicon. The results of this study show an overlap of around 60% i.e. the correctness, of the overall translations, is of about 60%.

The aim of this master’s thesis project is to investigate some other techniques to generate a bilingual lexicon. We will use a graph theoretical approach and the notion of graph similarity [2] to analyze the problem with linear algebra and from here give suggestions of algorithms that can be used.

1.2 Outline of the report

Chapter 2: Before we can apply any mathematical algorithms on a text collection we need to convert the text data into matrix data. An introduction of how to process the text collections is presented here. A method that often occurs when analyzing the documents in a text collection is the vector space model and in the end of this chapter there is an explanation of the model and some of the commonly used applications.

Chapter 3: The parallel texts can be represented by a graph and the chapter contains the theory behind this approach. We analyze the concept of
graph similarity and how it can be applied to the problem. The main focus is on solving the problem with parallel texts and why the solution gives a poor result in the end. Evaluation of the solution elicit an answer of how to proceed.

**Chapter 4:** The solution is derived from the use of singular value decomposition, SVD, and therefore the first algorithm is based on it. The parallel texts are represented by a large sparse matrix and it is necessary to have a matrix data format that adapts well with the implementation. A brief explanation of the theory behind the main part of the algorithm is represented. To extract the bilingual lexicon from the solution by SVD we need an algorithm and a statistical model to derive the precision of the translations in the lexicon. In order to verify the translations we set up some rules based on the structure and semantics of the two languages. The end of the chapter is devoted to tests and results of the algorithm and a concluding discussion.

**Chapter 5:** The data matrix is non-negative and it suggests the use of a non-negative matrix factorization, NMF, to solve the problem. There exists many different algorithms to solve this problem and we use one which takes the sparsity of the data matrix in account. In the middle of the chapter there is a description of the serial program and, in detail, how it can be parallelized by OpenMP. The last section contains tests and results and ends with a discussion of the obtained results.

**Chapter 6:** In this last chapter we aim to compare the results obtained from the computations with SVD and NMF and a summary and conclusions are presented here. Finally a discussion concerning the future work and some possible improvements are given.

**Appendix** Appendix A contains details about the Perron-Frobenius theory. Appendix B is giving a generalization of the theorems and propositions used in the report. Appendix C describes some more details about the preprocessing and text parsing of the parallel texts. Appendix D contains some theory concerning the 2-norm of a vector, the Frobenius norm of a matrix and some properties of the trace operator. Appendix E present the text parsing tools TMG and GTP and how they are used. Finally the Appendix F is listing a random sample of an automatic bilingual lexicon.
Chapter 2
Preprocessing and Text Parsing

2.1 Preprocessing of Parallel Texts

Parallel texts are a text collection, or corpus, translated into two different languages, e.g. a manual for an electronic device or in our case, which we will come back to later, the minutes of an European parliament meeting.

Before we can use any mathematical method or algorithm to generate a bilingual lexicon from the parallel texts we have to convert the text data into matrix data which is done by preprocessing of the text and text parsing. The main part of the preprocessing step consists of stemming and removal of stop words.

There are different kinds of stemming and the primary goal is to reduce a word to its stem. One common type of stemming, well developed for the English language, is Porter stemming which reduce every word to its stem by suffix stripping. Let us consider an example:

**Example 2.1.** Assume that we would like to stem the first main clause of this section:

> Parallel texts are a text collection, or corpus, translated into two different languages.

If we use a Porter stemmer then the outcome is:

> parallel text are a text collect, or corpus, translat into two differ

We see the effect of suffix stripping on the underlined words above and also that upper-case letters are transformed into lower-case letters.

The advantage of this method is that it is easy to implement and there are developed software free to use \(^1\). A drawback of the Porter stemming algorithm becomes clear when trying to stem a Swedish sentence.

\(^1\)http://snowball.tartarus.org/
Example 2.2. The Swedish sentence:

Katten jagar råttan.\(^2\) → katt jag råttan

The Swedish word “jagar” means “chasing”, but the word “jag” means “I” and that word will later be put in a stop list (words that we do not want to be parsed).

This small example shows that a word with a suffix, “jag-ar”, can be stemmed to another word with a different meaning. To overcome this problem we will use a stemming algorithm which takes a word and stem it to its possible base form.\(^3\)

Example 2.3. If we use this method on the Swedish sentence in example 2.2 the outcome is:

katt jaga rått

The underlined word “jaga” means “chase” i.e. the meaning of the word has not changed.

With this method we are able to stem the Swedish and English words respectively without loosing the actual meaning of the words. The underlying theory of the algorithms used by this method is complex and demands knowledge in linguistics and semantics.

A stop list is a list of common words that do not bring any real meaning to a sentence e.g. prepositions, pronouns, conjunctions, numbers etc. If we consider the sentence in the example 2.1, a sufficient stop list would be:

a, is, or, into, two.

The size of the stop list will naturally increase with the size of the text collection and the contents of the stop list depend on the contents of the text collection. The stop lists we use contain about 700 - 800 words due to that in the parallel texts there are many numbers/years, special characters and words combined by numbers and letters e.g. a12, b5, k2r4.

Example 2.4. The final result after stemming and removal of stop words of the sentence in example 2.1:

parallel text text collection corpus translate different language.

As we can see in the example above the key words of the sentence are preserved and possibly stemmed. If we apply this preprocessing procedure to the parallel texts we can now go to the next step; text parsing.

2.2 Text Parsing and the Vector Space Model

The purpose of text parsing is to convert a text collection into a term-by-document matrix \( \mathbf{A} \), which is a (in many times sparse) matrix where each row of \( \mathbf{A} \) represents a term, i.e. a word not in the stop list, and each column of \( \mathbf{A} \)

\(^2\)“The cat is chasing the rat.”

\(^3\)http://www.connexor.eu/technology/machinese/demo/tagger/
represents a document in the document collection. When using a text parser there is usually a requirement that each text document in the text collection is separated by a delimiter. This delimiter could be any character, but a common and an intuitive delimiter to separate the text documents with is an empty line.

To construct a simple term-by-document matrix the term frequencies of each document in the document collection are computed and stored. In many applications this is not sufficient because there is no relation between the term frequency in one document and the term frequency in another document. To solve this problem one have developed different weighting schemes for computation of the local and global frequencies.

The local frequency is, as above, defined by the term frequency \( f_{ij} \) i.e. the frequency of term \( i \) in document \( j \) and could be modified by e.g. a log or a binary operator. A common weighting scheme for the global frequency is \( idf \); inverse document frequency, denoted by:

\[
g_{ij} = f_{ij} \log \left( \frac{n}{n_i} \right)
\]

where \( n \) is the total number of documents and \( n_i \) is the number of documents where term \( i \) appears. This weighting scheme generates a small weight \( g_{ij} \) if a term appears in many documents but only appears a few times in each document, and it generates a large weight for the opposite case i.e. if a term appears many times in a small amount of documents. If the text documents are varying much in size e.g. from ten terms to hundreds of terms it is common to normalize the term frequency weight by dividing the frequency of the term in the document by the sum of all terms in the document. There are various types of global weighting schemes and which one to apply depends of the structure and the context of the text collection [4].

Apart from the term-by-document matrix, generated by a text parser, a dictionary is created. This dictionary contains all the words parsed by the text parser and it is necessary to have for any further analysis of the text collection, in particular for parallel texts and translations.

### 2.2.1 The Vector Space Model

The purpose of the vector space model [17, 13] is to represent a document \( d_i \) with a vector where each non-zero element in the vector corresponds to a weighted term e.g. a word in the document. If there are more than one document then it is possible to represent these documents by a term-by-document matrix generated from a text parser. The space spanned by the term-by-document matrix is called the document space. To measure how close a document \( d_i \) is to a document \( d_j \) it is common to apply the cosine of angle \( t_{ij} = \cos \theta(d_i, d_j) \) (3.10). If \( t_{ij} \) is close to one then we know that these document vectors point in almost the same direction, i.e. the documents \( d_i \) and \( d_j \) are likely to have many essential terms in common. If \( t_{ij} \) is nearly orthogonal then we can conclude that these documents do not have much in common.

The applications of the vector space model are several and one example is query matching which is a method for extracting the documents in a text collection which are most related to a query. The query could be a search phrase or a list of key words and it is represented by a column vector with the same dimension as the documents in the generated term-by-document matrix.
The query is then matched to each document in the document space by applying a suitable measure, e.g. the cosine of angle.

Another related application to query matching is clustering of the documents, in the document space. If the documents in a text collection are clustered by topics or contents it is much more likely to retrieve relevant documents when applying query matching. A standard algorithm for clustering a term-by-document matrix is the \textit{k-means algorithm}. For each of the \textit{k} clusters the algorithm creates a \textit{centroid} cluster and then aim to minimize the distance between the document vectors and the centroids. A more detailed description of the algorithm can be found in [13].
Chapter 3

Graph Theory and Graph Similarity

In the paper [1] the parallel texts are represented by a source text matrix $A_s$ and a target text matrix $A_t$. These matrices are the term-by-document matrices, generated from a text parser, for the two selected languages. Then by aligning each document in the source text with the documents in the target text we get the following matrix block structure:

$$
\begin{pmatrix}
A_s \\
A_t
\end{pmatrix}
= \begin{pmatrix}
A_1 \\
A_2
\end{pmatrix}
= A_p.
$$

(3.1)

This approach form a part of the latter graph theoretical approach.

3.1 A Graph Theoretical Approach

The parallel texts can be represented by a graph, and in order to understand this graph representation we have to consider some definitions [7].

**Definition 3.1.** A graph $\mathcal{G}(E,V)$ with edge set $E$ and vertex set $V$ is called a directed graph if each edge $e_{ij} \in E$ has a given direction from the vertex $v_i \in V$ to the vertex $v_j \in V$. The edges in a directed graph are denoted by arrows pointing in the given directions.

An undirected graph $\mathcal{G}(E,V)$ is a graph where each edge $e_{ij} \in E$ does not point in any given direction. The edge in a undirected graph is denoted by a simple line.

**Definition 3.2.** A simple graph $\mathcal{G}(E,V)$ is a graph with edge set $E$ and vertex set $V$ such that there are no loops and any two vertices in $V$ are connected by at most one edge in $E$.

**Example 3.1.** The left graph in the figure below is a simple graph and the right is not simple.
Definition 3.3. A graph $G(E, V)$ is called bipartite if there exists a bipartition $(X, Y)$ of the vertex set $V$ in $G$ such that every edge in $(X, Y)$ has one end in $X$ and one end in $Y$.

Example 3.2. A bipartite graph.

By definition a bipartite graph does not contain any loops, but it is possible that any two vertices can be connected by more than one edge, thus a bipartite graph does not have to be simple (see example 3.2 above). If we consider a text collection we can represent it with a bipartite graph where each term corresponds to the vertices in the vertex set $X$ of the first partition and the documents corresponds to the vertices in the vertex set $Y$ of the second partition. If the document $d_j$ contain a word $w_i$ then there is an edge $e_{ij}$ connecting them i.e. they are adjacent.

Definition 3.4. The adjacency matrix of a graph $G(E, V)$ and $v_i \in V, i = 1, \ldots, n$ is a $n \times n$ matrix $A(G) = (a_{ij})$, where $a_{ij}$ is the number of edges connecting $v_i$ and $v_j$.

Example 3.3. Assume that a text collection with documents $d_j, j = 1, 2, 3$ and words $w_i, i = 1, 2, 3, 4, 5$ can be represented by the bipartite graph below.

Then the adjacency matrix is

$$
\begin{bmatrix}
0 & A \\
A^T & 0
\end{bmatrix}
$$
where the matrix $A$ is

$$A = \begin{pmatrix}
2 & 0 & 1 \\
0 & 1 & 1 \\
1 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 0
\end{pmatrix}$$

$w$ and $d$ denotes the words and documents respectively.

For an arbitrary text collection the adjacency matrix will have the same structure as the adjacency matrix (3.2) above, but with a different matrix $A$. We can consider the parallel texts as two text collections merged together and they can be represented by a bipartite graph. This approach will not be sufficient for translations of words, because we have to separate the words in the first language from the words in the second language in order to generate a bilingual lexicon. Therefore a modification of the model is needed, by letting the words from the second language constitute a third partition in the graph, we get a tripartite graph in which the words from the two different languages are separated.

**Definition 3.5.** A $k$-partite graph is a graph $G(E, V)$ in which the vertex set $V$ can be partitioned into $k$ subsets such that no edge in $E$ has both ends in any one subset.

**Proposition 3.1.** Any bipartite graph $G(E, V)$ with the bipartition $(X, Y)$ of the vertices $V$, is by structure a $k$-partite graph, where $k \leq |V|$.

**Proof 3.1.** The graph $G(E, V)$ is a bipartite graph with the bipartition $(X, Y)$ of the vertices $V$. Then we can conclude that $G(E, V)$ does not contain any loops, since it is bipartite. If $|V| \geq 3$ and $k = 3$ then according to the pigeonhole principle, one of the partitions $X$ or $Y$ contain more than one vertex, e.g. $|X| = 2$. Assume, in this case, that we have $|X| = 2$ then we can take one of the vertices in $X$ and build a new partition consisting of this vertex, i.e. we have now constructed a tripartite graph.

For $k = 3$ we have shown that it is possible to construct a tripartite graph with tripartition $(X_1, X_2, X_3)$ from the bipartite graph $G(E, V)$. If $|V| \geq 4$ we can construct a 4-partite graph from the tripartite graph, since according to the pigeonhole principle one of the partitions $X_i \in (X_1, X_2, X_3)$ contain more than one vertex. Take one of the vertices in $X_i$ and build a new partition consisting of this vertex. We now have a 4-partite graph and we can continue this process of constructing new partitions until $k = |V|$.

**Example 3.4.** Assume that the parallel texts with documents $d_j$, $j = 1, 2, 3$ and words $w^{(1)}_i$, $i = 1, 2, 3, 4, 5$ and $w^{(2)}_i$, $i = 1, 2, 3, 4$ can be represented by the tripartite graph below.
Then the adjacency matrix is

\[ \begin{bmatrix} \mathbf{w}^{(1)} & \mathbf{w}^{(2)} & \mathbf{d} \\ \mathbf{w}^{(1)} & 0 & 0 \\ \mathbf{w}^{(2)} & 0 & 0 \\ \mathbf{d} & \mathbf{A}_1^T & \mathbf{A}_2^T \end{bmatrix} \]

\[ = \begin{bmatrix} \mathbf{0} & \mathbf{A}_1^T & \mathbf{A}_2^T \\ \mathbf{0} & 0 & 0 \end{bmatrix} \]

(3.3)

where the matrices \( \mathbf{A}_1 \) and \( \mathbf{A}_2 \) are

\[
\mathbf{A}_1 = \begin{pmatrix} 2 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}
\]

and

\[
\mathbf{A}_2 = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix} .
\]

The variables \( \mathbf{w}^{(1)} \) and \( \mathbf{w}^{(2)} \) denote the words from the first and the second language respectively and \( \mathbf{d} \) denotes the documents. The matrix to the right of the equality is the adjacency matrix formed by the parallel text approach in (3.1).

This example states how to represent the parallel texts with a tripartite graph and an adjacency matrix with the structure as the matrix (3.3) above. The matrices \( \mathbf{A}_1 \) and \( \mathbf{A}_2 \) corresponds to the two term-by-document matrices generated from e.g. GTP or another text parser. The tripartite graph representing the parallel texts with the term-by-document matrices is simple, but each edge has instead a weight corresponding to the frequencies derived from the frequency weighing scheme used by the parser.

### 3.2 Hub and Authority Scores

In the graph of the parallel texts we would like to rank the vertices in order to sort out “important” terms and documents. To define importance in this sense we consider the analogy with HITS [25] which is a method to rank web pages
3.2. Hub and Authority Scores

on the Internet. The Internet can be considered as a giant graph where each page is a vertex and the in and out links of a page are the edges. The page rank of a web page depends on the links pointing to the page and pointing out of the page. If we only consider the links pointing to a page as a measure of the rank we get an incorrect ranking system, due to it is easy to manipulate with e.g. link farms. HITS which stands for Hypertext Induced Topic Search, is a method that ranks a web page from two different scores; hub and authority score. These scores are updated simultaneous according to a mutual reinforcement principle which states:

A web page should have a high hub score if it points to many pages with high authority score. Conversely, a web page should have a high authority score if it is pointed to by pages having a high hub score.

Let \( G_S(E, V) \) be a directed subgraph of the Internet with the adjacency matrix \( B \) then define \( h_i \) as the hub score of vertex \( i \) and \( a_j \) as the authority score of vertex \( j \). The two scores are proportional to each other and derived as follows:

\[
\begin{align*}
    h_i &\propto \sum_{j=1}^{n} b_{ij} a_j, \quad i = 1, \ldots, m \\
    a_j &\propto \sum_{i=1}^{m} b_{ij} h_i, \quad j = 1, \ldots, n.
\end{align*}
\]

If we assign a vector \( h \in \mathbb{R}^m \) for all hub scores and a vector \( a \in \mathbb{R}^n \) for all authority scores and also let \( \lambda_h \) and \( \lambda_a \) be proportionality constants then the above equations can be expressed as:

\[
\begin{align*}
    \lambda_h h &= B a \quad \text{(3.4)} \nonumber \\
    \lambda_a a &= B^T h \quad \text{(3.5)}
\end{align*}
\]

The edges in the parallel text graph are not directed, but if we consider the terms to be the building blocks of the documents then it follows naturally that vertices corresponding to terms should point at the vertices corresponding to documents. A directed graph constructed from example 3.4 would then be:

```
with the adjacency matrix
```
This graph and matrix representation is the one we are going to use in the rest of the report and just by looking at the graph we can conclude that the vertices \( w^{(1)}_i \) and \( w^{(2)}_j \) have more in common with each other than either \( w^{(1)}_i \) and \( d_1, d_2, d_3 \) or \( w^{(2)}_j \) and \( d_1, d_2, d_3 \), due to the link structure. The vertices \( w^{(1)}_i \) and \( w^{(2)}_j \) only have outgoing links while \( d_1, d_2, d_3 \) only have incoming links, i.e. \( w^{(1)}_i \) and \( d_1, d_2, d_3 \) or \( w^{(2)}_j \) and \( d_1, d_2, d_3 \) do not match each other very well.

The hub and authority scores can now, with this interpretation of the parallel text graph, be computed by solving the eigenvalue problems generated from (3.4) and (3.5).

### 3.3 Graph Similarity

Now when we know how to rank the vertices in a directed graph we would like to have a measure that compares the vertices in two graphs and describes how similar they are. One method presented in [2] apply the theory of hub and authority score and generalise it to derive the similarity scores between two graphs. The main idea is based on that a vertex \( u \) in a graph \( G_A \) is similar to a vertex \( v \) in a graph \( G_B \) if the adjacent vertices to \( u \) and \( v \) are similar.

Before we can transform this approach into a mathematical expression we have to go through some definitions and propositions [21, 29] concerning the Kronecker product, the inner product of vectors and matrices, the cosine of angle, and a theorem by Perron-Frobenius.

**Definition 3.6.** The Kronecker or tensor product \( \otimes \) of two matrices \( A = A_{m \times n} \) and \( B = B_{p \times q} \) is the \( mp \times nq \) matrix:

\[
A \otimes B = \begin{pmatrix}
    a_{11}B & a_{12}B & \cdots & a_{1n}B \\
    a_{21}B & a_{22}B & \cdots & a_{2n}B \\
    \vdots & \vdots & \ddots & \vdots \\
    a_{m1}B & a_{m2}B & \cdots & a_{mn}B
\end{pmatrix}
\]

**Proposition 3.2.** If \( A \) and \( B \) are matrices then:

i) \( (A \otimes B)^T = A^T \otimes B^T \)

If \( C \) and \( D \) are matrices such that the matrix products \( AC \) and \( BD \) are possible then:

ii) \( (A \otimes B)(C \otimes D) = AC \otimes BD \)
3.3. Graph Similarity

Proof 3.2. (i) By a simple computation we get

\[(A \otimes B)^T = (A \otimes B)_i^T = (a_{i1}B \ a_{i2}B \ \cdots \ a_{in}B)^T =
\]

\[= \begin{pmatrix}
    a_{i1}B^T \\
a_{i2}B^T \\
    \vdots \\
a_{in}B^T
\end{pmatrix} = A_i^T \otimes B^T, \ \forall \ i
\]

\[\Rightarrow (A \otimes B)^T = A^T \otimes B^T.
\]

(ii) Let

\[(A \otimes B)_i = (a_{i1}B \ a_{i2}B \ \cdots \ a_{ip}B)
\]

and

\[(C \otimes D)_j = (c_{1j}D^T \ c_{2j}D^T \ \cdots \ c_{pj}D^T)^T
\]

be an arbitrary row and column in the matrices \((A \otimes B)\) and \((C \otimes D)\) respectively then the matrix multiplication \((A \otimes B)(C \otimes D)\) is:

\[(A \otimes B)_i \ (C \otimes D)_j = \sum_{k=1}^{p} a_{ik}c_{kj}BD = (AC)_{ij} \otimes BD, \ \forall \ i, j
\]

\[\Rightarrow (A \otimes B)(C \otimes D) = AC \otimes BD.
\] \qed

Definition 3.7. Let \(A_{m \times n}\) be a matrix, then the vectorization of \(A\), denoted by \(\text{vec}(A)\), is the linear transformation which converts \(A\) to a column vector with the dimension \(mn \times 1\).

\[\text{vec}(A) = \text{vec}\left(\begin{array}{c}
a_1 \\
a_2 \\
\vdots \\
a_n
\end{array}\right) = \begin{pmatrix}
a_1 \\
a_2 \\
\vdots \\
a_n
\end{pmatrix} \quad (3.7)
\]

Proposition 3.3. If \(A = A_{p \times q}, \ X = X_{q \times m} \) and \(B = B_{m \times n}\) are matrices such that \(Y = AXB\) then

\[\text{vec}(Y) = \text{vec}(AXB) = (B^T \otimes A) \text{vec}(X) \quad (3.8)
\]

Proof 3.3. Let

\[B = (b_1 \ b_2 \ \cdots \ b_n)
\]

and

\[X = (x_1 \ x_2 \ \cdots \ x_m)
\]
then the $k$:th column of $AXB$ can be expressed as:

$$(AXB)_k = AXb_k = A \sum_{i=1}^{m} x_i b_{ik} =$$

$$= (b_{1k}A b_{2k}A \ldots b_{mk}A) \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} =$$

$$= ((b_{1k}, b_{2k}, \ldots, b_{mk}) \otimes A) \text{vec}(X) =$$

$$= (b_k^T \otimes A) \text{vec}(X).$$

This implies:

$$\text{vec}(AXB) = \begin{pmatrix} (AXB)_1 \\ (AXB)_2 \\ \vdots \\ (AXB)_k \\ \vdots \\ (AXB)_n \end{pmatrix} = \begin{pmatrix} b_1^T \otimes A \\ b_2^T \otimes A \\ \vdots \\ b_k^T \otimes A \\ \vdots \\ b_n^T \otimes A \end{pmatrix} \text{vec}(X) =$$

$$= (B^T \otimes A) \text{vec}(X).$$

\[ \Box \]

**Definition 3.8.** If $x, y \in \mathbb{R}^n$ are two real vectors then the inner product between $x$ and $y$ is:

$$(x, y) = x^T y \quad (3.9)$$

**Definition 3.9.** If $x, y \in \mathbb{R}^n$ are two real vectors then:

$$\cos \theta(x, y) = \frac{x^T y}{\|x\|_2 \|y\|_2} \quad (3.10)$$

is called cosine of the angle and is a measure describing how close the directions of the vectors $x$ and $y$ are.

If $\cos \theta(x, y)$ is close to one then the vectors $x$ and $y$ are close. The vectors are orthogonal if $\cos \theta(x, y) = 0$ i.e. if $(x, y) = x^T y = 0$.

**Definition 3.10.** If $X, Y \in \mathbb{R}^{m \times n}$ are two real matrices then the inner product between $X$ and $Y$ is:

$$\langle X, Y \rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} X_{ij} Y_{ij} = \text{tr} (X^T Y) = \text{tr} (XY^T) \quad (3.11)$$

where $\text{tr}$ is the trace of a matrix $^1$.

\[ ^1 \text{See Appendix D for more properties about the trace operator.} \]
Definition 3.11. If $v \in \mathbb{R}^n$ is a vector then the 1-norm, $\| \cdot \|_1$, of $v$ is:

$$\|v\|_1 = \sum_{i=1}^{n} v_i.$$

See Appendix D for more details about vector norms.

Definition 3.12. The spectral radius $\rho$ is a number generated from a square matrix $A$ such that:

$$\rho(A) = \max_{\lambda \in \sigma(A)} |\lambda|$$

where $\sigma(A)$ is a set containing the distinct eigenvalues of $A$.

The matrices that we perform computations on are non-negative i.e. every element in the matrix is greater than or equal to zero. For this type of matrices there exists a well developed theory by Perron and Frobenius which states:

Theorem 3.1 (Perron-Frobenius Theorem).

If $A$ is a non-negative square matrix then the spectral radius $\rho(A)$ is a non-negative eigenvalue of $A$ with a corresponding non-negative eigenvector $v$. We say that $\rho$ is the Perron root of $A$ and $v_\rho = v/\|v\|_1$ is the Perron vector. The notation $(\rho, v_\rho)$ is called an eigenpair for $A$.

The idea of the proof can be read in Appendix A and for a full proof see Meyer [29, p. 670]. With the definitions, propositions and the theorem above in mind we now can go further in order to analyze the mathematical expression of graph similarity.

Definition 3.13 (Blondel et al. [2]).

Let $G_A$ and $G_B$ be two graphs with $|V_A|$ and $|V_B|$ number of vertices respectively then the similarity $S$ between $G_A$ and $G_B$ is the solution of the eigenvalue problem:

$$\rho S = BSA^T + B^T SA$$

where $A$ and $B$ are the adjacency matrices for $G_A$ and $G_B$. The eigenvalue $\rho$ is the Perron root for the non-negative matrix

$$M = (A \otimes B + A^T \otimes B^T)$$

which is the matrix obtained by vectorization of (3.13), i.e.

$$\rho \text{vec}(S) = (A \otimes B + A^T \otimes B^T) \text{vec}(S) = M \text{vec}(S).$$

The equation (3.13) can be justified as a generalization of the hub and authority score by studying the product graph $G_A \times G_B$, defined as the graph with $|V_A| \times |V_B|$ vertices and where each vertex is a pair $(v_A, v_B)$ with one vertex from $G_A$ and one vertex from $G_B$. If $(v_A^{(1)}, v_B^{(1)})$ and $(v_A^{(2)}, v_B^{(2)})$ are two vertex pairs of the product graph then there is an edge connecting them if $v_A^{(1)} v_A^{(2)} \in E_A$ and $v_B^{(1)} v_B^{(2)} \in E_B$. The similarity score of a vertex $v$ in the product graph is then proportional to the sum of the similarity scores of the vertices adjacent to $v$. 
3.3.1 The Similarity Matrix Operator

In the equation (3.13) the matrices $A$ and $B$ are real, non-negative and square which implies that the matrix $S$ representing the similarity between the graphs is real and non-negative due to the Perron-Frobenius theory. To compute the similarity matrix $S$ in the general case, if the matrix is large, one can use one of the algorithms suggested in [34], but nevertheless it is interesting to analyze the equation to understand what properties it has. The matrix $M$ of (3.14) is symmetric i.e.

$$M = (A \otimes B + A^T \otimes B^T)$$

$$\Rightarrow M = (A^T \otimes B^T)^T + (A \otimes B)^T$$

$$\Rightarrow M = (A \otimes B + A^T \otimes B^T)^T$$

and the linear operator $M$ acting on $S$ can be expressed as:

$$M : \mathbb{R}^{n \times m} \ni S \mapsto \begin{pmatrix} B & B^T \end{pmatrix} \begin{pmatrix} S & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} A^T \\ A \end{pmatrix} \in \mathbb{R}^{n \times m} (3.15)$$

\[
\langle M(X), Y \rangle = \text{tr}\left( \begin{pmatrix} B & B^T \end{pmatrix} \begin{pmatrix} X & 0 \\ 0 & X \end{pmatrix} \begin{pmatrix} A^T \\ A \end{pmatrix} Y^T \right) = \\
= \text{tr} \left( BXA^TY^T + B^TXAY^T \right) = \\
= \text{tr} \left( A^TY^TBX + AY^TB^TX \right) = \\
\]

We can solve the problem (3.13) analytically by applying the spectral theorem on $M$ to construct an orthonormal basis $U$ of eigenvectors such that $M = U \Lambda U^T$ where $\Lambda$ is a diagonal matrix which contains the eigenvalues of $M$. The spectral radius $\rho(M)$ is an eigenvalue to $M$ since $M$ is non-negative and the corresponding eigenvector $u_\rho$ forms the solution to (3.13). We obtain the similarity matrix $S$ by reshaping the eigenvector $u_\rho$ to a matrix of the right dimensions, i.e. by using the inverse of the vec operator.
3.4. Self-similarity

Example 3.5. The similarity matrix of the two graphs below is:

\[
S = \begin{pmatrix}
0.2112 & 0.1788 & 0.1977 & 0.0125 \\
0.2451 & 0.4480 & 0.0479 & 0 \\
0.2422 & 0.0835 & 0.3802 & 0.1068 \\
0.3239 & 0.2459 & 0.3612 & 0.0641 \\
0.0860 & 0 & 0.2338 & 0.1935
\end{pmatrix}
\]

We can see that the similarity between the vertices \(a_2\) and \(b_2\) is 0.4480, and they are the vertices which are most similar to each other. The vertices \(a_4\) and \(b_2\) do not have any similarity at all, since \(a_4\) has an edge pointing to the vertex and \(b_2\) has edges pointing from the vertex. The same thing occurs between the vertices \(a_2\) and \(b_5\) which do not have any similarity either.

If either of the two adjacency matrices \(A\) and \(B\) are symmetric (\(A = A^T\) or \(B = B^T\)) then one can show that the similarity matrix is a rank one matrix (see Appendix B for theorem and proof). The consequence of this will be discussed later when we use the graph from the parallel texts.

3.4 Self-similarity

The theory of graph similarity also applies well on a single graph to derive the similarity between vertices within the graph. Let \(\mathcal{G}_A\) be a graph. By replacing the part constituted by a second graph \(\mathcal{G}_B\) with the graph itself, i.e \(\mathcal{G}_B = \mathcal{G}_A\) from (3.13) we obtain the following equations:

\[\rho S = ASA^T + A^TSA\]  
(3.16)

and

\[\rho \vec{S} = M \vec{S}\]

(3.17)

where

\[M = (A \otimes A + A^T \otimes A^T)\]

The symmetric linear operator \(\mathcal{M}\) can be expressed as:

\[\mathcal{M} : \mathbb{R}^{m \times m} \ni S \mapsto (A \ A^T) \begin{pmatrix} S & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} A^T \\ A \end{pmatrix} \in \mathbb{R}^{m \times m}\]

(3.18)

for \(A \in \mathbb{R}^{m \times m}\).

It follows naturally that a vertex \(v\) in a graph \(\mathcal{G}\) is most similar to itself, which implies that the self-similarity matrix has the largest, but not necessarily dominant, values on the diagonal.
Example 3.6. The left graph in example 3.5 has the self-similarity matrix:

\[
S = \begin{pmatrix}
0.3378 & 0.2731 & 0.2930 & 0.0144 \\
0.2731 & 0.4852 & 0.0428 & 0 \\
0.2930 & 0.0428 & 0.5192 & 0.1129 \\
0.0144 & 0 & 0.1129 & 0.1746
\end{pmatrix}
\]

The vertices \( a_2 \) and \( a_4 \) do not have any similarity due to that \( a_2 \) only has edges pointing from the vertex and \( a_4 \) only has edges pointing to the vertex.

To compute the self-similarity matrix analytically we can use the same procedure proposed above for computation of similarity between two distinct graphs. The self-similarity matrix is by structure symmetric, and possesses properties that makes it possible to generate a bilingual lexicon from the parallel text graph.

3.5 Self-similarity of Parallel Texts

The adjacency matrix of the directed parallel text graph has a special structure that we have to take into account when solving the equation (3.16) to find the self-similarity for this graph. We are interested in the self-similarity between the vertices corresponding to terms in the first language and the vertices corresponding to terms in the second language. In order to extract the part of the similarity matrix we need the following proposition and to analyze the obtained problem we shall use the singular value decomposition, SVD.

Proposition 3.4. Let \( S \) be the self-similarity matrix in the equation (3.16) and \( A \) the adjacency matrix from the directed parallel text graph then the block matrix structure of \( S \) is:

\[
S = \begin{pmatrix}
S_{11} & S_{12} & 0 \\
S_{21} & S_{22} & 0 \\
0 & 0 & S_{33}
\end{pmatrix}
\]  

(3.19)

where \( S_{ij} \), \( i = 1, 2 \), \( j = 1, 2 \) is the self-similarity between the terms in language \( i \) and the terms in language \( j \). The matrix \( S_{33} \) is the self-similarity of the documents in the parallel texts.

Proof 3.4. Assume that \( S \) is a \( 3 \times 3 \) block matrix such that

\[
S = \begin{pmatrix}
S_{11} & S_{12} & S_{13} \\
S_{21} & S_{22} & S_{23} \\
S_{31} & S_{32} & S_{33}
\end{pmatrix}
\]

Then by inserting this matrix in the equation (3.16) we obtain:

\[
\rho \begin{pmatrix}
S_{11} & S_{12} & S_{13} \\
S_{21} & S_{22} & S_{23} \\
S_{31} & S_{32} & S_{33}
\end{pmatrix}
\]
\[3.5. \text{Self-similarity of Parallel Texts}\]

\[
\begin{pmatrix}
0 & 0 & A_1 \\
0 & 0 & A_2 \\
0 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
S_{11} & S_{12} & S_{13} \\
S_{21} & S_{22} & S_{23} \\
S_{31} & S_{32} & S_{33}
\end{pmatrix}
\begin{pmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
A_1^T & A_2^T & 0
\end{pmatrix}
+ \begin{pmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
A_1^T & A_2^T & 0
\end{pmatrix}
\begin{pmatrix}
S_{11} & S_{12} & S_{13} \\
S_{21} & S_{22} & S_{23} \\
S_{31} & S_{32} & S_{33}
\end{pmatrix}
\begin{pmatrix}
0 & 0 & A_1 \\
0 & 0 & A_2 \\
0 & 0 & 0
\end{pmatrix}
= \begin{pmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
A_1^T & A_2^T & 0
\end{pmatrix}
\begin{pmatrix}
S_{11} & S_{12} & S_{13} \\
S_{21} & S_{22} & S_{23} \\
S_{31} & S_{32} & S_{33}
\end{pmatrix}
\begin{pmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
A_1^T & A_2^T & 0
\end{pmatrix}
+ \begin{pmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & A_1^T S_{11} A_1 + A_1^T S_{12} A_2 + A_2^T S_{21} A_1 + A_2^T S_{22} A_2
\end{pmatrix}
= \begin{pmatrix}
A_1 S_{31} A_1^T & A_1 S_{33} A_2^T & 0 \\
A_2 S_{31} A_1^T & A_2 S_{33} A_2^T & 0 \\
0 & 0 & S_{33}
\end{pmatrix}
\]

where

\[\tilde{S}_{33} = A_1^T S_{11} A_1 + A_1^T S_{12} A_2 + A_2^T S_{21} A_1 + A_2^T S_{22} A_2\]

For \(\rho \neq 0\) this equality implies that

\[S_{13} = S_{23} = S_{31} = S_{32} = 0.\]

i.e. the similarity matrix \(S\) has the block matrix structure stated in (3.19). \(\square\)

In order to solve and analyze the equation (3.16) with the block structure similarity matrix (3.19) we will use the singular value decomposition.

**Theorem 3.2 (Singular Value Decomposition).**

Any real \(2 \times m\) matrix \(A_{m \times n}\) can be factorized

\[A_{m \times n} = U_{m \times m} \begin{pmatrix}
\Sigma_{k \times k} & 0 \\
0 & 0
\end{pmatrix}_{m \times n} V_{n \times n}^T \tag{3.20}\]

where \(U_{m \times m}\) and \(V_{n \times n}\) are orthogonal matrices and

\[\Sigma_{k \times k} = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_k), \ k = \text{rank}(A_{m \times n})\]

and \(\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k \geq 0\).

The values \(\sigma_i, \ i = 1, \ldots k\) are called the singular values of \(A_{m \times n}\) and the columns of \(U_{m \times m}\) and \(V_{n \times n}\) are called the left respectively the right singular vectors of \(A_{m \times n}\).

**Proof.** The proof can be read in Golub, V. Loan [19, p. 70]. \(\square\)

\(^2\)The complex case is derived in similar fashion by unitary singular matrices \(U_{m \times m}\) and \(V_{n \times n}\) and by replacing the transpose \(T\) with the conjugate transpose \(H\).
Let

\[ U = (U_1, U_2) \text{ and } V = (V_1, V_2) \]

where \( U_1 \) and \( V_1 \) corresponds to the first \( k \) left singular vectors and the first \( k \) right singular vectors respectively, then we can express the thin SVD as:

\[ A = U \begin{pmatrix} \Sigma & 0 \\ 0 & 0 \end{pmatrix} V^T = (U_1, U_2) \begin{pmatrix} \Sigma & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} V_1^T \\ V_2^T \end{pmatrix} = U_1 \Sigma V_1^T. \]

Henceforth, when we refer to SVD we mean the thin SVD. In our case, we for simplicity, assume for \( A \in \mathbb{R}^{m \times n} \) that \( n > m \) and \( \text{rank}(A) = m \). The SVD of this matrix is then:

\[ A = U (\Sigma \ 0) \begin{pmatrix} V_1^T \\ V_2^T \end{pmatrix} = U \Sigma V_1^T. \quad (3.21) \]

This will not be a restriction in the rest of the computations, since if we instead have \( m > n \) and \( \text{rank}(A) = n \) then we can just apply the SVD on the transpose of \( A \) such that:

\[ A^T = \hat{U} (\Sigma \ 0) \begin{pmatrix} V_1^T \\ V_2^T \end{pmatrix} = \hat{U} \Sigma \hat{V}_1^T. \]

Observe that \( U \neq \hat{U}, V_1 \neq \hat{V}_1 \) and \( V_2 \neq \hat{V}_2 \).

We claim that the bilingual lexicon can be extracted from the similarity matrix \( S_{12} \), and this is the hypothesis that we will apply in order to perform further computations. From the proof of the proposition (3.4) we can derive the expression:

\[ \rho \begin{pmatrix} S_{11} & S_{12} & 0 \\ S_{21} & S_{22} & 0 \\ 0 & 0 & S_{33} \end{pmatrix} = \begin{pmatrix} A_1 S_{33} A_1^T & A_1 S_{33} A_2^T & 0 \\ A_2 S_{33} A_1^T & A_2 S_{33} A_2^T & 0 \\ 0 & 0 & \tilde{S}_{33} \end{pmatrix} \quad (3.22) \]

where

\[ \tilde{S}_{33} = A_1^T S_{11} A_1 + A_1^T S_{12} A_2 + A_2^T S_{21} A_1 + A_2^T S_{22} A_2 \]

which implies:

\[ \rho S_{11} = A_1 S_{33} A_1^T \]
\[ \rho S_{12} = A_1 S_{33} A_2^T \]
\[ \rho S_{21} = A_2 S_{33} A_1^T \]
\[ \rho S_{22} = A_2 S_{33} A_2^T \]
\[ \rho S_{33} = \tilde{S}_{33}. \quad (3.23) \]

If we insert (3.23) into \( \tilde{S}_{33} \) we obtain:

\[ \rho^2 \tilde{S}_{33} = A_1^T A_1 S_{33} A_1^T A_1 + A_1^T A_1 S_{33} A_2^T A_2 + A_2^T A_2 S_{33} A_1^T A_1 + A_2^T A_2 S_{33} A_2^T A_2 = \]

\[ = (A_1^T A_1) S_{33} (A_1^T A_2^T) (A_1^T A_2). \quad (3.24) \]

To evaluate this expression we use the SVD, with the assumption made in (3.21), on the parallel text matrix:

\[ A_p = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \in \mathbb{R}^{m \times n}, \]
by partitioning the left singular matrix $U$ in two parts thus:

$$A_p = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} = \begin{pmatrix} U^{(1)} \\ U^{(2)} \end{pmatrix} \Sigma V_1^T$$  (3.25)

where $U = \begin{pmatrix} U^{(1)} \\ U^{(2)} \end{pmatrix}$ is the considered partition. By inserting this expression into equation (3.24) we get:

$$\rho^2 S_{33} = V_1 \Sigma^2 V_1^T S_{33} V_1 \Sigma^2 V_1^T$$  (3.26)

Letting $\hat{S}_{33} = V_1^T S_{33} V_1$ we then get:

$$\rho^2 \hat{S}_{33} = \Sigma^2 \hat{S}_{33} \Sigma^2.$$  (3.27)

The matrix $\Sigma$ is a diagonal matrix which implies that the matrix $\Sigma^2$ also is a diagonal matrix. If we assume that the singular values of $\Sigma$ are:

$$\sigma_1 > \sigma_2 \geq \sigma_3 \geq \cdots \geq \sigma_k > 0, \ m = \text{rank}(A_p),$$

then the matrix $\Sigma^2$ is:

$$\Sigma^2 = \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_k \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_k^2 \end{pmatrix}.$$  

By applying the vec operator on the equation (3.27) we get:

$$\rho^2 \text{vec} \left( \hat{S}_{33} \right) = (\Sigma^2 \otimes \Sigma^2) \text{vec} \left( \hat{S}_{33} \right) =$$

$$= \begin{pmatrix} \sigma_1^2 \Sigma^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 \Sigma^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_k^2 \Sigma^2 \end{pmatrix} \text{vec} \left( \hat{S}_{33} \right)$$

The value $\rho^2$ that we are interested in is by definition the maximum eigenvalue of the matrix $(\Sigma^2 \otimes \Sigma^2)$ and by the assumption above we can see that this eigenvalue can be found in the first non-zero submatrix of $(\Sigma^2 \otimes \Sigma^2)$:

$$\sigma_1^4 \Sigma^2 = \begin{pmatrix} \sigma_1^4 & 0 & \cdots & 0 \\ 0 & \sigma_1^4 \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_1^4 \sigma_k^2 \end{pmatrix}$$

since $\sigma_1^4$ is the maximum element of $\Sigma^2$. The value $\rho^2$ is then the maximum value of $\sigma_1^4 \Sigma^2$ which is:

$$\rho^2 = \max \left( \sigma_1^4 \Sigma^2 \right) = \sigma_1^4,$$

which implies that:

$$\rho = \pm \sigma_1^2.$$
The corresponding eigenvector of the eigenvalue $\sigma^2_1$ is the unit vector:

$$\text{vec} \left( \hat{S}_{33} \right) = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \in \mathbb{R}^k.$$

This unit vector is obtained by applying the vec operator on the matrix:

$$\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \in \mathbb{R}^{k \times k}$$

which implies that:

$$\hat{S}_{33} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix} = e_{\cdot 1} \cdot e_{\cdot 1}^T,$$

where $e_{\cdot 1} \in \mathbb{R}^k$.

The similarity matrix $S_{33}$ is implicitly:

$$V_1^T S_{33} V_1 = e_{\cdot 1} \cdot e_{\cdot 1}^T = \hat{S}_{33}.$$
where $v^{(1)}_1$ is the first column vector of $V_1$, $\rho = \pm \sigma^2_1$ and $S_{33} = (\pm v^{(1)}_1)(\pm v^{(1)}_1)^T$, i.e. the solution corresponds to the two eigenpairs: 

$$
(\sigma^2_1, \pm v^{(1)}_1)
$$

and 

$$
(-\sigma^2_1, \pm v^{(1)}_1).
$$

The entire similarity matrix $S$ is then:

$$
S = \begin{pmatrix}
 u^{(1)}_1 \cdot (u^{(1)}_1)^T & u^{(1)}_1 \cdot (u^{(2)}_1)^T & 0 \\
 u^{(2)}_1 \cdot (u^{(1)}_1)^T & u^{(2)}_1 \cdot (u^{(2)}_1)^T & 0 \\
 0 & 0 & (\pm v^{(1)}_1) \cdot (\pm v^{(1)}_1)^T
\end{pmatrix}
(3.28)
$$

We can express this matrix with a sum of two rank one outer products, thus:

$$
S = \begin{pmatrix}
 u^{(1)}_1 \cdot (u^{(1)}_1)^T & u^{(1)}_1 \cdot (u^{(2)}_1)^T & 0 \\
 u^{(2)}_1 \cdot (u^{(1)}_1)^T & u^{(2)}_1 \cdot (u^{(2)}_1)^T & 0 \\
 0 & 0 & (\pm v^{(1)}_1) \cdot (\pm v^{(1)}_1)^T
\end{pmatrix} = \\
\begin{pmatrix}
 u^{(1)}_1 \\
 u^{(2)}_1 \\
 \vec{0}
\end{pmatrix} \begin{pmatrix}
 (u^{(1)}_1)^T \\
 (u^{(2)}_1)^T \\
 \vec{0}^T
\end{pmatrix} + \begin{pmatrix}
 \vec{0} \\
 \vec{0} \\
 \pm v^{(1)}_1
\end{pmatrix} \begin{pmatrix}
 \vec{0}^T \\
 \vec{0}^T \\
 (\pm v^{(1)}_1)^T
\end{pmatrix}
$$

where $\vec{0}$ is a zero vector. From this expression we can conclude that the self-similarity matrix $S$ is a rank two matrix. If the parallel text graph is undirected the solution (3.28) is one possible solution. In this case there are infinitely many solutions.$^3$

To summarize the computations above we can say that by applying the SVD on the parallel texts we can find the self-similarity matrix $S$ by solving the eigenvalue problem (3.22). It turns out that the largest eigenvalue $\rho$, the Perron root, is $\rho = \pm \sigma^2_1$ and therefore there are two possible solutions which are related to the eigenpairs $(+\sigma^2_1, v^{(1)}_1)$ and $(-\sigma^2_1, -v^{(1)}_1)$. To extract a bilingual lexicon we are interested in the similarity matrix $S_{12}$, which is obtained from the the outer product $u^{(1)}_1 \cdot (u^{(2)}_1)^T$. This rank one solution is by itself not a good solution when measuring the precision of the translations in the extracted bilingual lexicon but it holds the answer of how to proceed.

### 3.6 Subgraphs and Rank $k$ Approximations

The rank one solution $S_{12} = u^{(1)}_1 \cdot (u^{(2)}_1)^T$, originates from the rank one approximation derived from the parallel text matrix:

$$
\begin{pmatrix}
 A_1 \\
 A_2
\end{pmatrix} \approx \sigma_1 \begin{pmatrix}
 u^{(1)}_1 \\
 u^{(2)}_1
\end{pmatrix} (v^{(1)}_1)^T.
(3.29)
$$

This matrix corresponds to a subgraph of the parallel text graph where all vertices corresponding to documents are contracted to one dominating document vertex $D$.

$^3$See Appendix B for the proposition and proof.
Example 3.7. The subgraph of the directed parallel text graph obtained from example (3.4) above can with the rank one solution be illustrated symbolically as:

![Graph Illustration](image)

If we use this solution to extract a bilingual lexicon the precision of the results will not be good. In order to obtain better results we have to increase the rank of the approximation. We can increase the rank by adding the second singular value $\sigma_2$ and the corresponding singular vectors $u_{21}$, $u_{22}$ and $v_{12}$ to the rank one approximation (3.29), then we get a rank two approximation which is:

$$
\begin{pmatrix}
A_1 \\
A_2
\end{pmatrix} \approx \begin{pmatrix} u_{11}^{(1)} & u_{12}^{(1)} \\
 u_{21}^{(1)} & u_{22}^{(1)}
\end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\
 0 & \sigma_2
\end{pmatrix} \begin{pmatrix} v_{11}^{(1)^T} \\
v_{21}^{(1)^T}
\end{pmatrix} \tag{3.30}
$$

In the graph we obtain the rank two approximation by first removing the rank one approximation from the parallel text graph $G(A_p) = G(A_1^p)$ and then generate a rank one approximation from this new graph, $G(A_2^p)$. The rank two approximation is the sum of the rank one approximations from the graphs $G(A_1^p)$ and $G(A_2^p)$ respectively.

Example 3.8. If the parallel text graph $G(A_p)$ is approximated by a rank two solution then it can be illustrated symbolically as:
where the two graphs in the sum are the rank one approximations obtained from $\mathcal{G}(A^1_p)$ and $\mathcal{G}(A^2_p)$ respectively.

According to (3.21) rank $A_p = m$ and if we let $k < m$. Then we can generate a rank $k$ approximation of $A_p$ by the summation of the outer products, 

$$\sigma_i \begin{pmatrix} u_i^{(1)} \\ u_i^{(2)} \end{pmatrix} \begin{pmatrix} v_i^{(1)} \end{pmatrix}^T, \text{ for } i = 1, 2, \ldots, k,$$

i.e. the sum of the rank one approximations obtained from $\mathcal{G}(A_i^p)$. The rank $k$ approximation of $A_p$ is

$$A_p = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \approx \sum_{i=1}^{k} \sigma_i \begin{pmatrix} u_i^{(1)} \\ u_i^{(2)} \end{pmatrix} \begin{pmatrix} v_i^{(1)} \end{pmatrix}^T \quad (3.31)$$

and expressed with matrices

$$A_p = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \approx \begin{pmatrix} U_k^{(1)} & U_k^{(2)} \end{pmatrix} \Sigma_k (V_k^{(1)})^T \quad (3.32)$$

where $U_k^{(1)}$ and $U_k^{(2)}$ are the partitioned matrices consisting of the first $k$ left singular vectors and $\Sigma_k$ is a diagonal matrix of the $k$ largest singular values and $V_k^{(1)}$ the $k$ right singular vectors.

The matrix $\begin{pmatrix} U_k^{(1)} \\ U_k^{(2)} \end{pmatrix}$ is an approximate orthogonal basis for the document space and the matrix product $\Sigma_k (V_k^{(1)})^T$ holds the coordinates for the documents. The columns of this orthogonal basis and the columns of the coordinate
matrix contains negative elements i.e. it is possible that a document, a column vector with non-negative elements, in $A_p$ is approximated by a column vector containing negative elements. If $a_j$ is a document in the parallel text matrix $A_p$ and if we set $H_k = \Sigma_k(V^{(1)}_k)^T$ then this document is approximated by

$$a_j = \begin{pmatrix} a_{1j} \\ a_{2j} \end{pmatrix} \approx \begin{pmatrix} U^{(1)}_k \\ U^{(2)}_k \end{pmatrix} h_j = \begin{pmatrix} c_{1j} \\ c_{2j} \end{pmatrix} = c_j$$  \hspace{1cm} (3.33)$$

where $h_j$ is a column vector of $H_k$ and $c_j$ the approximation of the document where some of the elements in $c_j$ are possibly negative. It is hard to interpret what the negative element actually mean in terms of the term-by-document matrix. This will also affect the similarity in an indistinct way.

The new similarity matrix $S_{33}$ with the rank $k$ approximation is:

$$S_{33} = V^{(1)}_k (V^{(1)}_k)^T$$

which yields:

$$\rho S_{12} = A_1 S_{33} A_2^T = U^{(1)} \Sigma V^{(1)}_k (V^{(1)}_k)^T V_1 \Sigma (U^{(2)})^T =$$

$$= U^{(1)} \Sigma \begin{pmatrix} I_{k \times k} & 0 \\ 0 & 0 \end{pmatrix} \Sigma U^{(1)}_2 = U^{(1)} \Sigma ^2 (U^{(2)}_k)^T.$$  \hspace{1cm} (3.34)

The equation (3.34) can be represented by the sum

$$\rho S_{12} = \sum_{i=1}^{k} \sigma_i^2 u_i^{(1)}(u_i^{(2)})^T$$  \hspace{1cm} (3.35)

where some of the elements in $S_{12}$ are possible negative. The rank $k$ approximations of the similarity matrices $S_{11}$, $S_{21}$ and $S_{22}$ can be computed in similar manner, as for $S_{12}$ in (3.34). The similarity matrix $S$ is then:

$$\rho S = \rho \begin{pmatrix} S_{11} & S_{12} & 0 \\ S_{21} & S_{22} & 0 \\ 0 & 0 & S_{33} \end{pmatrix} =$$

$$= \begin{pmatrix} U^{(1)} \Sigma ^2 (U^{(1)}_k)^T & U^{(1)} \Sigma ^2 (U^{(2)}_k)^T & 0 \\ U^{(2)} \Sigma ^2 (U^{(1)}_k)^T & U^{(2)} \Sigma ^2 (U^{(2)}_k)^T & 0 \\ 0 & 0 & V^{(1)}(V^{(1)}_k)^T \end{pmatrix}.$$  \hspace{1cm} (3.36)

We can observe that $S_{21} = S_{12}^T$ and that the submatrix

$$\rho \begin{pmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{pmatrix} = \begin{pmatrix} U^{(1)} \Sigma ^2 (U^{(1)}_k)^T & U^{(1)} \Sigma ^2 (U^{(2)}_k)^T \\ U^{(2)} \Sigma ^2 (U^{(1)}_k)^T & U^{(2)} \Sigma ^2 (U^{(2)}_k)^T \end{pmatrix}$$

can be factorized with the matrix product:

$$\begin{pmatrix} U^{(1)} \Sigma ^2 (U^{(1)}_k)^T & U^{(1)} \Sigma ^2 (U^{(2)}_k)^T \\ U^{(2)} \Sigma ^2 (U^{(1)}_k)^T & U^{(2)} \Sigma ^2 (U^{(2)}_k)^T \end{pmatrix} = \begin{pmatrix} U^{(1)}_k \\ U^{(2)}_k \end{pmatrix} \Sigma ^2 \begin{pmatrix} (U^{(1)}_k)^T & (U^{(2)}_k)^T \end{pmatrix}.$$
If we then apply the full SVD, by letting \( k = m = \text{rank}(A_p) \), in the above expression we get:

\[
\rho \left( \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \right) = \left( U^{(1)} \Sigma^2 U^{(1)T} \right) \left( U^{(2)} \Sigma^2 U^{(2)T} \right) = \\
\left( \begin{bmatrix} U^{(1)} \\ U^{(2)} \end{bmatrix} \right) \Sigma^2 \left( \begin{bmatrix} (U^{(1)})^T \\ (U^{(2)})^T \end{bmatrix} \right) = \\
\left( \begin{bmatrix} U^{(1)} \\ U^{(2)} \end{bmatrix} \right) \Sigma^2 \left( \begin{bmatrix} \Sigma^T \\ 0 \end{bmatrix} \right) \left( \begin{bmatrix} (U^{(1)})^T \\ (U^{(2)})^T \end{bmatrix} \right) = \\
= \left( A_1 \right) \left( A_2 \right)^T = \\
= \left( \begin{bmatrix} A_1^T \\ A_2 \end{bmatrix} \right) \left( \begin{bmatrix} A_2^T \\ A_1 \end{bmatrix} \right) .
\]

The equality:

\[
\rho \left( \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \right) = \left( A_1 A_2^T \right) = \\
= \left( \begin{bmatrix} A_1^T \\ A_2 \end{bmatrix} \right) \left( \begin{bmatrix} A_2^T \\ A_1 \end{bmatrix} \right)
\]

is an application of the vector space model, because if we normalizes the rows in the term-by-document matrices \( A_1 \) and \( A_2 \) then the matrix product \( A_i A_j^T \), \( i = 1, 2 \) \( j = 1, 2 \) is the cosine of angle (3.10) between the terms in the term-by-document matrix \( A_i \) and the terms in the term-by-document matrix \( A_j \) respectively. If we consider the matrix product \( A_1 A_2^T \) and let \( a_{ij}^{(1)(2)} \) be an arbitrary element of this matrix, then we can express this element with the cosine of the angle:

\[
a_{ij}^{(1)(2)} = \cos \theta((A_1)_i, ((A_2)_j)^T)
\]

where \((A_1)_i\) is the \( i \)th row of \( A_1 \) and \((A_2)_j\) the \( j \)th row of \( A_2 \). We can interpret the element \( a_{ij}^{(1)(2)} \) as the probability that a term \( t_j^{(2)} \) in the term-by-document \( A_2 \) is an accurate translation of the term \( t_i^{(1)} \) in the term-by-document \( A_1 \). If \( a_{ij}^{(1)(2)} \) is close to one then it is likely that the term \( t_j^{(2)} \) is an accurate translation of the term \( t_i^{(1)} \), but if \( a_{ij}^{(1)(2)} \) is nearly zero then it is less likely that the term \( t_j^{(2)} \) is an accurate translation of the term \( t_i^{(1)} \). Another interpretation of the element \( a_{ij}^{(1)(2)} \) is given by considering the term space spanned by the parallel text matrix:

\[
A_p = \left( A_1 \\ A_2 \right)
\]

then \( a_{ij}^{(1)(2)} \) describes the relation between the directions of the term vectors \( t_i^{(1)} \) and \( t_j^{(2)} \) in the term space. If the vectors \( t_i^{(1)} \) and \( t_j^{(2)} \) are close then \( a_{ij}^{(1)(2)} \) is close to one and similarly if the vectors \( t_i^{(1)} \) and \( t_j^{(2)} \) are nearly orthogonal then \( a_{ij}^{(1)(2)} \) is close to zero.
We have shown that the similarity matrix $S_{12}$ can be expressed with the rank $k$ approximation obtained from the SVD of $A_p$:

$$S_{12} = U_k^{(1)} \Sigma_k^2 (U_k^{(2)})^T$$

and we also showed, if we use the full SVD, that $S_{12}$ can be expressed with the matrix product:

$$S_{12} = A_1 A_2^T$$

and that the matrix product is an application of the vector space model. We will refer to these two expressions with the notations SS-SVD and VSM respectively.

The first notation represents the self-similarity matrix computations of $S_{12}$ by SVD and the second represents the computations by the vector space model. We can use the VSM to find a third way in which $S_{12}$ can be expressed, by making a rank $k$ approximation of the matrix product with the SVD such that:

$$\hat{S}_{12} = A_1 A_2^T \approx \hat{U}_k^{(1)} \hat{\Sigma}_k (\hat{V}_k^{(1)})^T.$$  

(3.40)

This expression is not equal to the one in (3.38), since the SVD used in the SS-SVD method originates from the parallel text matrix $A_p$ and the SVD used in (3.40) is directly applied on the matrix product (3.39). We will refer to this third method with the notation VSM-SVD, i.e. the SVD of VSM.

From the approximation (3.33) we concluded that a non-negative document vector $a_{.j} \in A_p$ is approximated by a vector $c_{.j}$, which can contain possible negative elements. This is also the case when we apply the VSM-SVD method, i.e. the non-negative vector:

$$a_{.j}^{(1)(2)} = A_1 ((A_2)_{.j})^T$$

is approximated by:

$$a_{.j}^{(1)(2)} \approx \hat{U}_k^{(1)} \hat{\Sigma}_k (\hat{V}_k^{(1)})^T, j = \hat{U}_k^{(1)} \hat{H}_k, j = \hat{c}_j$$  

(3.41)

where $\hat{\Sigma}_k (\hat{V}_k^{(1)})^T = \hat{H}_k$ and $\hat{c}_j$ is a vector which contains possible negative elements.

The interpretation of the approximations (3.33) and (3.41) is unclear with respect to the non-negativity of $A_p$ and $A_1 A_2^T$. This will also affect the elements in the similarity matrices $S_{12}$ and $\hat{S}_{12}$ represented by these two approaches, i.e. these elements are not necessarily non-negative. Despite this fact it is possible to extract a bilingual lexicon from $S_{12}$ and $\hat{S}_{12}$ due to a hypothesis describing a property of the positive elements in these similarity matrices.

**Hypothesis 3.1.** We have discovered by several independent computations that if $s_{.q}$ is an arbitrary column $q$ of $S_{12}$ then if there exists a maximum positive element $s_{iq} \in s_{.q}$ such that $s_{iq} \gg 0$ and if $s_{jq} \in s_{.q}$ is the maximum negative element such that $s_{jq} \leq s_{iq} \leq 0, \forall l : s_{il} \leq 0$, then it follows that $|s_{iq}| \geq |s_{jq}|$.

This hypothesis also holds for the similarity matrix $\hat{S}_{12}$ computed by VSM-SVD.
3.7 The Automatic Bilingual Lexicon Acquisition from $S_{12}$

Example 3.9. If we have the matrices $A_1$ and $A_2$ derived from the adjacency matrix in example (3.3), the similarity matrices $S_{12}$ and $\hat{S}_{12}$ can be computed by a rank two approximation using the methods SS-SVD and VSM-SVD respectively. With SS-SVD we get the similarity matrix:

$$S_{12} = \begin{pmatrix} 3.0012 & 1.9987 & -0.0054 & 2.0049 \\ 1.0969 & -0.0998 & 0.5684 & 2.2924 \\ 1.0412 & 0.9576 & -0.1834 & 0.1671 \\ 1.0969 & -0.0998 & 0.5684 & 2.3934 \\ 1.0412 & 0.9576 & -0.1834 & 0.1671 \end{pmatrix}$$

and with VSM-SVD we get the similarity matrix:

$$\hat{S}_{12} = \begin{pmatrix} 3.0080 & 2.0576 & 0.2143 & 1.9009 \\ 0.9968 & -0.0234 & 0.9130 & 2.0403 \\ 0.9911 & 0.9361 & -0.2377 & 0.1100 \\ 0.9968 & -0.0234 & 0.9130 & 2.0403 \\ 0.9911 & 0.9361 & -0.2377 & 0.1100 \end{pmatrix}$$

We see that there are negative elements in two of the column vectors but as the hypothesis states the absolute value of these elements is less than the corresponding maximum positive value in each of the two column vectors.

The hypothesis and the example (3.9) holds the answer of how to extract the automatic bilingual lexicon from the self-similarity matrices $S_{12}$ and $\hat{S}_{12}$, and also emphasizes the implementation of a SVD routine to compute these matrices.

## 3.7 The Automatic Bilingual Lexicon Acquisition from $S_{12}$

Let as for a moment assume that the matrix $S_{12}$ symbolizes the similarity matrices we can obtain from SS-SVD, VMS and VMS-SVD, then $S_{12}$ is a $t^{(1)} \times t^{(2)}$ matrix where $t^{(1)}$ is the number of terms in the first language and $t^{(2)}$ is the number of terms in the second language. We will consider the text collection in the first language as the source text and the text collection in the second language as the target text. This is the same notation which is used in (3.1).

The approach to find a (possible) translation between a term $t^{(1)}_i$ and a term $t^{(2)}_j$ is described in the following algorithm.

### Bilingual lexicon from $S_{12}$

1. For $j = 1, 2, \ldots, t^{(2)}$
2. Take the column vector $s_j$ from $S_{12}$.
3. Find the maximum element and the position $i$ of this element, i.e. $m_{ij} = \max(s_{ij})$, $i = 1, 2, \ldots, t^{(1)}$.
4. Match the term $t^{(1)}_i$ from the source text dictionary, with the term $t^{(2)}_j$ from the target text dictionary.
5. end for

This algorithm implies that we do not need to compute \( S_{12} \) explicitly, since the maximum element obtained from one column is independent from the maximum elements obtained from the other columns. Thus we can compute each column of \( S_{12} \) and then put the matched term indices \( i \) and \( j \) in a list.

We obtain the columns \( s_{j} \) of \( S_{12} \), with SS-SVD and VSM-SVD respectively, by the matrix-vector multiplications:

\[
s_{j} = \left( U^{(1)} k \Sigma^{2} k \right) \left( u^{(2)} j \right)^T
\]

and

\[
\hat{s}_{j} = \left( \hat{U}^{(1)} k \hat{\Sigma} k \right) \left( \hat{v}^{(1)} j \right)^T
\]

where \( u^{(2)} j \) is the \( j \):th column vector of \( U^{(2)} k \) and \( \hat{v}^{(1)} j \) is the \( j \):th column vector of \( \hat{V}^{(1)} k \).

In order to receive accurate results when using SS-SVD we normalize the rows of \( U^{(1)} k \Sigma^{2} k \) and of \( U^{(2)} k \), then each term \( s_{ij} \in S_{12} \) can be interpreted with the distance measure (3.10) \( s_{ij} = \cos \theta((U^{(1)} k \Sigma^{2} k) i, (u^{(2)} j)^T) \). If \( s_{ij} \) is close to one then the terms \( t^{(1)} i \) and \( t^{(2)} j \) are likely to be exact translations. In similar manner when using VSM-SVD, we normalize the rows of \( \hat{U}^{(1)} k \hat{\Sigma} k \) and \( \hat{V}^{(1)} k \), then the interpretation of each term \( \hat{s}_{ij} \in \hat{S}_{12} \) is the same as described for SS-SVD.

### 3.7.1 Verifying the Translations, Semantics of Swedish and English Words

The translations in the smaller sample must be verified with some existing bilingual lexicon 4. Before we can start verifying the translations we must set up some rules for describing the accuracy of a translation. In order to do that we have to consider the structural differences of the Swedish and English language. One factor that is affecting the translations is the compound words. In Swedish there are many more compound words than in English, for example the Swedish compound word “bruttonationalprodukt” has the English translation “gross national product” i.e. one word in Swedish corresponds to three words in English, in this case. The semantics of the two languages is also an important factor. One Swedish word can have several English translations and vice versa and in some cases there does not exist an accurate translation of a word. For instance the Swedish word “lagom” 5 does not have an exact English correspondence, but the word can of course be described in English depending on the context of the Swedish sentence where the word appears. The part of speech between the words in the translations can also differ, even when we know that the words in the Swedish and in the English text collections are stemmed to their base form it does not imply that the part of speech between the words in a translation is the same.

When verifying the translations we will use three different scores:

4http://www.websters-online-dictionary.org/ and http://lexin.nada.kth.se/sve-eng.html
5“Not to little and not to much” or “just right”.
3.7. The Automatic Bilingual Lexicon Acquisition from $S_{12}$

- 1, if the translation is correct.
- 0.5, if the translation is almost correct.
- 0, if the translation is incorrect.

The first score is used when a translation is considered to be correct, i.e. if the translation can be found in an existing bilingual lexicon. The second score, 0.5, is a consequence of the discussion above. This score is used when for example an English translation of a Swedish compound word only contains one part of the compound word or if the tense of an English word is present but the tense of the translated Swedish word is imperfect. The last score is applied when a translation is incorrect i.e. when there is no semantic connection between the words in the two languages. The precision $\hat{S}_{pre}$ is then computed by the summation of all the scores divided by the size of the observed sample i.e.

$$\hat{S}_{pre} = \frac{1}{n} \sum_{i=1}^{n} s_i$$

(3.44)

where $s_i$ is the given score of the $i$:th translation and $n$ is the sample size. The precision for the whole sample, the bilingual lexicon, is denoted by:

$$S_{pre} = \frac{1}{N} \sum_{i=1}^{N} s_i$$

(3.45)

where $N$ is the size of the whole lexicon.

3.7.2 An Interface with MATLAB, the Bilingual Lexicon and a Statistical Model

With the dictionaries generated from GTP (see Appendix E.2 for more details) and the list obtained from the algorithm above we can now construct the bilingual lexicon in MATLAB. By sorting the maximum values, in the list and the corresponding indices, in descending order, we get a bilingual lexicon with the most precise translations at the top of the lexicon and the more inexact translation at the bottom of the lexicon. The precision of the translations is decreasing, the more further down we check in the bilingual lexicon.

The next step is to verify the precision $S_{pre}$ of the translations in the bilingual lexicon. We do not want to check each and every translation in the lexicon, because it corresponds to about 6000 translations. Instead we use a statistical model to generate a smaller sample from the bilingual lexicon. The precision $\hat{S}_{pre}$ of the translations from the observed sample is verified and to derive the precision for all translations in the bilingual lexicon we create a confidence interval $I_{pre}$ from the verified sample. We then know, with a given probability, that the precision $S_{pre}$ will be in the confidence interval, i.e. $S_{pre} \in I_{pre}$ with e.g. confidence level 0.95.

In order to create a confidence interval we need to describe the observed sample $\hat{S}_{pre}$. The theory presented below is referred to [6].
Let $X_i$ be one observation from the whole sample $p$ then the probability mass function (PMF) is:

$$P_{X_i}(s_k) = \begin{cases} p_1 & s_1 = 1 \\ p_2 & s_2 = 0.5 \\ p_3 & s_3 = 0 \\ 0 & \text{otherwise,} \end{cases}$$

where $\sum_{k=1}^3 p_k = 1$ and $p_k, k = 1, 2, 3$ is the probability that the translation from the observation $X_i$ has the score $s_k, k = 1, 2, 3$. If we do $n$ independent observations from the sample $p$ and let $X_k, k = 1, 2, 3$ denote the number of times we obtain the score $s_k, k = 1, 2, 3$ then the discrete random variable $(X_1, X_2, X_3)$ is multinomial with the multivariate joint PMF:

$$P_{X_1, X_2, X_3}(s_1, s_2, s_3) = \binom{n}{s_1, s_2, s_3} \prod_{k=1}^3 p_k^{s_k}.$$ 

If we let $X_{\text{pre}} = X_1 + X_2 + \cdots + X_i + \cdots + X_n$ be the sum of $n$ independent observations $X_i$ from the whole sample. Then the distribution of the precision variable $X_{\text{pre}}$ can be approximated by the normal distribution, such that $X_{\text{pre}}$ is approximate $N(S_{\text{pre}}, D)$, where $D = \sqrt{S_{\text{pre}}(1 - S_{\text{pre}})/n}$ \(^6\). We do not know $D$ but can approximate it with the estimator $d = \sqrt{\hat{S}_{\text{pre}}(1 - \hat{S}_{\text{pre}})/n}$. The confidence interval for $\hat{S}_{\text{pre}}$ with these parameters is:

$$I_{\text{pre}} = (\hat{S}_{\text{pre}} - \lambda_{\alpha/2}d, \hat{S}_{\text{pre}} + \lambda_{\alpha/2}d) \quad (3.46)$$

where $1 - \alpha$ is the confidence level and $\alpha/2$ the area derived from the normal distribution $N(0, 1)$. The value $\lambda_{\alpha/2}$ is computed by solving $P(X > \lambda_{\alpha/2}) = \alpha/2$ and $X \in N(0, 1)$. If $\alpha/2 = (1 - 0.95)/2 = 0.05/2$ then the value $\lambda_{0.05/2} = 1.96$ can be found in a table of the standardized normal distribution [31].

If we use the confidence level 0.95 and would like an interval of range less than 0.1 then we estimate the sample size $n$ by the following inequality:

$$2 \cdot \lambda_{\alpha/2} \cdot \sqrt{\frac{\hat{S}_{\text{pre}}(1 - \hat{S}_{\text{pre}})}{n}} = 2 \cdot 1.96 \cdot \sqrt{\frac{\hat{S}_{\text{pre}}(1 - \hat{S}_{\text{pre}})}{n}} < 0.1 = 2 \cdot 0.05$$

$$\Rightarrow n \geq \frac{1}{4} \left( \frac{1.96}{0.05} \right)^2 \approx 400$$

since $\hat{S}_{\text{pre}}(1 - \hat{S}_{\text{pre}}) \leq \frac{1}{4}$ for all $\hat{S}_{\text{pre}} \in [0, 1]$.

By taking a random sample of size 400 from the whole bilingual lexicon we now can construct a confidence interval of range less than 0.1 and with the confidence level 0.95.

---

\(^6\)This is an application of the central limit theorem, i.e for large $n$ the sum $Y_n = X_1 + X_2 + \cdots + X_n$ is approximate $N(n\mu, \sigma\sqrt{n})$ and $Y_n/n$ is approximate $N(\mu, \sigma\sqrt{n})$. 

---
Chapter 4

The Computation by Singular Value Decomposition

From the theory of graph similarity and the proposed solution by SVD we now need to implement an algorithm which in an effective way generates the bilingual lexicon. To do this we have to investigate the term-by-document matrix generated from the text parser, and then find a method which we can use to compute the SVD of this matrix.

4.1 The Data Set and some Preprocessing issues

The data set is the minutes of an European Parliament meeting translated into Swedish and English. Each text collection contains 875857 documents where many of the documents can be considered as short sentences, i.e. many of the documents are containing just a few words. With this in mind it is possible that the text parser can delete a whole text document from the text collection if the document contains few words where the majority of these words are stop words. When we use GTP on the Swedish and English text collections, without any concern of the fact mentioned above, the resulting term-by-document matrices have different number of columns. It is then impossible to align the matrices according to the suggested approach (3.1). Neither GTP nor TMG (see Appendix E.1) generates any direct information if a text document in the text collection is stored, as a column vector in the term-by-document matrix, or not. However it is possible to overcome this problem by adding a tag to each text document. GTP consider terms that consist of letters and numbers as a term that should be parsed, if it is not in a stop list. Then by adding the tag doc1 to the beginning of the first document and doc2 to the beginning of the second document and so forth for each text document all the documents in the text collection will be parsed. Each tag appears only once in the whole text collection which means that we have to set the minimum local and global frequencies to 0. This is done by the option -d 0 -g 0 in GTP.

The resulting term-by-document matrices generated by GTP from the data
set above contain a term-by-document submatrix of size \(875857 \times 875857\). This submatrix is a diagonal matrix where the elements are the parsed terms \(doc1\) to \(doc875857\) and by removing this tag submatrix from the two term-by-document matrices we obtain the original term-by-document matrices which have the same number of columns and without the tags. The dictionaries created by GTP also contain these tags and they must be removed as well.

Apart from the tags many of the parsed terms can be considered as low frequent terms which, in a statistical point of view, are terms that are hard to match with each other. The precision of the generated automatic bilingual lexicon is likely to increase if we decide to remove these terms from the parallel texts. In the dictionary created by GTP the terms and the corresponding term frequencies are listed. By extracting all the low frequent terms from the dictionaries we can add these terms to the corresponding stop list. In our case a low frequent term is a term that appears less than 100 times in each of the two text collections. A short description of the measures taken to create the final term-by-document matrices and the corresponding dictionaries is

1. Add the tags \(doc1 - doc875857\) in the beginning of each document in the two text collections.
2. Parse each text collection with GTP, with the option \(-d 0 -g 0\).
3. Extract the low frequent terms \(f_{ij} < 100\) in our case) from each dictionary and put them in the corresponding stop list.
4. Parse the text collections again with GTP and the new stop lists.
5. Remove the tags from the dictionaries and the tag submatrix from the term-by-document matrices.

A more detailed description of the preprocessing and text parsing of the text collections is found in Appendix C.

### 4.2 Sparse Matrices and Data Matrix Formats

The text-by-document matrix generated from a text parser is (likely to be) sparse which means that a majority of the elements in this matrix is zero. In our case with the term-by-document matrices the proportion of non-zero elements in these matrices is less than 0.2%. To make use of the sparseness, when storing a sparse data matrix on the primary or the secondary memory of a computer, it is desirable to have a data matrix representation that only stores the non-zero elements and their corresponding position in the data matrix. There are several developed data matrix formats one can use to store a sparse data matrix and we will give a brief description of two of them.

#### 4.2.1 The Harwell–Boeing Format

The most intuitive way of representing a spares data matrix is by the Harwell–Boeing format e.g. used in MATLAB where for each non-zero element the position and the value is stored in a \(3 \times nnz\) array, where \(nnz\) is the number of non-zero elements. The non-zero element \(v_{ij}\) with indices \((i, j)\) is stored in the \(k:\text{th}\) position
of the \(2 \times n_{nz}\), where \(1 \leq k \leq n_{nz}\). We illustrate the Harwell–Boeing format with an example.

**Example 4.1.** The sparse matrix

\[
\begin{pmatrix}
0 & 0 & 0.3259 & 0.5945 \\
0.7445 & 0.3678 & 0 & 0 \\
0 & 0 & 0 & 0.3048 \\
0.0642 & 0 & 0 & 0
\end{pmatrix}
\]

is represented in the Harwell–Boeing format as:

\[
\begin{align*}
(2,1) & : 0.7745 \\
(4,1) & : 0.0642 \\
(2,2) & : 0.3678 \\
(1,3) & : 0.3259 \\
(1,4) & : 0.5945 \\
(3,4) & : 0.3048
\end{align*}
\]

The matrix above have been generated with the MATLAB command `sprand(m, n, density)` which creates a random \(m \times n\) sparse matrix with the proportion of non-zero elements \(\leq\) `density`. The sparse representation of this matrix in MATLAB is:

\[
\begin{align*}
(2,1) & : 0.7445 \\
(4,1) & : 0.0642 \\
(2,2) & : 0.3678 \\
(1,3) & : 0.3259 \\
(1,4) & : 0.5945 \\
(3,4) & : 0.3048
\end{align*}
\]

When using the MATLAB toolbox TMG for smaller text collections the generated term-by-document matrix is stored in this sparse matrix format.

### 4.2.2 The Compressed Column Storage

The compressed column storage, CCS, is another way to store a sparse data matrix. The CCS format is specified by three arrays `val(k)`, `rowind(k)` and `colptr(i)` where `val(k)` holds all the values of the non-zero elements for \(k = 1, \ldots, n_{nz}\) and `rowind(k)` holds the row index for each non-zero element. The last array `colptr(i)` stores the location in the array `val(k)` that starts a column, e.g. the \(i\):th column starts at `colptr(i)` and ends at `colptr(i + 1) - 1` for \(i = 1, \ldots, n\) where \(n\) is the number of columns in the data matrix. The length of the `colptr` array is by convention set to \(n + 1\).

**Example 4.2.** The sparse data matrix in the example (4.1) is with the CCS format

<table>
<thead>
<tr>
<th>val</th>
<th>0.7445</th>
<th>0.0642</th>
<th>0.3678</th>
<th>0.3259</th>
<th>0.5945</th>
<th>0.3048</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowind</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>colptr</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
A question that naturally follows from this way of storing a sparse data matrix is, if it is, possible to perform basic matrix operations on the matrix, such as matrix-vector multiplication and matrix-matrix multiplication.

If \( A := A_{n \times n} \) is a sparse data matrix stored with the CCS format, and \( x := x_n \) is a vector then the matrix-vector multiplication \( y = A^T x \) is defined as the sum
\[
y = \sum_{i=1}^{n} \sum_{j=1}^{n} (A^T)_{ij} x_j = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ji} x_j
\]
and is in pseudo-code given by
\[
\text{for } i = 1, n \\
\quad y(i) = 0 \\
\quad \text{for } j = \text{colptr}(i), \text{colptr}(i + 1) - 1 \\
\quad \quad y(i) = y(i) + \text{val}(j) \times x(\text{rowind}(j)) \\
\quad \text{end for} \\
\text{end for}
\]

To perform the usual matrix-vector multiplication, \( y = Ax \), we cannot use the sum
\[
y = \sum_{j=1}^{n} \sum_{i=1}^{n} a_{ij} x_i
\]
directly because it is traversing the rows of the matrix, which is a costly operation when the data matrix is stored with the CCS format. Thus by switching the indices we obtain the column-oriented version or \_AXPY\(^1\) which is
\[
\text{for all } i, \text{ do for all } j: y_j \leftarrow y_j + a_{ij} x_i
\]
and the pseudo-code is
\[
\text{for } j = 1, n \\
\quad y(j) = 0 \\
\text{end for} \\
\text{for } i = 1, n \\
\quad \text{for } j = \text{colptr}(i), \text{colptr}(i + 1) - 1 \\
\quad \quad y(\text{rowind}(j)) = y(\text{rowind}(j)) + \text{val}(j) \times x(i) \\
\quad \text{end for} \\
\text{end for}
\]
The matrix-matrix multiplications \( AX \) and \( A^T X \) are performed by repeating the matrix-vector multiplication \( Ax \) and \( A^T x \) for all the column vectors \( x \in X \).

The transposed term-by-document matrix generated from GTP is stored in the CCS format, which implies that the transposed matrix operations must be considered when performing the matrix operations described above, i.e. if \( \tilde{A} = A^T \) is the stored term-by-document matrix then
\[
y = \tilde{A}^T x = (A^T)^T x = Ax
\]
and
\[
y = \tilde{A} x = (A^T)x = A^T x.
\]
\(^1\)The \_AXPY (constant times a vector plus a vector) subroutine is a standard matrix operation implemented in the Fortran linear algebra routine library BLAS, which now is a part of the linear algebra package LAPACK [3].
This is of importance when these operations later are considered as basic subroutines in the main program used to compute the SVD of $\tilde{A}$.

### 4.3 A Krylov Decomposition Method

A well developed software for computing large scale eigenvalue problems is the Fortran 77 subroutine package ARPACK [30, 33]. The package contains several driver routines developed for different types of eigenvalue problems depending on the structure of the input data matrix. The main algorithm used by ARPACK is the implicitly restarted Arnoldi method which in turn is a Krylov decomposition method. In order to understand the basic theory of this method we have to consider some results about Krylov subspaces, the QR decomposition and the Arnoldi decomposition method [34, 35].

**Definition 4.1.** Let $A$ be a square matrix of size $n \times n$ and $v \neq 0$ a vector of size $n \times 1$. Then the sequence

$$v, Av, A^2v, A^3v, \ldots$$

is called a Krylov sequence based on $A$ and $v$. The matrix

$$K_k(A, v) = \begin{pmatrix} v & Av & A^2v & \cdots & A^{k-1}v \end{pmatrix}$$

is called the $k$:th Krylov matrix and the space spanned by this matrix is called the Krylov subspace, denoted by

$$K_k(A, v) = \text{span} \{v, Av, A^2v, \ldots, A^{k-1}v\}$$

The Krylov subspace plays an essential role when applying the Arnoldi decomposition method. This method forms an orthonormal basis for the Krylov subspace by an orthogonal projection of the data matrix onto this subspace. To orthogonalize the columns of $K_{k+1}$ one can use the QR decomposition.

**Theorem 4.1 (The QR Decomposition).**

Let $A$ be a matrix of size $m \times n$ with $m \geq n$. Then there is an orthogonal matrix $Q$ of size $m \times m$ and an upper triangular matrix $R$ of size $n \times n$ with non-negative diagonal entries such that

$$A = QR$$

(4.3)

If the columns of $A$ is linear independent then the matrix $R$ is non-singular.

**Proof.** See Golub and V. Loan [19].

The following approach is referred to the one used by Stewart [34] and it is a theoretical approach not suitable for scientific computations.

Let $K_{k+1}$ be the Krylov matrix obtained from the Krylov sequence in definition 4.1 then it is possible to express $K_{k+1}$ as

$$K_{k+1} = V_{k+1}R_{k+1}$$

(4.4)

where $V_{k+1}$ is a orthogonal matrix and $R_{k+1}$ is an upper triangular matrix. This factorization is the thin QR decomposition of $K_{k+1}$ and it can be shown that the
Krylov matrix $K_{k+1}$ can be eliminated from this equation. The equation (4.4) can be transformed to

$$\mathbf{A}\mathbf{V}_k = \mathbf{V}_{k+1}\tilde{\mathbf{H}}_k = \mathbf{V}_k\mathbf{H}_k + \beta_k v_{k+1} e_k^T$$

(4.5)

where $\tilde{\mathbf{H}}_k$ is an upper Hessenberg matrix i.e. an upper triangular matrix with a subdiagonal. This equation is called the $k$-step Arnoldi decomposition of $\mathbf{A}$ and is the consequence of multiplying the equation (4.4) with the matrix $\mathbf{A}$ from the left. The right hand side of the equation (4.5) is obtained by partitioning the matrix $\tilde{\mathbf{H}}_k$ such that

$$\tilde{\mathbf{H}}_k = \begin{pmatrix} \mathbf{H}_k & e_k^T \\ h_{k+1} & 0 \end{pmatrix} = \begin{pmatrix} \mathbf{H}_k \\ \beta_k e_k^T \end{pmatrix}$$

where $e_k^T$ is the $k$:th unit vector and $\mathbf{H}_k$ is an upper Hessenberg matrix.

If the value $\beta_k$ is small then we can see that the eigenvalues of $\mathbf{H}_k$ is an accurate approximation to the eigenvalues of $\mathbf{A}$ i.e.

$$\mathbf{A}\mathbf{V}_k \approx \mathbf{V}_k\mathbf{H}_k$$

(4.6)

From the Arnoldi decomposition it is possible to form a recursive algorithm for computing the vector $v_{k+1}$. Let $v_k$ be a vector satisfying $\|v_k\| = 1$ then it follows

$$\mathbf{A}v_k = \mathbf{V}_k h_k + \beta_k v_{k+1}$$

where $\mathbf{V}_k$ is orthogonal, and this implies

$$h_k = \mathbf{V}_k^T \mathbf{A}v_k$$

we desire to have $\|v_{k+1}\|_2 = 1$ and we obtain that from the observation

$$\beta_k v_{k+1} = \mathbf{A}v_k - \mathbf{V}_k h_k \Rightarrow \beta_k \|v_{k+1}\|_2 = \beta_k = \|\mathbf{A}v_k - \mathbf{V}_k h_k\|_2$$

and

$$v_{k+1} = \frac{1}{\beta_k} (\mathbf{A}v_k - \mathbf{V}_k h_k).$$
4.3. A Krylov Decomposition Method

The given algorithm derived from the above considerations is:

**The Arnoldi process**

1. Starting vector \( v_1 \), such that \( \|v_1\|_2 = 1 \).

2. for \( k = 1, 2, \ldots \)

3. \( h_k = V_k^T A v_k \)

4. \( w = A v_k - V_k h_k \)

5. \( \beta_k = h_{k+1} = \|w\|_2 \)

6. \( v_{k+1} = \frac{1}{\beta_k} v_k \)

7. end for

As mentioned previously, if \( \beta_k \) becomes small \(^2\) the eigenvalues of \( A \) can be approximated by the eigenvalues of \( H_k \). However, this requires that we let the algorithm continue, by successively increasing \( k \), until we obtain an acceptable approximation. This approach is not likely to be efficient when the size of the data matrix is large and we have to store the full matrix \( V_k \) on the primary memory. Even if the data matrix is sparse it does not imply that the matrix \( V_k \) will be sparse. If we instead store the matrix \( V_k \) on a secondary memory e.g. a disk the computer will spend most of the time, when running the algorithm, by communicating between the main memory and the disk. To overcome this problem a method of restarting the algorithm has been developed by Y. Saad \(^3\) and an implicitly restarted method by R. Lehoucq and D. Sorensen \(^4\).

The implicitly restarted Arnoldi method, IRAM, which is based on the Arnoldi decomposition and provides an efficient way to extract the desired eigenvalues. Assume that we have an \( m \)-step Arnoldi decomposition where \( m = k + p \) and \( k \) is the number of eigenvalues that we are interested in then the implicitly restarted Arnoldi method simply compresses the Krylov subspace onto a \( k \)-dimensional subspace where from the eigenvalues of interest can be extracted. When restarting the algorithm we have to find a new starting vector \( v'_1 \). This can be done by choosing a polynomial that, together with the original starting vector, computes the new starting vector. How this is done and what the underlying theory is behind implicitly restarting can be read in detail in [35, 34, 33].

4.3.1 Reverse Communication

The Fortran 77 subroutine library ARPACK is provided with IRAM and a reverse communication interface. The reverse communication interface is an important feature of ARPACK and gives the user the liberty to choose the data matrix format of the input data matrix, e.g. the CCS format, and the only operation that a user has to implement is a matrix-vector multiplication subroutine. The code segment where this action is taking place is

\[ \text{If } |\beta_k| < \epsilon_{tol}, \text{ where } \epsilon_{tol} \text{ is a tolerance, e.g. } e^{-8}. \]

\(^3\)Section 7.5 Arnoldi Method, pages 161 - 166 in [35].
\(^4\)Section 7.6 Implicitly Restarted Arnoldi Method, pages 166 - 185 in [35].
Chapter 4. The Computation by Singular Value Decomposition

4.4 The SVD Implementation with ARPACK

The problem we wish to solve with ARPACK is the SVD of the term-by-document matrix from the parallel texts in order to compute the similarity matrix $S_{12}$. The solution (3.34) indicates that we do not need to compute the right singular matrix $V_k$ explicitly, which implies that the main focus of the ARPACK routine should be to compute the partitioned left singular matrices $U^{(1)}_k$ and $U^{(2)}_k$ and the singular values of $\Sigma_k$.

If

$$A_p = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} = \begin{pmatrix} U^{(1)} \\ U^{(2)} \end{pmatrix} \Sigma V^T_1$$

is the full SVD of the term-by-document matrix from the parallel texts then the similarity matrix $S_{12}$ can be derived from the equation:

$$A_p A_p^T = \begin{pmatrix} A_1^T & A_2^T \end{pmatrix} \begin{pmatrix} A_1 & A_2 \end{pmatrix} = \begin{pmatrix} U^{(1)} & U^{(2)} \end{pmatrix} \Sigma^2 \begin{pmatrix} (U^{(1)})^T & (U^{(2)})^T \end{pmatrix} =$$

$$= \begin{pmatrix} U^{(1)} \Sigma^2 (U^{(1)})^T \\ U^{(2)} \Sigma^2 (U^{(1)})^T \end{pmatrix} \begin{pmatrix} U^{(1)} & U^{(2)} \end{pmatrix} \Sigma^2 (U^{(2)})^T =$$

$$= \rho \begin{pmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{pmatrix}.$$

This equation can be seen as a combination of the SS-SVD method and the VSM method (see chapter 3.6).

In the example program $dsvd$ provided with ARPACK, the SVD of a matrix is computed by solving the symmetric eigenvalue problem $(A^T A) v = \lambda v$, for $v \in \mathbb{V}$, and then the left singular matrix $U$ is obtained from the equation:

$$AV = \Sigma U$$

For our application we have to use the transposed version, i.e. $(A A^T) u = \lambda u$, for $u \in \mathbb{U}$. The number of eigenvalues to be computed is set by the integer $nev$. We also have to set the size of the uncompressed Krylov subspace; the number of Arnoldi basis vectors that the program may use, which is done with the integer $ncv$. A good rule of thumb is to let $ncv \geq 2 \cdot ncv$. The integers $nev$ and $ncv$ must not exceed the dimensions $m \times m$ of the square matrix $A A^T$, respectively.
neither can we compute more eigenvalues than \( m \) nor can the Krylov subspace be spanned by more basis vectors than \( \mathbf{A} \mathbf{A}^T \).

Our data matrix, the sparse term-by-document matrix generated from GTP, stored in the CCS format, is called with the subroutine \texttt{matin}(n, nz, colptr, rowind, val, infile). In the part of the program where the reverse communication is taking place we have to compute two matrix-vector multiplications since

\[
\mathbf{A} \mathbf{A}^T \mathbf{u} = \mathbf{A} (\mathbf{A}^T \mathbf{u}) = \mathbf{A} \tilde{\mathbf{u}} = \lambda \mathbf{u}
\]

and these multiplications are called with the subroutines \texttt{atv} and \texttt{av}. As mentioned previously our data matrix is transposed, therefore some caution must be taken in the implementation of the matrix-vector multiplication subroutines. If we let \( \tilde{\mathbf{A}} = \mathbf{A}^T \) then the first subroutine \texttt{atv} should compute \( \tilde{\mathbf{A}}^T \mathbf{u} = \mathbf{A} \mathbf{u} \) which is done with the code obtained from (4.2). Consequently the second subroutine \texttt{av} should compute \( \tilde{\mathbf{A}} \tilde{\mathbf{u}} = \mathbf{A}^T \tilde{\mathbf{u}} \) which is done with the code obtained from (4.1).

After the reverse communication step there is an optional loop which derives the right singular vectors, but we skip this step because we do not need these vectors to compute the similarity matrix \( \mathbf{S}_{12} \).

When the SVD is computed we would like to measure the difference between the original matrix and the approximation of the matrix. We will use the Frobenius norm to derive the residual and the relative residual defined as:

\[
\text{Residual} = \| \mathbf{A} - \overline{\mathbf{A}} \|_F \quad (4.7)
\]

\[
\text{Relative residual} = \frac{\| \mathbf{A} - \overline{\mathbf{A}} \|_F}{\| \mathbf{A} \|_F} \quad (4.8)
\]

where \( \overline{\mathbf{A}} \) is the approximation of \( \mathbf{A} \). In order to compute the residual of \( \mathbf{A} \) and the approximation \( \overline{\mathbf{A}} \) by SVD we do the following observation:

\[
\| \mathbf{A} - \mathbf{U}_k \Sigma_k \mathbf{V}_k^T \|_F^2 =
\]

\[
= \| \mathbf{A} \|_F^2 + \| \mathbf{U}_k \Sigma_k \mathbf{V}_k^T \|_F^2 - 2 \cdot \langle \mathbf{A}, \mathbf{U}_k \Sigma_k \mathbf{V}_k^T \rangle =
\]

\[
= \| \mathbf{A} \|_F^2 + \| \mathbf{U}_k \Sigma_k \mathbf{V}_k^T \|_F^2 - 2 \cdot \text{tr}(\Sigma_k^2) =
\]

\[
= \| \mathbf{A} \|_F^2 + \| \Sigma_k \|_F^2 - 2 \cdot \| \Sigma_k \|_F^2 = \| \mathbf{A} \|_F^2 - \| \Sigma_k \|_F^2
\]

\[
\Rightarrow \| \mathbf{A} - \mathbf{U}_k \Sigma_k \mathbf{V}_k \|_F = \sqrt{\| \mathbf{A} \|_F^2 - \| \Sigma_k \|_F^2}
\]

The relative residual is then:

\[
\sqrt{\| \mathbf{A} \|_F^2 - \| \Sigma_k \|_F^2} \over \| \mathbf{A} \|_F
\]

To obtain the Frobenius norm of \( \mathbf{A} \) we apply the BLAS function \texttt{dnrm2} on the \( \text{val} \) array, which computes the Euclidean norm of a vector, since

\[
\| \mathbf{A} \|_F^2 = \sum_i \sum_j a_{ij}^2 = \| \text{vec}(\mathbf{A}) \|_2^2 = \| \text{val} \|_2^2.
\]
4.4.1 The Implementation of the Automatic Bilingual Lexicon Acquisition Algorithm

The algorithm presented in section 3.7 to extract the bilingual lexicon from $S_{12}$ consists of a matrix-vector multiplication to compute the columns of $S_{12}$ and a maximum function to find the indices and the corresponding maximum value of the elements in the column vector.

The most efficient way to perform a full matrix-vector multiplication in Fortran is with the level-2 BLAS subroutine $\text{dgemv}$ and together with a subroutine to compute the maximum value and indices of a vector the Fortran code segment of the algorithm when using the SS-SVD method is:

```fortran
  do 10 i = 1, \text{min}(t1,t2)
    call dgemv ('N', t1, nev, one, u1s2, t1, u2row, 1, one, s12j, 1)
    call maxvalue (t1, s12j, mval, mind)
    write(nout, *) mind, i, mval
  10 continue
```

where $u1s2$ is the matrix $U^{(1)}_k \Sigma^2_k$ and $u2row$ is the $i$:th row of $U^{(2)}_k$ i.e. the $i$:th column of $(U^{(2)}_k)^T$. The subroutine $\text{maxvalue}$ computes the maximum value of the column vector $s12j$ and also returns the indices of this value. These indices and the maximum value are then written, as an unsorted list, to a file.

The implementation of the algorithm, when using the VSM-SVD method is done in similar manner by replacing $u1s2$ with the corresponding matrix $\hat{U}^{(1)}_k \hat{\Sigma}_k$ and $u2row$ with the corresponding vector $(\hat{V}^{(1)}_k)^i$.

To compute the precision of the translations from the bilingual lexicon we use the MATLAB interface and the statistical model described above.

4.5 Tests and Results

The theory presented so far is sufficient for constructing an automatic bilingual lexicon. The purpose of the tests below is to see how the precision of the translations in the bilingual lexicon is affected by the number of basis vector used in the approximated solution of the SVD. The first test was performed with the SS-SVD method and the second and third test was performed with the VSM-SVD method. In these tests we included the results obtained by the VSM method. The data sets were preprocessed according to the description in section 4.1.

4.5.1 Test 1 with SS-SVD

The data sets of the first test were the Swedish and English term-by-document matrices generated from GTP, with the dimensions $6901 \times 875857$ and $5691 \times 875857$ respectively. These matrices corresponded to a sparse data matrix stored in the CCS format with 15808242 non-zero elements and a column pointer, colptr, of length 12593. A small submatrix of the parallel text matrix is illustrated in figure 4.7 with the program vismatrix.  

\footnote{http://www.stanford.edu/~dgleich/programs/vismatrix/}
4.5. Tests and Results

A test was performed at NSC\(^6\) on the shared memory supercomputer Mozart\(^7\). In this test we did not parallelize the serial code, instead we ran the serial code on one processor with access to the shared memory.

The similarity matrices \(S_{12}\) and the bilingual lexicons were computed by increasing the numbers of left singular vectors from the SVD. Then the estimated precisions of each bilingual lexicon were derived from the statistical model (3.46) with a 95 \% confidence interval of range less than 0.1.

<table>
<thead>
<tr>
<th># basis vectors</th>
<th>100</th>
<th>250</th>
<th>500</th>
<th>1000</th>
<th>2000</th>
<th>3000</th>
<th>4000</th>
<th>5000</th>
<th>6000</th>
<th>7000</th>
</tr>
</thead>
<tbody>
<tr>
<td>precision (\hat{S}_{pre})</td>
<td>0.23</td>
<td>0.36</td>
<td>0.44</td>
<td>0.52</td>
<td>0.59</td>
<td>0.64</td>
<td>0.65</td>
<td>0.72</td>
<td>0.74</td>
<td>0.76</td>
</tr>
</tbody>
</table>

Table 4.1: The number of basis vectors used and the precision \(\hat{S}_{pre}\).

![Figure 4.1: A 95 \% confidence interval over the precision \(\hat{S}_{pre}\).](image)

The table (4.1) and the figure (4.1) shows that the precision is non-decreasing when increasing the numbers of basis vectors. The maximum number of basis vectors used in this test was 7000 due to the time, it took almost 18 h to compute the SVD on Mozart, and due to the \(ncv\) integer which in this case was set to 12580 < 2 \cdot 7000 = 14000, i.e. less than recommended. This was unavoidable because the dimensions of the problem was 12592 \times 12592 which \(ncv\) not could exceed.

The relative residual between the original data matrix and the approximated solution by SVD, was also computed and is presented in the table (4.2). It is

---

\(^6\) National Supercomputer Center in Linköping
\(^7\) A Linux based supercomputer with 64 processors and 512 GiB memory
remarkable that when we use 2000 basis vectors the precision is nearly 60% and the relative residual is 0.46 i.e. a good result in the view of the precision but a rather poor result in the view of the relative residual. We can see in figure (4.2) that the first 2000 singular values are significantly larger than the remaining 5000, which also shows in the relative residuals.

4.5.2 Test 2 with VSM-SVD

The data set of test 2 was the matrix product $A_1A_2^T$ with the Swedish and English term-by-document matrices generated from GTP. The data matrix is illustrated in figure 4.8. The dimensions of this matrix was $6901 \times 5691$ and it was stored in the CCS format with a $\text{vol}$ array consisting of 12024208 non-zero elements and a $\text{colptr}$ array of length 6902. This test was performed with a home computer \(^8\) with Ubuntu Linux 7.10 (Gutsy Gibbon) and ATLAS \(^9\) for minimizing the time consumed by the BLAS subroutines used in ARPACK.

The table (4.3) and the figure (4.3) shows how the precision varies over the first 4700 singular basis vectors used in VSM-SVD. The precision is decreasing a little when using over 3000 basis vector, which can be an effect of that the $\text{nev}$

\[\begin{array}{cccccccccc}
# \text{basis vectors} & 100 & 250 & 500 & 1000 & 2000 & 3000 & 4000 & 5000 & 6000 & 7000 \\
\text{rel. res.} & 0.85 & 0.77 & 0.69 & 0.58 & 0.46 & 0.38 & 0.32 & 0.27 & 0.23 & 0.19
\end{array}\]

Table 4.2: The number of basis vectors used and the relative residual.
4.5. Tests and Results

<table>
<thead>
<tr>
<th># basis vectors</th>
<th>100</th>
<th>250</th>
<th>500</th>
<th>1000</th>
<th>2000</th>
<th>3000</th>
<th>4000</th>
<th>4700</th>
</tr>
</thead>
<tbody>
<tr>
<td>precision $\hat{S}_{pre}$</td>
<td>0.17</td>
<td>0.29</td>
<td>0.35</td>
<td>0.51</td>
<td>0.74</td>
<td>0.78</td>
<td>0.74</td>
<td>0.75</td>
</tr>
</tbody>
</table>

Table 4.3: The number of basis vectors used and the precision $\hat{S}_{pre}$.

integer i.e. the dimensions of the Krylov subspace, was set to $5680 \approx 2 \cdot 3000$. In the two remaining computations with 4000 and 4700 basis vectors we have exceeded the recommended value of \texttt{nev}. The relative residuals are small, but

![Figure 4.3: A 95 % confidence interval over the precision $\hat{S}_{pre}$.](image)

it does not have a notable effect on the precision. The logarithmic graph of the singular values in figure (4.4) shows that the first 1000 singular values are decreasing in value much faster than the remaining 3700.

When using the full SVD, i.e. the VSM method we obtain the estimated precision $\hat{S}_{pre} = 0.82$ with a 95 % confidence interval $I_{pre} = (0.78, 0.86)$.

<table>
<thead>
<tr>
<th># basis vectors</th>
<th>100</th>
<th>250</th>
<th>500</th>
<th>1000</th>
<th>2000</th>
<th>3000</th>
<th>4000</th>
<th>4700</th>
</tr>
</thead>
<tbody>
<tr>
<td>rel. res.</td>
<td>0.23</td>
<td>0.15</td>
<td>0.092</td>
<td>0.048</td>
<td>0.020</td>
<td>0.0092</td>
<td>0.0032</td>
<td>0.0012</td>
</tr>
</tbody>
</table>

Table 4.4: The number of basis vectors used and the relative residual.
Chapter 4. The Computation by Singular Value Decomposition

4.5.3 Test 3 with VSM-SVD

In the text parser GTP there is an option \(-s\) for normalizing the documents in the document space. The third test was performed on the data set \(A_1 A_2^T\) where the columns of \(A_1\) and \(A_2\) were normalized respectively. The normalization of documents is often applied when the documents are varying much in size, e.g. if one document contains about ten words and another document contains about one hundred words (see chapter 2.2).

The sparse data matrix of dimension 6892 \(\times\) 5691 was stored in the CCS format with a \(\text{val}\) array containing 11996324 non-zero elements and a column pointer, \(\text{colptr}\) of length 6893. The test was performed at the same computer as test 2 and the precision of the translations in the bilingual lexicons was estimated by a 95 \% confidence interval of range less than 0.1.

<table>
<thead>
<tr>
<th># basis vectors</th>
<th>100</th>
<th>250</th>
<th>500</th>
<th>1000</th>
<th>2000</th>
<th>3000</th>
<th>4000</th>
<th>4700</th>
</tr>
</thead>
<tbody>
<tr>
<td>precision (\hat{S}_{pre})</td>
<td>0.19</td>
<td>0.29</td>
<td>0.36</td>
<td>0.52</td>
<td>0.76</td>
<td>0.76</td>
<td>0.74</td>
<td>0.71</td>
</tr>
</tbody>
</table>

Table 4.5: The number of basis vectors used and the precision \(\hat{S}_{pre}\).

The table (4.5) and the figure (4.5) shows how the precision varies over the first 4700 singular basis vectors used in VSM-SVD.

The precision was decreasing when using more than 2000 basis vectors, i.e. the best result was achieved when using about 1000 - 2000 basis vectors. The estimated result when applying the VSM method was \(\hat{S}_{pre} = 0.72\) with a 95 \% confidence interval \(I_{pre} = (0.68, 0.77)\) which is a worse result than we get with
VSM-SVD method used above.

Figure 4.5: A 95 % confidence interval over the precision $\hat{S}_{pre}$.

<table>
<thead>
<tr>
<th># basis vectors</th>
<th>100</th>
<th>250</th>
<th>500</th>
<th>1000</th>
<th>2000</th>
<th>3000</th>
<th>4000</th>
<th>4700</th>
</tr>
</thead>
<tbody>
<tr>
<td>rel. res.</td>
<td>0.26</td>
<td>0.17</td>
<td>0.11</td>
<td>0.056</td>
<td>0.024</td>
<td>0.011</td>
<td>0.0036</td>
<td>0.0013</td>
</tr>
</tbody>
</table>

Table 4.6: The number of basis vectors used and the relative residual.
Figure 4.6: A logarithmic graph for the 4700 first singular values.

Figure 4.7: A small submatrix of the sparse parallel text matrix illustrated with the program vismatrix. Each black dot represent a non-zero element.
Figure 4.8: The $\mathbf{A}_1 \mathbf{A}_2^T$ matrix used in test 2 illustrated with the program vis-matrix. Each black dot represents a non-zero element.
Chapter 4. The Computation by Singular Value Decomposition

4.6 Discussion of the Results

We can first conclude that the overall result is good and if we compare these results with them presented in [1] the improvement is nearly 20 percentage units. It also turns out that the VSM method, despite its simple interpretation and implementation, performs very well with an estimated precision of about 80%.

**SS-SVD:** In the results of the first test with SS-SVD we commented that the precision is remarkable good compared to the corresponding relative residual. This phenomena can be a result of that the Frobenius norm perhaps not is the ultimate measure to use when computing the relative residual and that the approximation by SVD has an underlying structure that selectively extracts the most important terms from the parallel texts and project the generated document space on a document subspace where these terms are represented.

**VSM-SVD:** When we use the VSM-SVD method in test 2 to compute the bilingual lexicon the relative residuals become much lower than those computed from test 1 when using the SS-SVD method. The cause of this is that in test 1 we approximate about 876000 documents with \( k \in (100, 7000) \) documents and in test 2 and test 3 we approximate about 7000 terms with \( k \in (100, 4700) \). The ratio between the approximated number of documents and the total number of documents in test 1 is much smaller than the corresponding ratio, for test 2 and test 3, between the approximated number of terms and the total number of terms, thus the differences in the relative residuals. In test 2 and test 3 with the we can see that the precision tends to decrease when the number of basis vectors used exceeds about 3000. As mentioned above we cannot use the recommended value on the \( \text{nev} \) integer due to the restriction of the dimensions on the Krylov subspace. This can be one factor explaining the decrease of the precision but it is not the only reason. One other explanation is that when we use more basis vectors we get a better approximation of the parallel text matrix which also implies that there is more noise added to the approximation. When we look at the logarithmic graph of the singular values we see that the last 2000 do not change much in value and therefore not improve the precision enough to overcome the added noise.

**VSM:** With the VSM method we obtain better results than with either SS-SVD or VSM-SVD with a precision about 80%. The only exception is, in test 3 with VSM-SVD, when the documents are normalized the result achieved by VSM is worse then the result with the SVD approximation of the VSM, i.e. here the noise reduction by the SVD approximation overcome the result computed by VSM.

To summarize we can conclude that the solution by all the three methods SS-SVD, VSM-SVD and VSM yields bilingual lexicon where the precision of the translations is remarkable good. In spite of the fact that the SVD approximation of the non-negative matrices \( A_p \) and \( A_1 A_2^T \) generates basis matrices containing negative elements the translations in the extracted bilingual lexicon have high precision due to the properties of the SVD and the hypothesis (3.1).
Chapter 5

The Computation by Non-negative Matrix Factorization

The term-by-document matrix generated from the parallel texts is non-negative and when we use the SVD to approximate a non-negative document vector (3.33), in the parallel text matrix, the approximated document vector can contain negative elements. How to interpret these elements with respect to the non-negativity of the parallel text matrix is unclear, therefore, in many applications, a non-negative representation is required.

In this chapter we will consider some theory concerning non-negative matrix factorization, NMF, and an algorithm suited for sparse matrices. The serial code of this algorithm is parallelized in Fortran 90 by OpenMP, and the bilingual lexicon is computed by the same model described in chapter 3.7.

5.1 The Non-negative Matrix Factorization Problem

If \( A \in \mathbb{R}^{m \times n} \) is a non-negative matrix i.e. all elements in \( A \) are greater or equal to zero then the non-negative factorization of \( A \) can be found if there for some \( k < \min(m, n) \) exists non-negative matrices \( W \in \mathbb{R}^{m \times k} \) and \( H \in \mathbb{R}^{k \times n} \) such that the functional

\[
 f(W, H) = \|A - WH\|_F^2
\]

is minimized. The aim is to derive a rank \( k \) approximation of \( A \) such that

\[
 A \approx WH
\]

where \( W \geq 0 \) and \( H \geq 0 \).

The solution of this minimization problem is not unique, since we can insert a \( k \times k \) positive invertible diagonal matrix \( D \) into the solution:

\[
 WH = WDD^{-1}H = (WD) (D^{-1}H) = \tilde{W}\tilde{H}.
\]

Törnfeldt, 2008.
Proposition 5.1. The functional $f(W, H)$ is convex in $W$ and $H$ separately.

Proof 5.1. We only prove it for $W$, since the proof for $H$ is derived in similar fashion.

Let $0 \leq \lambda \leq 1$ be a constant and $W^{(1)}$ and $W^{(2)}$ be two non-negative matrices, then $f(W, H)$ is a convex functional in $W$ if the following is true:

$$f(\lambda W^{(1)} + (1 - \lambda)W^{(2)}, H) \leq \lambda f(W^{(1)}, H) + (1 - \lambda)f(W^{(2)}, H)$$

(5.2)

The left-hand side is:

$$f(\lambda W^{(1)} + (1 - \lambda)W^{(2)}, H) = \|A - (\lambda W^{(1)} + (1 - \lambda)W^{(2)})H\|_F^2 = \|A\|_F^2 + \lambda\|W^{(1)}H\|_F^2 + (1 - \lambda)\|W^{(2)}H\|_F^2 - 2\langle A, \lambda W^{(1)}H + (1 - \lambda)W^{(2)}H \rangle$$

and the right-hand side is:

$$\lambda f(W^{(1)}, H) + (1 - \lambda)f(W^{(2)}, H) = \lambda\|A - W^{(1)}H\|_F^2 + (1 - \lambda)\|A - W^{(2)}H\|_F^2 = \|A\|_F^2 + \lambda\|W^{(1)}H\|_F^2 + (1 - \lambda)\|W^{(2)}H\|_F^2 - 2\langle A, \lambda W^{(1)}H + (1 - \lambda)W^{(2)}H \rangle$$

The left-hand side and the right-hand side are equal, thus the condition (5.2) holds.

This proposition is not true for $f(W, H)$ in both $W$ and $H$ simultaneous, i.e. $f(W, H)$ is non-convex. This implies that if $W^*H^*$ is a local minimum to (5.1) then we cannot tell if there are other local minimum solutions that are improving the optimality.

This brief analysis shows that there are no analytical method for finding the best solution to (5.1) but instead there are several developed numerical algorithms based on different types of optimization methods. These methods often find a stationary point that can be a local minimum, i.e. the solution depends on the initialization of $W$ and $H$ and the convergence of the algorithm used. We will not discuss this matter any further but there is ongoing research about this [28] and published articles e.g. [9].

The term-by-document matrix $A_p$ obtained from the parallel texts is a non-negative matrix and with a rank $k$ NMF approximation we can express each document $a_j$ in $A_p$ as:

$$a_j = \begin{pmatrix} a_{1j} \\ a_{2j} \end{pmatrix} \approx \begin{pmatrix} W^{(1)}_k \\ W^{(2)}_k \end{pmatrix} h_j = \begin{pmatrix} c_{1j} \\ c_{2j} \end{pmatrix} = c_j$$

(5.3)

where

$$W = \begin{pmatrix} W^{(1)}_k \\ W^{(2)}_k \end{pmatrix}$$

is the non-negative basis matrix spanning the approximated document space and $h_j$ is a non-negative coefficient vector holding the coordinates for document $j$. The approach (5.3) has a distinct interpretation due to the non-negativity constraint, compared to approach (3.33).
In order to extract a bilingual lexicon with NMF we will use NMF to approximate the similarity matrix obtained from the VSM method:

$$A_1 A_2^T \approx W_{m \times k} H_{k \times q}$$  \hspace{1cm} (5.4)

where $A_1 \in \mathbb{R}^{m \times n}$ and $A_2 \in \mathbb{R}^{q \times n}$. We will use the notation VSM-NMF when referring to this method.

### 5.2 The GD-CLS Algorithm

The algorithm we will use to compute the non-negative matrix factorization is called GD-CLS [32] which stands for gradient descent with constrained least squares and is a hybrid method between a multiplicative method and a least squares method.

The multiplicative method (MM) [27] has become a standard method for non-negative matrix factorization due to its simple implementation and practical use. The convergence of this method is not certain which can be showed with the obtained KKT-conditions [5].

The multiplicative method uses multiplicative update rules and can be described as follows:

---

**The multiplicative method**

1. Initialize $W$ and $H$ with non-negative values.

2. Iterate for each $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$ and $l = 1, 2, \ldots, k$ until convergence or after maximum number of iterations.
   
   (a) $W_{il} \leftarrow W_{il} \frac{(A H^T)_{il}}{(W H H^T)_{ij} + \epsilon}$

   (b) $H_{lj} \leftarrow H_{lj} \frac{(W^T A)_l}{(W^T W H H^T)_{ij} + \epsilon}$

---

The initialization is often done by taking random non-negative matrices $W$ and $H$ and the convergence is depending on the structure of $A$. The positive constant $\epsilon \ll 1$ is added to avoid division by zero.

The least squares part of the GD-CLS algorithm originates from the alternating least square (ALS) algorithm which is based on the partial convexity of $W$ and $H$ in $f(W, H)$. The algorithm is solving the least squares problems alternately:

$$\min_{H \geq 0} \|A - WH\|_F$$

and

$$\min_{W \geq 0} \|A - WH\|_F$$

which has the solutions

$$H = (W^T W)^{-1} W^T A$$

and

$$W = AH^T \left( HH^T \right)^{-1}$$
respectively. The negative elements of the derived matrices $H$ and $W$ are then set to zero to fulfill the non-negativity constraint. The ALS algorithm is:

The ALS algorithm

1. Initialize $W$ with non-negative values.
2. Iterate for $i = 1, 2, \ldots, m$ until convergence or after maximum number of iterations, $m$.
   (a) Solve for $H$ the least squares problem 
       \[ \min ||A - WH||_F. \]
   (b) Set all the negative elements in $H$ to zero.
   (c) Solve for $W$ the least squares problem 
       \[ \min ||A - WH||_F. \]
   (d) Set all the negative elements in $W$ to zero.

The generated matrices $W$ and $H$ can be interpreted as a sparse representation of the input matrix $A$, i.e. $W$ is a sparse $m \times k$ basis matrix and $H$ is a sparse $k \times n$ coordinate matrix. In order to control the sparseness of the matrices $W$ and $H$ a NMF method with sparseness constraints [22] is used in the GD-CLS algorithm to penalize the number of non-zero entries in the matrix $H$.

The GD-CLS algorithm is:

The GD-CLS algorithm

1. Initialize $W$ with non-negative values.
2. Iterate for each $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$ and $l = 1, 2, \ldots, k$ until convergence or after maximum number of iterations.
   (a) Solve for $H_{j,l}$ the constrained least squares problem 
       \[ \min \left( ||A_{j,l} - WH_{j,l}||_2^2 + \lambda ||H_{j,l}||_2^2 \right). \]
   (b) Set all the negative elements in $H$ to zero.
   (c) Update $W$: 
       \[ W_{il} \leftarrow W_{il} \frac{(AH^T)_{il} - (WHH^T)_{il}}{||W||_F^2} \]

The constant $\lambda$ is a regularization value to control the sparseness of $H$ and a common value of $\lambda$ is 0.1. The algorithm solves a least squares problem to obtain $H$ and uses the multiplicative update rule to obtain $W$, thus a combination of MM and ALS with a sparsity constraint.

5.3 The Implementation of the GD-CLS Algorithm

The implementation of the GD-CLS Algorithm in Fortran 90 is fairly straightforward. To compute the required matrix-matrix multiplications we use the
5.3. The Implementation of the GD-CLS Algorithm

level-3 BLAS subroutine \texttt{dgemm} and to solve the least squares problem we use the LAPACK subroutine \texttt{dposv}.

The solution of the least squares problem

$$\min_{H} (\|A - WH\|_F + \lambda \|H\|_F)$$

is obtained by solving the system of linear equations for \(H\):

$$(W^T W + \lambda I) H = W^T A$$

(5.5)

The matrix \(B = (W^T W + \lambda I)\) is a \(k \times k\) symmetric and positive definite thus the system (5.5) can be solved by applying the Cholesky decomposition on the matrix \(B = U^T U\) such that:

$$(W^T W + \lambda I) H = W^T A \Rightarrow BH = W^T A \Rightarrow U^T UH = W^T A$$

where \(U\) is an upper triangular matrix. The system of linear equations is then solved by substitution of the matrices \(U^T\) and \(U\) respectively.

The Fortran code for the GD-CLS algorithm is:

```fortran
call random_number(W)
do iter = 1,maxiter
   call dgemm(W', W, B)
   B = B + lambdaI
   call dgemm(W', A, H)
   call dposv(B, H)
   where (H < zero)
      H = zero
   end where
   call dgemm(A, H', AHt)
call dgemm(H, H', HHt)
call dgemm(W, HHt, WHHt)
   W = W * (AHt / (WHHt + myeps))
end do
```

The subroutines \texttt{dgemm} and \texttt{dposv} do have more input parameters but the code above only contain the input and output matrices to make the code more easy to read. The subroutine \texttt{dgemm}(X, Y, Z) computes the matrix-matrix product \(Z = XY\) and the subroutine \texttt{dposv}(B, H) solves the equation \(BX = H\) by the Cholesky decomposition and put the result in the matrix \(H\). The notation \(X'\) is the transpose of \(X\) which is the notation used in MATLAB. The second last line of the code is the multiplicative update rule and the multiplication and the division is automatically done element-wise (the Hadamard product) in Fortran 90.

The sparse data matrix \(A_1 A_2^T\) is allocated as a full matrix. The proportion of non-zero elements in this matrix is about 30 % and we have observed that the matrix-matrix multiplication with the subroutine \texttt{dgemm} between \(A_1 A_2^T\)
and some other full matrix takes a shorter time than if we do the matrix-matrix multiplication by iterating any of the matrix-vector multiplications obtained from the pseudo-codes of (4.1) and (4.2).

We have not used any convergence condition in this code due to the uncertainty of how to set up such a condition. Instead we have used a maximum number of iterations which in our case were set to 1000. The constant $\text{myeps}$ is a small value added to avoid division by zero and is set to $10^{-9}$.

### 5.3.1 Parallelization with OpenMP

The serial code above performs well on a home computer for small problems or if we set $k$ to a small value. We are interested in computing the NMF for $k \in (100, 5000)$ and this is not possible to do with one computer in a reasonable amount of time. Therefore we have chosen to parallelize the serial code with OpenMP\(^1\), which is a platform independent standard for parallel programming of shared memory systems. The theory of how to program parallel computers can be read in [12, 24].

The OpenMP code for the GD-CLS algorithm will be presented in a “pseudo-OpenMP” code with MATLAB like notation for readability.

#### The GD-CLS algorithm with OpenMP:

```matlab
call random_number(W)
!$omp parallel private (myid, nthr, mf, ml, nl, kf, kl, ... )
nthr = omp_get_num_threads()
myid = omp_get_thread_num()
mq = m / nthr
nq = n / nthr
kq = k / nthr
mf = myid * mq + 1
nf = myid * nq + 1
kf = myid * nq + 1

if (myid == nthr-1) then
  ml = (myid + 1) * qm + mod(m,nthr)
  nl = (myid + 1) * qn + mod(n,nthr)
  nk = (myid + 1) * qk + mod(k,nthr)
else
  ml = (myid + 1) * qm
  nl = (myid + 1) * qn
  nk = (myid + 1) * qk
end if
!$omp barrier

do iter = 1, maxiter
  B(:,kf:kl) = W' * W(:,kf:kl) % BLAS-3 routine DGEMM
  B(:,kf:kl) = B(:,kf:kl) + lambdaI(:,kf:kl)
  H(:,nf:nl) = W' * A(:,nf:nl) % BLAS-3 routine DGEMM
!$omp barrier
```

\(^1\)www.openmp.org
5.3. The Implementation of the GD-CLS Algorithm

!$omp master
B = U' * U % Cholesky factorization
!$end master % with the LAPACK routine % DPOTRF

H(:,nf:nl) = inv(U' * U) * H(:,nf:nl) % Solving the system % of linear equations % with the LAPACK routine % DPOTRS

where (H(:,nf:nl) < zero)
H(:,nf:nl) = zero
end where

!$omp barrier

AHt(mf:ml,:) = A(mf:ml,:) * H' % BLAS-3 routine DGEMM
HHt(:,kf:kl) = H * H(:,kf:kl)' % BLAS-3 routine DGEMM
WHHt(:,kf:kl) = W * HHt(:,kf:kl) % BLAS-3 routine DGEMM

!$omp barrier
end do
!$omp end parallel

The beginning of the code is an initialization of pointers such that each thread knows which columns or rows it holds when performing e.g. a matrix-matrix multiplication. These pointers must be private since they are different for each thread. The rest of the code is a parallelized version of the serial code above, where each matrix-matrix multiplication is performed using the pointers and the Cholesky decomposition is done by first factorizing the matrix in a master thread and then the system of linear equations is solved by letting each thread solve a part of the system using the pointers. We have to use some barriers in order to maintain the synchronization, e.g. before we can do the Cholesky factorization step all threads must finish their matrix-matrix multiplications.

We ran this parallelized code on the shared memory system Mozart at NSC with a full $2000 \times 1000$ matrix and with $k = 500$ and obtained the following times:

<table>
<thead>
<tr>
<th># processors p</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>time (sec)</td>
<td>1295</td>
<td>806</td>
<td>443</td>
<td>335</td>
<td>272</td>
<td>255</td>
</tr>
</tbody>
</table>

Table 5.1: The expenditure of time for the parallelized GD-CLS algorithm.

The speed-up $S_p$ and the efficiency $E_p$ of the algorithm were then computed by the equations:

$$S_p = \frac{T_1}{T_p}$$

and

$$E_p = \frac{S_p}{p}$$
where $T_1$ is the time required to execute the code on one processor and $T_p$ the time requires to execute the code on $p$ processors. The speed-up and the efficiency is presented in the table and graphs below.

<table>
<thead>
<tr>
<th># processors $p$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed-up $S_p$</td>
<td>1.00</td>
<td>1.61</td>
<td>2.92</td>
<td>3.87</td>
<td>4.76</td>
<td>5.08</td>
</tr>
<tr>
<td>Efficiency $E_p$</td>
<td>1.00</td>
<td>0.80</td>
<td>0.73</td>
<td>0.64</td>
<td>0.60</td>
<td>0.42</td>
</tr>
</tbody>
</table>

Table 5.2: The speed-up and efficiency of the parallelized GD-CLS algorithm.

The speed-up is not ideal but acceptable because the Cholesky factorization is performed with one master thread and requires the same amount of time to be executed independent of how many processors we use. When we have more than eight processors the expected time reduction is limited due to the time it required to communicate between the processors is increasing.

The efficiency is decreasing due to the same arguments as for the speed-up above, and we obtain a mean of about 70 % which is marked in the figure (5.2) with a dashed line.
5.4 Test and Result with VSM-NMF

The test with the NMF algorithm GD-CLS to extract a bilingual lexicon was performed on the same data set used in test 2 of the VSM-SVD computation, and the precision of the translations were verified with a 95% confidence interval of range less than 0.1. The number of iterations were set to 1000 and the regularization constant $\lambda = 0.1$. The parallelized GD-CLS algorithm were executed on eight processors on Mozart at NSC and the obtained precision of the extracted bilingual lexicon is presented in table (5.3) and figure (5.3). The

<table>
<thead>
<tr>
<th># basis vectors</th>
<th>100</th>
<th>250</th>
<th>500</th>
<th>1000</th>
<th>2000</th>
<th>3000</th>
<th>4000</th>
<th>5000</th>
</tr>
</thead>
<tbody>
<tr>
<td>precision $\widehat{S}_{pre}$</td>
<td>0.32</td>
<td>0.47</td>
<td>0.50</td>
<td>0.49</td>
<td>0.55</td>
<td>0.59</td>
<td>0.63</td>
<td>0.62</td>
</tr>
</tbody>
</table>

Table 5.3: The number of basis vectors used and the precision $\widehat{S}_{pre}$.

times for $100 \leq k \leq 5000$ are presented in table (5.4).

<table>
<thead>
<tr>
<th># basis vectors</th>
<th>100</th>
<th>250</th>
<th>500</th>
<th>1000</th>
<th>2000</th>
<th>3000</th>
<th>4000</th>
<th>5000</th>
</tr>
</thead>
<tbody>
<tr>
<td>rel. res.</td>
<td>0.24</td>
<td>0.15</td>
<td>0.10</td>
<td>0.060</td>
<td>0.044</td>
<td>0.037</td>
<td>0.030</td>
<td>0.019</td>
</tr>
<tr>
<td>time (h)</td>
<td>0.24</td>
<td>0.42</td>
<td>0.92</td>
<td>1.91</td>
<td>4.68</td>
<td>9.53</td>
<td>14.12</td>
<td>21.23</td>
</tr>
</tbody>
</table>

Table 5.4: The number of basis vectors used and the expenditure of time.
We only managed to do one test because the expenditure of time was too high in order to do more tests.

5.5 Discussion of the Results

These results are similar to those presented in [1]. We can see that the precision is increasing much in the beginning and then when the number of basis vectors used is 3000 to 5000 the precision is about 60\% and does not tend to change much in value.

It is hard to know why we do not obtain higher precision with the GD-CLS algorithm. Perhaps more iterations are needed when we increase the number of basis vectors and that a refinement of the initialization of the matrix $W$ is required in order to obtain faster convergence.

The expenditure of time when increasing the numbers of basis vectors is not satisfying. If we had executed the algorithm on one processor with $k = 5000$ then it would take approximately one week to perform all the iterations.

In order to optimize the parallel code we would need to rewrite the Cholesky factorization step such that the part performed with the master thread could be performed with all the threads. There are available software\footnote{ScaLAPACK: http://www.netlib.org/scalapack/scalapack_home.html} containing the parallelized versions of the LAPACK routines, but these are developed for distributed memory systems and MPI [10].

We can conclude that this algorithm works well if we are interested in extracting the bilingual lexicon using only a few of the basis vectors, e.g. 100 to 500, but if we use more basis vectors the precision does not improve much.
Chapter 6

Comparison of the Results, Summary and Conclusions

The theory of graph similarity and the vector space model constitute the two parts which were necessary to combine in order to perform any of the computations described in chapter 4 and chapter 5. The comparison of the results obtained from these computations are presented here. We summarize the theory and the results and present some final conclusions. The chapter ends with a short discussion about the future work that can be done and some further improvements.

6.1 Comparison of the Results

First of all we can conclude that the overall results derived from the computations by SVD and NMF are good, compared with the corresponding results presented in [1]. Sahlgren and Karlgren achieve a precision of about 60 % and we obtain about the same results with the NMF algorithm GD-CLS and approximately 15 to 20 percentage units better results with the computations by SVD.

When we apply the VMS method to describe the similarity matrix $S_{12}$ the precision of the translations in the extracted bilingual lexicon is about 80 %. This result is better than any of results obtained from the computations by SS-SVD, VSM-SVD or VSM-NMF. It is interesting that the method easiest to implement actually generates the best results.

If we compare the results from test 2 with the VSM-SVD computation and the results from the test with the VSM-NMF computations we can see the differences in figure (6.1).

The GD-CLS algorithm performs better than the SVD algorithm in the beginning when we use $0 \leq k \leq 900$, but when $k > 900$ the precision obtained from the VSM-SVD computation increases much faster then the corresponding precision obtained from the VSM-NMF computation. In the end of the two graphs we see that the confidence intervals coincide, i.e. it is possible that the precisions from the two different methods have similar values.

The computations by VSM-SVD and ARPACK were possible to perform on a home computer, but the corresponding computations with NMF-VSM
Figure 6.1: A 95% confidence interval over the precision $p_s$. The dashed line with circles are the results obtained with VSM-SVD, and the solid line with squares are the results obtained with VSM-NMF and the NMF algorithm GD-CLS.

and the GD-CLS algorithm had to be performed on a supercomputer in order to execute the algorithm in a reasonable amount of time. The time taken to execute the SVD algorithm using the SVD-VSM method with e.g. $k = 4000$ on one processor is less\footnote{Approximately five to six hours on one processor with SVD-VSM compared to approximately one week on one processor with VSM-NMF and GD-CLS.} than the time taken to execute the GD-CLS algorithm using the VSM-NMF method on eight processors with the same $k$.

At the most it differ about 20 percentage units between the two methods in favour of the SVD algorithm (with the SS-SVD and VSM-SVD methods) and therefore there are no doubts that the SVD algorithms are better to use when it comes to the precision of the translations in the bilingual lexicon and also the time taken in order to execute the algorithms.

### 6.2 Summary and Conclusions

In this report we have carefully analyzed the concept of parallel texts and graph similarity with the aim to create a method for automatic bilingual lexicon acquisition. The parallel texts were preprocessed and parsed with the text parser GTP and then with the approach (3.1) we obtained a matrix representation of the parallel texts. We also discovered that the parallel texts could be represented by a directed tripartite graph and an adjacency matrix containing the term-by-document matrices generated from the parallel texts.

In Appendix C there are more details concerning the preprocessing and text parsing of the parallel texts. It took many trails before we managed to construct
the parallel text matrix, partly because we, in order to obtain the same number of column vectors in the to term-by-document matrices, had to come up with some idea of how this could be done. There were some trails and errors before we thought of tagging the documents and in that way constructing the parallel text matrix. Neither TMG nor GTP generates any direct information about which documents of the text collection that are parsed and put in the term-by-document matrix, therefore we can conclude that it would be good if some sort of index containing information about the structure of the generated term-by-document matrix could be extracted when using these text parsers.

We analyzed the theory of graph similarity and applied it on the adjacency matrix and we found out that the similarity matrix has a special matrix block structure and could be derived by computing the Perron root of an eigenvalue problem. This similarity matrix was computed by applying the rank one approximation obtained from the SVD on the parallel text matrix and it turned out that this solution would give a poor result with respect to the precision of the translations in the latter generated bilingual lexicon.

The computed Perron root was $\pm \rho$ which generated more than one possible solution of the self-similarity equation, i.e. we can conclude that the self-similarity matrix $S$ is not unique. In the case when we have the undirected parallel text graph we obtain infinitely many solutions (see Appendix B).

Instead of using the rank one solution we introduced a possible method in order to get a rank $k$ solution. The parallel text matrix was non-negative but the rank $k$ solution with SVD contained possible negative elements. From this rank $k$ solution we derived three possible methods for representing the similarity matrix $S_{12}$: SS-SVD, VSM and VSM-SVD.

The interpretations of the approximated SVD solution with the methods SS-SVD and VSM-SVD were in this case indistinct but we discovered by several independent computations that the maximum elements of the column vectors in the similarity matrix were positive and we formulated a hypothesis (3.1) describing this phenomena. In order to extract the bilingual lexicon from the similarity matrix we presented an algorithm and a statistical model used for estimation of the obtained precisions.

Perhaps it is possible, with stronger condition for the hypothesis, to prove it but we did not analyze that any further. The algorithm applied to extract the bilingual lexicon from the similarity matrix is based on the hypothesis, and with respect to the good results obtained by the SVD the hypothesis is likely to be true.

The bilingual lexicon was computed by four different methods; SS-SVD, VSM, VSM-SVD and NMF-SVD (see figure 6.2 for a flowchart of the computations).

We did not think of the VSM method before the results from test 1 were computed, and we have showed that this method actually generates the best results in this case. In other applications, e.g. query matching, the vector space model does not usually generates the best results therefore it was a little surprising when we saw that it did in this case.

In the SVD methods SS-SVD and VSM-SVD the needed SVD algorithms were implemented with IRAM and ARPACK.

ARPACK is a very robust and powerful package and it is relatively easy to use. In our case it did not require much time to get a working SVD algorithm, and when we started to use the vector space model application we could perform
the tests on a home computer.

The parallel text matrix was non-negative which suggested the use of an non-negative matrix factorization method, NMF. We presented the theory behind NMF and the algorithm GD-CLS was implemented by parallelizing the serial code of the algorithm with OpenMP.

The VSM-NMF method with the GD-CLS algorithm did not perform as well as we had hoped for. In other applications, e.g. topic extraction from text collections [32] the GD-CLS algorithm has shown to give good results. The GD-CLS algorithm has a clustering effect on the text documents, i.e. when we increases the number of basis vectors, \( k \), we simultaneously increase the number of clusters that the documents can be placed in. It seems that when \( k \) exceeds 3000 the result is not affected if we add more clusters. These clusters must in some sense be very alike, and it is possible that they overlap each other in the approximated document space and therefore do not improve the result.

The results from these two methods shows that the performance of the SVD algorithm is better than the GD-CLS algorithm. We obtained the best result by the vector space model approach with a precision of about 80%.

### 6.3 Future Work and Improvements

This masters’ thesis project has shown that the computations by SVD and the application with the vector space model generates good results with high precision of the translations in the bilingual lexicon. A future work could be to implement an interface which combines a text parser with this method for automatic bilingual lexicon acquisition. The application of the vector space model is simple to implement and only requires a matrix-matrix multiplication, since it is fast and yields good results.

It would be interesting to test more NMF algorithms and see if it is possible to improve the results. The parallelized code we used must be optimized in order to obtain better performance. It could be done either by translating the code such that it can be executed on a distributed memory system with MPI and ScaLAPACK or by writing a parallelized routine for the Cholesky factorization step in OpenMP. The area of constructing good NMF algorithms is an ongoing research area and hopefully there will be improved algorithms taking the non-convexity, initialization and the convergence into account.

The graph similarity model used is not bound to this particular application, for instance we could add more languages to the graph and then generate a multilingual lexicon. If we have \( k \) languages the obtained graph will be a \( k + 1 \)-partite graph and the solution of the similarity matrix \( S_{ij} \) given with the vector space model is then:

\[
S_{ij} = A_i A_j^T
\]

where \( i \) and \( j \) denote the languages.

Instead of matching words in two different languages, we could match images, short movies, topics from medical journals, court sentences, DNA, sports results etc. The list of possible applications is endless and we can conclude that the graph similarity model is interesting and useful in many data mining applications.
6.3. Future Work and Improvements

Figure 6.2: A flowchart describing the computational steps, beginning with the parallel texts and ending with the VSM-SVD and VSM-NMF methods.
Appendix A

Perron-Frobenius Theory

In this chapter we will present some results about positive and non-negative matrices (Meyer [29]). The theory was developed in the beginning of the 20th century by the mathematicians Oscar Perron and Ferdinand Georg Frobenius, hence the name of the theory.

We use the notation $A > 0$ to say that the matrix $A$ is positive, which means that all the elements $a_{ij}$ of $A$ are positive, i.e. $a_{ij} > 0, \forall i, j$. Similarly we use the notation $A \geq 0$ to say that the matrix $A$ is non-negative, which means that all the elements $a_{ij}$ of $A$ are non-negative, i.e. $a_{ij} \geq 0, \forall i, j$. All positive matrices $A > 0$ are automatically non-negative, since the set of all positive matrices is a subset in the set of all non-negative matrices.

The following results are known as Perron’s theorem and applies for positive matrices.

**Theorem A.1** (Perron’s Theorem). If $\mathbb{R}^{n \times n} \ni A > 0$ and $r = \rho(A)$. Then it follows that $\sigma(A) \ni r > 0$ and there exists an eigenvector $v > 0$ such that $Av = rv$. The value $r$ is called the Perron root and the vector $v_r = v/\|v\|_1$ is called the Perron vector, where

$$\|v\|_1 = \sum_{i=1}^{n} v_i$$

is the 1-norm of $v$.

**Proof.** The proof can be read in Meyer [29, p. 663].

For non-negative matrices it is possible to find a non-negative eigenpair, i.e. it is possible that the Perron root is zero.

**Theorem A.2.** If $\mathbb{R}^{n \times n} \ni A \leq 0$ then $A$ has a non-negative eigenpair $(r, v)$ where $r$ is the spectral radius $\rho(A) \in \sigma(A)$ and $v \geq 0$ is an associated eigenvector.

**Idea of the proof.** The main idea of the proof is to construct a positive sequence $A_k = A + \frac{1}{k}E > 0$, where $E$ is the matrix of all ones, and then construct sequences $(r_k)_{k=1}^\infty$ and $(v_k)_{k=1}^\infty$ where $r_k$ is the Perron root of $A_k$ and $v_k$ is the Perron vector of $A_k$. Then it holds that $(v_k)_{k=1}^\infty$ is bounded and therefore there exists a subsequence of $(v_k)_{k=1}^\infty$ that is convergent, i.e.

$$(v_k)_{i=1}^\infty \to v$$
where \( v \geq 0 \) and \( v \neq 0 \). It can also be observed that the condition:

\[
A_1 > A_2 > \cdots > A \geq 0
\]

imply that:

\[
r_1 \geq r_2 \geq \cdots \geq r
\]

i.e. \( \inf \{ r_k : k \in \mathbb{N} \} = r \) and the sequence \( (r_k)_{k=1}^{\infty} \) is bounded below by \( r \) and therefore the following limits exists:

\[
\lim_{k \to \infty} r_k = r^*
\]

where \( r^* \geq r \) and

\[
\lim_{i \to \infty} r_{k_i} = r^*
\]

Now, the construction of the sequence \( A_k = A + \frac{1}{k}E \) give that:

\[
\lim_{k \to \infty} A_k = A
\]

and hence:

\[
\lim_{i \to \infty} A_{k_i} = A
\]

Finally, when we know that all the considered sequences are convergent, we obtain:

\[
Av = \lim_{i \to \infty} A_{k_i}v_{k_i} = \lim_{i \to \infty} r_{k_i}v_{k_i} = r^*v
\]

which implies that: \( r^* \in \sigma(A) \) and \( r^* \leq r \), i.e. we have shown that \( r^* = r \) and \( v \geq 0 \), \( v \neq 0 \) are an eigenpair of \( A \).

\[\square\]

A more rigorous proof requires the use of the Bolzano–Weierstrass theorem, the monotone convergence theorem and the algebraic limit theorem.
Appendix B

Theorems and Proofs

Theorem B.1. If we let $G_A$ and $G_B$ be two graphs with real and non-negative adjacency matrices $A = A_{m \times m}$ and $B = B_{n \times n}$, and if $A$ is symmetric, then the similarity matrix $S$ is a rank one matrix.

Proof. We know that $A$ is real, symmetric and non-negative, and therefore the Perron root is a real eigenvalue of $A$ and the eigenvector corresponding to the Perron root $\rho(A)$ is real and non-negative. From the symmetry of $A$ and the spectral theorem we obtain: $A = Q\Lambda Q^T$ where $Q^TQ = I$ and $\Lambda$ is a diagonal matrix with the eigenvalues of $A$ on the diagonal. The similarity matrix equation is:

$$\rho S = BSA^T + B^TSA$$

and with $A = Q\Lambda Q^T$ we get:

$$\rho S = BSA^T + B^TSA = BSQQ^T + B^TSSQ^T = (B + B^T)SQQ^T.$$  

The matrix sum $B + B^T$ is real, non-negative and symmetric which implies that we can use the spectral theorem to diagonalize it as:

$$B + B^T = UDU^T,$$

where $U$ is an orthogonal matrix and $D$ a diagonal matrix with the eigenvalues of $B + B^T$ on the diagonal.

If we insert this into the similarity matrix equation above we get:

$$\rho S = (B + B^T)SQQ^T = UDU^TSSQ^T.$$  

Let $\tilde{S} = U^TSQ$ then the equation evolves to:

$$\rho \tilde{S} = D\tilde{S}A.$$  

If we assume that $\delta$ and $\lambda$ are the largest eigenvalues of $D$ and $\Lambda$ then the Perron root $\rho$ is the product:

$$\rho = \delta \lambda$$
which we obtain by letting
\[ \tilde{S} = e_\delta \cdot e_\lambda^T \]
where \( e_\delta \) and \( e_\lambda \) are the \( \delta \) and \( \lambda \) unit vectors respectively.

Now we can solve the equation by the equality:
\[ S = U\tilde{S}Q^T = Ue_\delta \cdot e_\lambda^T Q^T = u_\delta \cdot q_\lambda^T \]
where \( u_\delta \) and \( q_\lambda \) are the corresponding eigenvectors of \( U \) and \( Q \) respectively.

The solution of the similarity matrix equation is:
\[ \rho = \delta \lambda \]
and
\[ S = u_\delta \cdot q_\lambda^T. \]

We can see that the similarity matrix is obtained by an outer product which is rank one, i.e. \( \text{rank}(S) = 1. \)

\[ \text{Proposition B.1. If} \]
\[ \mathbb{R}^{(m+n) \times (m+n)} \ni A = \begin{pmatrix} 0 & B_{m \times n} \\ (B^T)_{n \times m} & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & A_1 \\ 0 & 0 & A_2 \\ A_1^T & A_2^T & 0 \end{pmatrix} \]
is the adjacency matrix of the parallel text graph and if \( n > m \) then the corresponding self-similarity matrix \( S \) is spanned by the solution matrices:
\[ S = \begin{pmatrix} S_{11} & S_{12} & S_{13} \\ S_{21} & S_{22} & S_{23} \\ S_{31} & S_{32} & S_{33} \end{pmatrix} \in \text{span} \begin{pmatrix} u_1^{(1)}(u_1^{(1)})^T & u_1^{(2)}(u_1^{(1)})^T & v_1^{(1)}(v_1^{(1)})^T \\ u_1^{(1)}(u_1^{(2)})^T & u_1^{(2)}(u_1^{(2)})^T & v_1^{(1)}(v_1^{(2)})^T \\ v_1^{(1)}(u_1^{(1)})^T & v_1^{(1)}(u_1^{(2)})^T & v_1^{(1)}(v_1^{(1)})^T \end{pmatrix}, \]
where
\[ u_1 = \begin{pmatrix} u_1^{(1)} \\ u_1^{(2)} \end{pmatrix} \]
and \( v_1 \) are the first singular vectors of \( U \) and \( V_1 \) respectively derived from the SVD:
\[ B = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} = \begin{pmatrix} U^{(1)} \\ U^{(2)} \end{pmatrix} \Sigma V_1. \]

\[ \text{Proof B.1. The similarity matrix of the undirected parallel graph is symmetric i.e we have to show that the the matrix \( S \) in the equation:} \]
\[ \rho S = ASA \]
has the matrix block structure stated above. We have:
\[ S = \begin{pmatrix} S_{11} & S_{12} & S_{13} \\ S_{21} & S_{22} & S_{23} \\ S_{31} & S_{32} & S_{33} \end{pmatrix} \]
and

\[ A = \begin{pmatrix} 0 & B \\ B^T & 0 \end{pmatrix} \]

where \( B = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \in \mathbb{R}^{m \times n} \). We also have \( n > m \) which implies that the thin SVD of \( B \) is:

\[ B = U (\Sigma \ 0) \begin{pmatrix} V_1^T \\ V_2^T \end{pmatrix} = U \Sigma V_1^T. \]

If we let \( Z \) be the linear orthogonal transformation:

\[
Z = \begin{pmatrix}
\frac{1}{\sqrt{2}} U & -\frac{1}{\sqrt{2}} U & 0 \\
\frac{1}{\sqrt{2}} V_1 & \frac{1}{\sqrt{2}} V_1 & V_2
\end{pmatrix}
\]

then it follows:

\[
\begin{pmatrix} 0 & B \\ B^T & 0 \end{pmatrix} Z = Z \begin{pmatrix} \Sigma & 0 & 0 \\ 0 & -\Sigma & 0 \\ 0 & 0 & 0 \end{pmatrix}
\]

and by applying this on (B.1) we get:

\[
\rho Z^T S Z = Z^T A S A Z = (AZ)^T S A Z =
\]

\[
= \begin{pmatrix} \Sigma & 0 & 0 \\ 0 & -\Sigma & 0 \\ 0 & 0 & 0 \end{pmatrix} Z^T S Z \begin{pmatrix} \Sigma & 0 & 0 \\ 0 & -\Sigma & 0 \\ 0 & 0 & 0 \end{pmatrix}
\]

Let \( S = Z^T S Z \) then the equation above is:

\[
\rho \begin{pmatrix} S_{11} & S_{12} & S_{13} \\ S_{21} & S_{22} & S_{23} \\ S_{31} & S_{32} & S_{33} \end{pmatrix} = \begin{pmatrix} \Sigma S_{11} \Sigma & -\Sigma S_{13} \Sigma & 0 \\ -\Sigma S_{21} \Sigma & \Sigma S_{22} \Sigma & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \text{(B.2)}
\]

The Perron root \( \rho \) is non-negative and we obtain non-trivial solutions if \( \rho > 0 \) which implies that:

\[
S_{12} = 0, \quad S_{21} = 0 \\
S_{13} = 0, \quad S_{31} = 0 \\
S_{23} = 0, \quad S_{32} = 0 \\
S_{33} = 0.
\]

By applying the vec operator on \( S \) we can solve the equation (B.2) and we obtain two systems of linear equations:

\[
(\Sigma \otimes \Sigma) \text{vec} (S_{11}) = \rho \text{vec}(S_{11}) \quad \text{(B.3)}
\]

\[
(\Sigma \otimes \Sigma) \text{vec} (S_{22}) = \rho \text{vec}(S_{22}) \quad \text{(B.4)}
\]
If the singular values of $\Sigma$ are $\sigma_1 > \sigma_2 \geq \sigma_3 \geq \ldots \geq 0$ then $\rho = \sigma_1^2$. The four possible solutions derived from (B.3) and (B.4) are then:

1. $\text{vec}(S_{11}) = e_{1}^{(11)} \in \mathbb{R}^{m}$, $\text{vec}(S_{22}) = 0^{(22)} \in \mathbb{R}^{n}$
2. $\text{vec}(S_{11}) = 0^{(11)} \in \mathbb{R}^{m}$, $\text{vec}(S_{22}) = e_{1}^{(22)} \in \mathbb{R}^{n}$
3. $\text{vec}(S_{11}) = \frac{1}{\sqrt{2}} e_{1}^{(11)} \in \mathbb{R}^{m}$, $\text{vec}(S_{22}) = \frac{1}{\sqrt{2}} e_{1}^{(22)} \in \mathbb{R}^{n}$
4. $\text{vec}(S_{11}) = \frac{1}{\sqrt{2}} e_{1}^{(11)} \in \mathbb{R}^{m}$, $\text{vec}(S_{22}) = -\frac{1}{\sqrt{2}} e_{1}^{(22)} \in \mathbb{R}^{n}$

where $0^{(11)}$ and $0^{(22)}$ are zero vectors and $e_{1}^{(11)}$ and $e_{1}^{(22)}$ are the unit vectors with dimensions $m \times 1$ and $n \times 1$ respectively. The corresponding solutions, when applying the inverse of the vec operator, are:

1. $S_{11} = e_{1}^{(11)} (e_{1}^{(11)})^T \in \mathbb{R}^{m \times m}$, $S_{22} = 0^{(22)} \in \mathbb{R}^{n \times n}$
2. $S_{11} = 0^{(11)} \in \mathbb{R}^{m \times m}$, $S_{22} = e_{1}^{(22)} (e_{1}^{(22)})^T \in \mathbb{R}^{n \times n}$
3. $S_{11} = \frac{1}{\sqrt{2}} e_{1}^{(11)} (e_{1}^{(11)})^T \in \mathbb{R}^{m \times m}$, $S_{22} = \frac{1}{\sqrt{2}} e_{1}^{(22)} (e_{1}^{(22)})^T \in \mathbb{R}^{n \times n}$
4. $S_{11} = \frac{1}{\sqrt{2}} e_{1}^{(11)} (e_{1}^{(11)})^T \in \mathbb{R}^{m \times m}$, $S_{22} = -\frac{1}{\sqrt{2}} e_{1}^{(22)} (e_{1}^{(22)})^T \in \mathbb{R}^{n \times n}$

The matrix $Z$ is orthogonal and therefore we can derive $S$ from:

$$S = Z^T SZ \Leftrightarrow S = ZSZ^T.$$ 

With the solutions (1), (2), (3) and (4) we can now compute the similarity matrix $S$. Solution (1):

$$S = ZSZ^T =$$

$$= \begin{pmatrix}
\frac{1}{\sqrt{2}} U & -\frac{1}{\sqrt{2}} U & 0 \\
\frac{1}{\sqrt{2}} V_1 & \frac{1}{\sqrt{2}} V_1 & V_2^T \\
\frac{1}{\sqrt{2}} V_1 e_{1}^{(11)} (e_{1}^{(11)})^T & 0 & 0 \\
\frac{1}{\sqrt{2}} V_1 e_{1}^{(11)} (e_{1}^{(11)})^T & 0 & 0
\end{pmatrix}
\begin{pmatrix}
e_{1}^{(11)} (e_{1}^{(11)})^T & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
\frac{1}{\sqrt{2}} U^T & \frac{1}{\sqrt{2}} V_1^T \\
\frac{1}{\sqrt{2}} U^T & \frac{1}{\sqrt{2}} V_1^T
\end{pmatrix} =$$

$$= \frac{1}{\sqrt{2}} \begin{pmatrix} u_1 u_1^T & u_1 v_1^T (v_1^T)^T \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \end{pmatrix} = \left\{ u_1 = \begin{pmatrix} u_1^{(1)} \\
u_1^{(2)} \\
u_1^{(3)} \\
u_1^{(4)} \\
u_1^{(5)} \end{pmatrix} \text{ and } v_1 = \begin{pmatrix} v_1^{(1)} \\
v_1^{(2)} \\
v_1^{(3)} \\
v_1^{(4)} \\
v_1^{(5)} \end{pmatrix} \right\} =$$

$$= \frac{1}{2} \begin{pmatrix} u_1^{(1)} (u_1^{(1)})^T & u_1^{(1)} (v_1^{(1)})^T & u_1^{(1)} (v_1^{(2)})^T & u_1^{(1)} (v_1^{(3)})^T \\
u_1^{(2)} (u_1^{(1)})^T & u_1^{(2)} (v_1^{(1)})^T & u_1^{(2)} (v_1^{(2)})^T & u_1^{(2)} (v_1^{(3)})^T \\
u_1^{(3)} (u_1^{(1)})^T & u_1^{(3)} (v_1^{(1)})^T & u_1^{(3)} (v_1^{(2)})^T & u_1^{(3)} (v_1^{(3)})^T \\
u_1^{(4)} (u_1^{(1)})^T & u_1^{(4)} (v_1^{(1)})^T & u_1^{(4)} (v_1^{(2)})^T & u_1^{(4)} (v_1^{(3)})^T \\
u_1^{(5)} (u_1^{(1)})^T & u_1^{(5)} (v_1^{(1)})^T & u_1^{(5)} (v_1^{(2)})^T & u_1^{(5)} (v_1^{(3)})^T \end{pmatrix}.$$
Solution (2) is:
\[ S = Z S Z^T = \]
\[
\begin{pmatrix}
\frac{1}{\sqrt{2}} U & -\frac{1}{\sqrt{2}} U & 0 \\
\frac{1}{\sqrt{2}} V_1 & \frac{1}{\sqrt{2}} V_1 & V_2
\end{pmatrix}
\begin{pmatrix}
e_1^{(22)}(e_1^{(22)})^T & 0 & 0 \\
0 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
\frac{1}{\sqrt{2}} U^T & \frac{1}{\sqrt{2}} V_1^T \\
\frac{1}{\sqrt{2}} V_2
\end{pmatrix} = 
\frac{1}{2}
\begin{pmatrix}
u_1^{(1)}(u_1^{(1)})^T & u_1^{(1)}(u_1^{(1)})^T & -u_1^{(1)}(v_1^{(1)})^T \\
u_1^{(2)}(u_1^{(2)})^T & u_1^{(2)}(u_1^{(2)})^T & -u_1^{(2)}(v_1^{(1)})^T \\
v_1^{(1)}(u_1^{(1)})^T & -v_1^{(1)}(u_1^{(2)})^T & v_1^{(1)}(v_1^{(1)})^T
\end{pmatrix}.
\]

Solution (3) is:
\[ S = Z S Z^T = 
\]
\[
\begin{pmatrix}
\frac{1}{\sqrt{2}} e_1^{(11)}(e_1^{(11)})^T & 0 \\
0 & \frac{1}{\sqrt{2}} e_1^{(22)}(e_1^{(22)})^T & 0 \\
0 & 0 & 0
\end{pmatrix} Z^T = 
\frac{1}{2\sqrt{2}}
\begin{pmatrix}
u_1^{(1)}(u_1^{(1)})^T & u_1^{(1)}(u_1^{(1)})^T & 0 \\
u_1^{(2)}(u_1^{(2)})^T & u_1^{(2)}(u_1^{(2)})^T & 0 \\
v_1^{(1)}(v_1^{(1)})^T & v_1^{(1)}(u_1^{(2)})^T & 0
\end{pmatrix}.
\]

Solution (4) is:
\[ S = Z S Z^T = 
\]
\[
\begin{pmatrix}
\frac{1}{\sqrt{2}} e_1^{(11)}(e_1^{(11)})^T & 0 \\
0 & \frac{1}{\sqrt{2}} e_1^{(22)}(e_1^{(22)})^T & 0 \\
0 & 0 & 0
\end{pmatrix} Z^T = 
\frac{1}{2\sqrt{2}}
\begin{pmatrix}
0 & 0 & u_1^{(1)}(v_1^{(1)})^T \\
0 & 0 & u_1^{(2)}(v_1^{(1)})^T \\
v_1^{(1)}(u_1^{(1)})^T & v_1^{(1)}(u_1^{(2)})^T & 0
\end{pmatrix}.
\]

We can see that the two last solutions (3) and (4) are linear combinations of the first two solutions (1) and (2), hence the self-similarity matrix S is spanned by the solutions (1) and (2), thus:

\[
S \in \text{span}\left\{
\begin{pmatrix}
u_1^{(1)}(u_1^{(1)})^T & u_1^{(1)}(u_1^{(2)})^T & u_1^{(1)}(v_1^{(1)})^T \\
u_1^{(2)}(u_1^{(1)})^T & u_1^{(2)}(u_1^{(2)})^T & u_1^{(2)}(v_1^{(1)})^T \\
v_1^{(1)}(u_1^{(1)})^T & v_1^{(1)}(u_1^{(2)})^T & v_1^{(1)}(v_1^{(1)})^T
\end{pmatrix},
\begin{pmatrix}
u_1^{(1)}(u_1^{(1)})^T & u_1^{(1)}(u_1^{(2)})^T & -u_1^{(1)}(v_1^{(1)})^T \\
u_1^{(2)}(u_1^{(1)})^T & u_1^{(2)}(u_1^{(2)})^T & -u_1^{(2)}(v_1^{(1)})^T \\
v_1^{(1)}(u_1^{(1)})^T & -v_1^{(1)}(u_1^{(2)})^T & v_1^{(1)}(v_1^{(1)})^T
\end{pmatrix}\right\}.
\]
One observation we can do from the proposition above is that if we add the two solution matrices we obtain the same solution that is given in the case when the parallel text graph is directed, i.e.

\[
S = \begin{pmatrix}
S_{11} & S_{12} & S_{13} \\
S_{21} & S_{22} & S_{23} \\
S_{31} & S_{32} & S_{33}
\end{pmatrix}
= \begin{pmatrix}
(u_1^{(1)})^T & (u_1^{(1)})^T & 0 \\
(u_2^{(1)})^T & (u_2^{(1)})^T & 0 \\
0 & 0 & v_1^{(1)}(v_1^{(1)})^T
\end{pmatrix}.
\]

If we for \( B \in \mathbb{R}^{m \times n} \) instead have \( m > n \) then the proposition can be proved in similar fashion, using the SVD of \( B^T \) such that:

\[
B^T = (A_1^T \ A_2^T) = V \Sigma U_1^T
\]

and the linear orthogonal transformation:

\[
\hat{Z} = \begin{pmatrix}
\frac{1}{\sqrt{2}} U_1^T & -\frac{1}{\sqrt{2}} V^T \\
\frac{1}{\sqrt{2}} U_1^T & \frac{1}{\sqrt{2}} V^T
\end{pmatrix}
\]

By multiplying the adjacency matrix:

\[
A = \begin{pmatrix}
0 & B \\
B^T & 0
\end{pmatrix}
\]

from the left we get:

\[
\hat{Z} \begin{pmatrix}
0 & B \\
B^T & 0
\end{pmatrix} = \begin{pmatrix}
\Sigma & 0 & 0 \\
0 & -\Sigma & 0 \\
0 & 0 & 0
\end{pmatrix} \hat{Z}
\]

and then the self-similarity matrix equation is:

\[
\rho \tilde{Z} S \tilde{Z}^T = \tilde{Z} A S \tilde{Z}^T =
\begin{pmatrix}
\Sigma & 0 & 0 \\
0 & -\Sigma & 0 \\
0 & 0 & 0
\end{pmatrix} \tilde{Z} S \tilde{Z}^T
\begin{pmatrix}
\Sigma & 0 & 0 \\
0 & -\Sigma & 0 \\
0 & 0 & 0
\end{pmatrix}.
\]

Letting \( \tilde{S} = \tilde{Z} S \tilde{Z}^T \) we get:

\[
\rho \tilde{S} = \begin{pmatrix}
\Sigma \tilde{S}_{11} \Sigma & -\Sigma \tilde{S}_{12} \Sigma & 0 \\
-\Sigma \tilde{S}_{21} \Sigma & \Sigma \tilde{S}_{22} \Sigma & 0 \\
0 & 0 & 0
\end{pmatrix}.
\]

From here we can use the same computations applied above to find the solutions.
of the equation. The self-similarity matrix $S$ is then spanned by:

$$S \in \text{span} \left\{ \begin{pmatrix} u^{(11)}_1 & (u^{(11)}_2) \end{pmatrix}^T, \begin{pmatrix} u^{(21)}_1 & (u^{(21)}_2) \end{pmatrix}^T, \begin{pmatrix} u^{(11)}_1 & v^T_1 \\ v_1(u^{(11)}_2) & v^T_1 v^T_1 \end{pmatrix}, \begin{pmatrix} u^{(11)}_1 & (u^{(11)}_2) \end{pmatrix}^T, \begin{pmatrix} u^{(21)}_1 & (u^{(21)}_2) \end{pmatrix}^T, \begin{pmatrix} u^{(21)}_1 & v^T_1 \\ v_1(u^{(21)}_2) & v^T_1 v^T_1 \end{pmatrix}, \begin{pmatrix} u^{(11)}_2 & (u^{(11)}_2) \end{pmatrix}^T, \begin{pmatrix} u^{(21)}_2 & (u^{(21)}_2) \end{pmatrix}^T, \begin{pmatrix} u^{(21)}_2 & v^T_1 \\ v_1(u^{(21)}_2) & v^T_1 v^T_1 \end{pmatrix} \right\}$$

where $u^{(1)}_1 = \begin{pmatrix} u^{(11)}_1 \\ u^{(12)}_1 \end{pmatrix}$ is the partition of the first left singular vector $u^{(1)}_1 \in U_1$ with respect to the matrices $A_1$ and $A_2$ in:

$$B = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}.$$
Appendix C

Details of the Preprocessing and the Text Parsing of the Text Collections

In this chapter a more detailed description of the preprocessing of the text collections is presented.

In the beginning the text collections were stored as a fdg-file, i.e. each line contained one word, the base form of the word and a semantic description of the word. The first English document or sentence was in our case:

1 Resumption resumption main:>0 @NH %NH N NOM SG
2 of of mod:>1 @<NOM-OF %N< PREP
3 the the attr:>4 @A> %>N DET
4 session session pcomp:>2 @<P %NH N NOM SG
</sent1>

All the sentences in this text collection ended with the tag </sent#, where # denote the number of the sentence. We were only interested in the base form of the sentences, i.e. we had to remove all unwanted words and tags and we also wanted to construct a more readable document. This was done by a MATLAB-script, which removed all the unwanted information and let each line of the new document contain at most ten words. In order to obtain a text collection that we could parse with GTP we replaced the delimiter </sent#> with an empty line. Using the MATLAB-script the first English document was:

resumption of the session

This procedure worked well for most of the sentences in the text collections, except for words containing special characters, e.g. é, à, ō, ä, ö. We had a problem with the text encoding for these characters, for example the letter ö where represented with the symbol ¶. The only way that worked in order to get the text encoding right, was to replace all these types of symbols with the corresponding letter. Once again we used a MATLAB-script, which replaced all these symbols with the right letter. In order to find the correspondence between a symbol and a special character, e.g. ¶ = ö, we had to do some detective work, which unfortunately took longer time than expected.

Törnfeldt, 2008. 77
Appendix C. Details of the Preprocessing and the Text Parsing of the Text Collections

After the preprocessing of the text collections, the text parser GTP was used in order to get the term-by-document matrices. We observed that the number of documents parsed in the English text collection was not the same as the number of documents parsed in the Swedish text collection, i.e. we could not align the term-by-documents matrices according to the suggested approach. The idea of adding a tag to each document, presented in chapter 4, was not obvious in the beginning when we discovered the problem. We used different MATLAB-scripts and approaches in order to generate an index of which documents GTP did not parse in these text collections, but without any success. This was a time-consuming step with all the trials and errors before we solved the problem, since parsing one of the English or Swedish text collections took about 45 minutes.
Appendix D

The Euclidean Vector Norm, the Frobenius Matrix Norm and the Trace Operator

When we use SVD to approximate the parallel texts we want to know, in some sense, how much the approximation differs from the original matrix. To measure the “size” of a vector or a matrix we have to introduce the norm of a vector and the norm of matrix.

**Definition D.1.** If $x, y \in \mathbb{R}^n$ then the vector norm, $\| \cdot \|$, is a mapping $\mathbb{R}^n \to \mathbb{R}$ that satisfies the following conditions:

- $\|x\| \geq 0$ and $\|x\| = 0$ if and only if $x = 0$,
- $\|\alpha x\| = |\alpha| \|x\|$, for all $\alpha \in \mathbb{R}$, \hspace{1cm} (D.1)
- $\|x + y\| \leq \|x\| + \|y\|$, the triangle inequality.

One of the most common vector norms is the Euclidean norm or 2-norm which is:

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} \hspace{1cm} (D.2)$$

This norm is related to the inner product of two vectors $x, y \in \mathbb{R}^n$ defined as:

$$(x, y) = x^T y$$

i.e. the inner product of $x$ is $(x, x) = x^T x = \|x\|_2^2$. If a vector $x$ has the 2-norm $\|x\|_2 = 1$ then $x$ is said to be orthonormal. Any vector can be normalized by dividing the vector with its 2-norm i.e. if $x$ is a vector it is normalized by:

$$x \leftarrow \frac{x}{\|x\|_2}.$$
A generalization of the Euclidean norm can be done for matrices, if $A$ is a matrix then by applying the Euclidean norm on $x = \text{vec}(A)$ we obtain the Frobenius matrix norm. A more formal definition is stated below.

**Definition D.2.** If $A \in \mathbb{R}^{m \times n}$ then the Frobenius norm of $A$ is:

$$
\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2}
$$

(D.3)

The Frobenius norm is related to the inner product of two matrices $A, B \in \mathbb{R}^{m \times n}$ defined as:

$$
\langle A, B \rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij} B_{ij}
$$

$\Rightarrow \langle A, A \rangle = \|A\|^2_F$.

**Definition D.3.** The trace, denoted $\text{tr}(\cdot)$, is the linear operator computing the sum of the diagonal elements in a square matrix $A$:

$$
\text{tr}(A) = \sum_{i} a_{ii}.
$$

**Proposition D.1.** If $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times m}$ are matrices, then the following properties are true:

i) $\text{tr}(AB) = \text{tr}(BA)$,

ii) $\text{tr}(AB) = \langle A, B \rangle$.

**Proof D.1.** i) The proof follows by the computation:

$$
\text{tr}(AB) = \sum_{i=1}^{m} (AB)_{ii} = \sum_{i=1}^{m} a_{i} b_{i} = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} b_{ji}
$$

$= \sum_{j=1}^{n} \sum_{i=1}^{m} b_{j} a_{ij} = \sum_{j=1}^{n} b_{j} a_{j} = \sum_{j=1}^{n} (BA)_{jj} = \text{tr}(BA).
$

ii) The proof follows by the computation:

$$
\text{tr}(AB) = \sum_{i=1}^{m} (AB)_{ii} = \sum_{i=1}^{m} a_{i} b_{i} = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} b_{ji} = \langle A, B \rangle.
$$

$\Box$
The property ii) in proposition D.1 implies, for any matrix $A$, that:

$$\|A\|_F^2 = \text{tr}(A^T A) = \text{tr}(AA^T)$$

If the matrix product, $ABC$, is well defined then the trace of the product is preserved if we perform any of the *cyclic permutations* $ABC$, $BCA$ or $CAB$ i.e:

$$\text{tr}(ABC) = \text{tr}(BCA) = \text{tr}(CAB),$$

since:

$$\begin{align*}
\text{tr}(ABC) &= \text{tr}((A)BC) = \text{tr}(BC(A)) = \\
&= \text{tr}(BCA) = \text{tr}(B(CA)) = \text{tr}((CA)B) = \\
&= \text{tr}(CAB),
\end{align*}$$

according to property i) of proposition D.1. This equality implies that if $A = U\Sigma V^T$ is the SVD of a matrix $A$ then the Frobenius norm is:

$$\|A\|_F^2 = \text{tr}(A^T A) = \text{tr}(V\Sigma^2 V^T) = \text{tr}(\Sigma^2 V^T V) = \text{tr}(\Sigma^2)$$

$$\Rightarrow \|A\|_F = \sqrt{\sum_{i=1}^{k} \sigma_i^2}, \quad k = \text{rank}(A).$$

The Frobenius norm is easy to compute and therefore it is practical to use in different applications. If a sparse data matrix is stored in the CCS format, the Frobenius norm of this matrix is the Euclidean norm of the $\text{val}$ array, since if $A = (a_{1}, a_{2}, \ldots, a_{n}) \in \mathbb{R}^{m \times n}$ then:

$$\|A\|_F = \sqrt{\sum_{i=1}^{n} \|a_{i}\|_2^2} = \|\text{vec}(A)\|_2 = \|\text{val}\|_2.$$
Appendix D. The Euclidean Vector Norm, the Frobenius Matrix Norm and the Trace Operator
Appendix E

Text Parsing Tools: TMG and GTP

E.1 TMG: Text to Matrix Generator

TMG [18] is a MATLAB toolbox developed by Dimitrios Zeimpekis and Efstratios Gallopoulos to generate term-by-document matrices from English text collections. This toolbox is easy to use and it is provided with a graphical user interface (GUI). The preprocessing steps, stemming and removal of stop words, is implemented as an option and the stemmer is a Porter stemmer.

There are several options that can be added \(^1\), for example the minimum local [global] frequency which determine the minimum local [global] term frequency to be used in a text document [collection]. If we for instance set the minimum local frequency to one and the minimum global frequency to two, a term is parsed if it appears in any text document at least once and in the whole text collection at least twice.

The advantage of TMG, is that it is user friendly and all the data (text collection, term-by-document matrix and dictionary) can be managed in MATLAB. TMG is perfect to use when handling small English text collections, but there is no support for special characters and if the text collection is large, as in our case, MATLAB will run out of memory.

E.2 GTP: General Text Parser

GTP [4, 23] was in the beginning a C/C++ program designed for text parsing. Now the program contains several features as e.g. query matching and matrix decompositions. There is also a Java version and a Windows XP compatible version available.

GTP works in general as TMG, we have a text collection and additional options as input and after the text parsing step we obtain a term-by-document matrix and a dictionary. The removal of stop word is implemented in GTP, but there is no stemming thus we have to stem the text collection before applying

\(^1\) Type help tmg in MATLAB for more details.
GTP. There are also filter routines implemented if the original text collection for instance contains HTML or \LaTeX tags it is possible to remove them.

**Example E.1.** Assume that we want to parse a text collection, containing international characters and we also want to use additional options. Then we type:

```
./gtp file_name -c common_words -t ./temp_dir -h -u -d 0 -g 1
-w tf idf -e "'special characters e.g. åöûä'" -o file
```

Here *file_name* is the text collection, *common_words* the stop list containing the stop words, *temp_dir* a directory where temporary variables and files are saved during the text parsing procedure. The options -h and -u tells the program to save the term-by-document matrix and not to compress it. By -d 0 and -g 1 a term has to appear at least once in any text document and at least twice in the whole text collection to be parsed. The weighting scheme used is set by -w tf idf which means that we are using term frequency as local weighting scheme and inverse document frequency as global weighting scheme. To handle special characters the option -e "'special characters'" tells the text parser not to delete these characters when parsing the text. The last option -o file saves the dictionary and term frequencies in a file.

The example above demonstrates the standard input and some of the options that one can use\(^2\). We can combine these options in many different ways and as can be read in the GTP guide “... GTP has grown into the sizable program (over 13,000 lines of code) ... Even expert users have a difficult time keeping all the options straight.” it is hard to sort out which option to use when.

\(^2\)Type ./gtp -help or consult the manual for more details.
Appendix F

A Random Sample from the Bilingual Lexicon

Here is a random sample of the bilingual lexicon. We have derived the bilingual lexicon by using the VSM method. The first column indicates the scores used to compute the estimated precision $\hat{S}_{pre}$, the second column contains the self-similarity score extracted from the self-similarity matrix $S_{12}$, and the two last columns are the suggested translations.

| 1 | 0.95043 barn | child |
| 1 | 0.94476 vatten | water |
| 1 | 0.94342 patent | patent |
| 1 | 0.94124 falconer | falconer |
| 1 | 0.93608 rödsla | fear |
| 1 | 0.93375 lax | salmon |
| 1 | 0.93287 vin | wine |
| 1 | 0.93081 lantbrukare | farmer |
| 1 | 0.92999 argument | argument |
| 1 | 0.92987 kristen | christian |
| 1 | 0.92831 sommar | summer |
| 1 | 0.9215 framtid | future |
| 1 | 0.91937 augusi | august |
| 1 | 0.91899 poettering | poettering |
| 1 | 0.9158 kuba | cuba |
| 1 | 0.91169 eftergift | concession |
| 1 | 0.91048 ton | tonne |
| 1 | 0.90838 meter | metre |
| 1 | 0.90783 palestinier | palestinian |
| 1 | 0.90688 ell | egg |
| 1 | 0.90408 personlig | personal |
| 1 | 0.90368 olaf | olaf |
| 1 | 0.90178 årtionde | decade |
| 1 | 0.90131 konferens | conference |
| 1 | 0.90044 protektionism | protectionism |
| 1 | 0.90001 nicaragua | nicaragua |
| 1 | 0.89767 genombrott | breakthrough |
| 1 | 0.89746 cancer | cancer |
| 1 | 0.89717 intervju | interview |
| 1 | 0.89688 individuell | individual |
| 1 | 0.89626 brasilen | brazil |
| 1 | 0.89436 perfekt | perfect |
| 1 | 0.89244 kaliningrad | kaliningrad |
| 1 | 0.89223 burna | burna |
| 1 | 0.89147 korruption | corruption |
| 1 | 0.88972 beslut | decision |
| 1 | 0.88831 produktion | production |
| 1 | 0.88812 dupuis | dupuis |

Törnfeldt, 2008.
86 Appendix F. A Random Sample from the Bilingual Lexicon

1 0.88802 byråkratisk bureaucratic
1 0.88724 skandal scandal
1 0.8869 februari february
1 0.88672 lag law
1 0.88637 kapitalism capitalism
1 0.8862 farlig dangerous
1 0.88418 måndag monday
1 0.88361 kontroversiell controversial
1 0.88311 steg step
1 0.88235 dödsfall death
1 0.88216 haarder haarder
1 0.88078 resolution resolution
1 0.88055 eib eib
1 0.87773 berlin berlin
1 0.87643 fenomen phenomenon
0.8 0.87572 förhandel negotiator
1 0.87445 gömna hide
1 0.87442 solana solana
1 0.87359 beständig stock
1 0.87303 lärna learn
1 0.87027 däck tyre
1 0.87022 period period
1 0.86967 acceptera accept
1 0.86943 svårighet difficulty
1 0.86875 överskott surplus
1 0.86804 visum visa
1 0.86279 teknisk technical
1 0.86186 multinationell multinational
1 0.86153 respekttera respect
1 0.86099 förbättra improve
1 0.86077 federation federation
1 0.86037 reservation reservation
1 0.85922 modell model
1 0.85691 osnödig unnecessary
1 0.85612 analys analysis
1 0.85393 ungerrsk hungarian
1 0.85356 lannoys lannoys
1 0.85263 slaveri slavery
1 0.85212 undra wonder
1 0.85113 funktionshindrad disabled
1 0.84844 resultat result
1 0.84825 civilisation civilisation
1 0.84789 ansöktning application
1 0.84751 tillväxt growth
1 0.84643 klar clear
1 0.84614 serb serb
1 0.84546 slutsats conclusion
1 0.84295 eta eta
1 0.84227 dogm dogma
1 0.84188 utpressning blackmail
1 0.84138 konsument consumption
1 0.84054 parlamentarisk parliamentary
1 0.83883 administration administration
1 0.83848 ledarskap leadership
1 0.83586 metod method
1 0.83582 berömd famous
1 0.83533 vattenbruk aquaculture
1 0.83314 produktivitet productivity
1 0.83098 radikal radical
1 0.83098 potentiellt potentially
1 0.82955 flyga fly
1 0.82803 enkel simple
1 0.82569 horisont horizon
1 0.82569 kaukasus caucasus
1 0.82543 liikanen liikanen
1 0.8231 hota threaten
1 0.82293 paramilitär paramilitary
1 0.82281 tolka interpret
1 0.82278 totalitär totalitarian
1 0.82179 oreja oreja
1 0.82032 artikel article
1 0.81971 papoutsis papoutsis
1 0.81729 giftig toxic
1 0.81726 diskriminerande discriminatory
Appendix F. A Random Sample from the Bilingual Lexicon

<table>
<thead>
<tr>
<th>Swedish</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td>återförening</td>
<td>reunification</td>
</tr>
<tr>
<td>socialt</td>
<td>socially</td>
</tr>
<tr>
<td>fiskefartyg</td>
<td>vessel</td>
</tr>
<tr>
<td>tillfredsställa</td>
<td>satisfy</td>
</tr>
<tr>
<td>språklig</td>
<td>linguistic</td>
</tr>
<tr>
<td>schöring</td>
<td>schöring</td>
</tr>
<tr>
<td>lucka</td>
<td>gap</td>
</tr>
<tr>
<td>orimlig</td>
<td>unreasonable</td>
</tr>
<tr>
<td>trafik</td>
<td>traffic</td>
</tr>
<tr>
<td>harmonisera</td>
<td>harmonise</td>
</tr>
<tr>
<td>aria</td>
<td>aria</td>
</tr>
<tr>
<td>fejs</td>
<td>fifteen</td>
</tr>
<tr>
<td>intensiv</td>
<td>intensive</td>
</tr>
<tr>
<td>manöverutrymme</td>
<td>manoeuvre</td>
</tr>
<tr>
<td>utbredd</td>
<td>widespread</td>
</tr>
<tr>
<td>landskap</td>
<td>landscape</td>
</tr>
<tr>
<td>falla</td>
<td>fall</td>
</tr>
<tr>
<td>bryta</td>
<td>break</td>
</tr>
<tr>
<td>subventionera</td>
<td>subsidise</td>
</tr>
<tr>
<td>olba</td>
<td>olba</td>
</tr>
<tr>
<td>åmma</td>
<td>intend</td>
</tr>
<tr>
<td>fraga</td>
<td>fraga</td>
</tr>
<tr>
<td>weiler</td>
<td>weiler</td>
</tr>
<tr>
<td>bissau</td>
<td>bissau</td>
</tr>
<tr>
<td>effekttivt</td>
<td>effectively</td>
</tr>
<tr>
<td>påverka</td>
<td>affect</td>
</tr>
<tr>
<td>erkänna</td>
<td>recognise</td>
</tr>
<tr>
<td>smet</td>
<td>smet</td>
</tr>
<tr>
<td>längtgående</td>
<td>reaching</td>
</tr>
<tr>
<td>kontrollerad</td>
<td>controlled</td>
</tr>
<tr>
<td>elev</td>
<td>pupil</td>
</tr>
<tr>
<td>flytta</td>
<td>move</td>
</tr>
<tr>
<td>cotonouavtal</td>
<td>cotonou</td>
</tr>
<tr>
<td>loxand</td>
<td>promising</td>
</tr>
<tr>
<td>underlätta</td>
<td>facilitate</td>
</tr>
<tr>
<td>avfolkning</td>
<td>depopulation</td>
</tr>
<tr>
<td>marsret</td>
<td>marsret</td>
</tr>
<tr>
<td>utvidgad</td>
<td>enlarged</td>
</tr>
<tr>
<td>början</td>
<td>beginning</td>
</tr>
<tr>
<td>bekymrad</td>
<td>concerned</td>
</tr>
<tr>
<td>vana</td>
<td>habit</td>
</tr>
<tr>
<td>fördrev</td>
<td>displaced</td>
</tr>
<tr>
<td>stillracklighet</td>
<td>inadequacy</td>
</tr>
<tr>
<td>omtvissla</td>
<td>injustice</td>
</tr>
<tr>
<td>affärsmann</td>
<td>businessman</td>
</tr>
<tr>
<td>utrotning</td>
<td>eradication</td>
</tr>
<tr>
<td>efterålde</td>
<td>lag</td>
</tr>
<tr>
<td>solid</td>
<td>solid</td>
</tr>
<tr>
<td>deklarera</td>
<td>declare</td>
</tr>
<tr>
<td>botten</td>
<td>basically</td>
</tr>
<tr>
<td>oomen</td>
<td>oomen</td>
</tr>
<tr>
<td>graefe</td>
<td>graefe</td>
</tr>
<tr>
<td>förknippa</td>
<td>associate</td>
</tr>
<tr>
<td>fördrev</td>
<td>person</td>
</tr>
<tr>
<td>undantagsbestämmelse</td>
<td>derogation</td>
</tr>
<tr>
<td>kombinerad</td>
<td>combined</td>
</tr>
<tr>
<td>anöverutrymme</td>
<td>room</td>
</tr>
<tr>
<td>extra</td>
<td>extra</td>
</tr>
<tr>
<td>basel</td>
<td>basle</td>
</tr>
<tr>
<td>variation</td>
<td>variation</td>
</tr>
<tr>
<td>sek</td>
<td>sec</td>
</tr>
<tr>
<td>ankomst</td>
<td>slot</td>
</tr>
<tr>
<td>osäker</td>
<td>uncertain</td>
</tr>
<tr>
<td>sedgivande</td>
<td>consent</td>
</tr>
<tr>
<td>oförmögen</td>
<td>incapable</td>
</tr>
<tr>
<td>anordning</td>
<td>device</td>
</tr>
<tr>
<td>oförmögen</td>
<td>unable</td>
</tr>
<tr>
<td>lön</td>
<td>salary</td>
</tr>
<tr>
<td>meningsfull</td>
<td>meaningful</td>
</tr>
<tr>
<td>dubbel</td>
<td>dual</td>
</tr>
<tr>
<td>stimulerande</td>
<td>stimulating</td>
</tr>
<tr>
<td>enhet</td>
<td>unit</td>
</tr>
<tr>
<td>småsklig</td>
<td>inhumane</td>
</tr>
<tr>
<td>impuls</td>
<td>impetus</td>
</tr>
</tbody>
</table>
invända await
rubrik heading
orätt wrongly
förestående forthcoming
olost unresolved
vacker beautiful
funktion function
upprättig sincere
uttryckligt explicit
overträdelseförfarande proceeding
övervakningscentrum observatory
ini ini
ompröva reconsider
sjöfart shipping
återtagande readmission
taget practically
plocka pick
förmodad supposed
berg berg
begriplig comprehensible
förmögenhet fortune
orando worry
ansluta join
europaparlamentariker mep
fram frame
erkänd recognised
förmoda suppose
drama drama
deruhpör ceasefire
upphävning dwell
emergence
östutvidgning eastward
bostäding housing
ale ale
omständigt definitely
utsätta expose
ring ring
maskin machinery
kommande coming
fullkomligt totally
ofarlig hazardous
i? i?
önsker unsafe
verkse operare
finansiär funded
reklam advertise
bekant familiar
skick stand
kontinuerligt continuously
arbetskraft workforce
brist shortage
hund client
omständigt mentioned
knappt scarcely
uppfatta perceive
räta rectify
försvarbar justifiable
snabbt rapidly
nedan whilst
döe die
inledande introductory
likväl nonetheless
resa journey
uteslutande solely
bokpris fixed
förstärkt strengthened
överensstämma consistent
bevarande preservation
personalpolitik staffing
uppskjuta defer
undervisning teaching
förgrund fore
slutgiltigt definitively
ansluta endorse
The estimated precision of this sample is $\hat{S}_{pre} = 0.87$, and the 95% confidence interval is:

$$I_{pre} = (0.84, 0.90),$$

i.e. $S_{pre} = 0.87 \pm 0.03$. If we set all the values that are 0.5 to 1.0 the precision $S_{pre}$ becomes $S_{pre} = 0.92 \pm 0.03$, when using a 95% confidence interval. If we set all the values that are 0.5 to 0.0 the precision $S_{pre}$ becomes $S_{pre} = 0.82 \pm 0.04$, when using a 95% confidence interval.
This result shows that we might have been a little too hard when using the scoring system described in subsection 3.7.1. We almost get a precision of about 90%, which can be considered as a very good result.
Bibliography


[28] Chih-Jen Lin. Projected gradient methods for non-negative matrix factorization. Department of Computer Science National Taiwan University, Taipei 106, Taiwan.


Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years from the date of publication barring exceptional circumstances. The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

Upphovsrätt

Detta dokument hälls tillgängligt på Internet - eller dess framtidens ersättare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår. Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art. Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådan sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart. För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

© 2008, Tobias Törnfeldt