Examensarbete

Computing Word Senses by Semantic Mirroring and Spectral Graph Partitioning

Martin Fagerlund

LiTH - MAT - EX - - 2010 / 11 - - SE

Computing Word Senses by Semantic Mirroring and Spectral Graph Partitioning

Department of Mathematics, Linköpings Universitet

Martin Fagerlund

LiTH - MAT - EX - - 2010 / 11 - - SE

Examensarbete: **30 hp**

Level: **D**

Supervisor: **Lars Eldén**,
Department of Mathematics, Linköpings Universitet

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

Examiner: **Lars Eldén**,
Department of Mathematics, Linköpings Universitet

Linköping: **june 2010**

LINKÖPINGS UNIVERSITET

Avdelning, Institution
Division, Department

Matematiska Institutionen
581 83 LINKÖPING
SWEDEN

Datum
Date

june 2010

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English
☐ _____

Rapporttyp
Report category

☐ Licentiatavhandling
☒ Examensarbete
☐ C-uppsats
☐ D-uppsats
☐ Övrig rapport
☐ _____

ISBN

ISRN
LiTH - MAT - EX - - 2010 / 11 - - SE

Serietitel och serienummer **ISSN**
Title of series, numbering

URL för elektronisk version

http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-57103

Titel
Title

Computing Word Senses by Semantic Mirroring and Spectral Graph Partitioning

Författare
Author

Martin Fagerlund

Sammanfattning
Abstract

In this thesis we use the method of Semantic Mirrors to create a graph of words that are semantically related to a seed word. Spectral graph partitioning methods are then used to partition the graph into subgraphs, and thus dividing the words into different word senses. These methods are applied to a bilingual lexicon of English and Swedish adjectives. A panel of human evaluators have looked at a few examples, and evaluated consistency within the derived senses and synonymy with the seed word.

Nyckelord
Keyword

Cheeger inequality, Fiedler value, Fiedler vector, Graph partitioning, Normalised Laplacian, Semantic mirror, Word senses.

Abstract

In this thesis we use the method of Semantic Mirrors to create a graph of words that are semantically related to a seed word. Spectral graph partitioning methods are then used to partition the graph into subgraphs, and thus dividing the words into different word senses. These methods are applied to a bilingual lexicon of English and Swedish adjectives. A panel of human evaluators have looked at a few examples, and evaluated consistency within the derived senses and synonymy with the seed word.

Keywords: Cheeger inequality, Fiedler value, Fiedler vector, Graph partitioning, Normalised Laplacian, Semantic mirror, Word senses.

Acknowledgements

I would like to thank my supervisor professor Lars Eldén for his commitment throughout this project. His support have been invaluable.
I would also like to thank my co-supervisors, professors Magnus Merkel and Lars Ahrenberg. Their knowledge in Linguistics have been of great importance.
My opponent Ronny Hedell deserves a great thank for reading the report and giving suggestions of improvement.
I would like to thank my family for their support, and for always believing in me.
Finally, a great thank to all of my friends.

Contents

Chapter 1

Introduction

1.1 Background

A great deal of linguistic knowledge is encoded implicitly in bilingual resources such as parallel texts and bilingual dictionaries. Dyvik [6, 8] has provided a knowledge discovery method based on the semantic relationship between words in a source language and words in a target language, as manifested in parallel texts. His method is called Semantic mirrors and the approach utilizes the way that different languages encode lexical meaning by mirroring source words and target words back and forth, in order to establish semantic relations like synonymy and hyponymy. Work in this area is strongly related to work within Word Sense Disambiguation (WSD) and the observation that translations are a good source for detecting such distinctions [5, 13, 17]. A word that has multiple meanings in one language is likely to have different translations in other languages. This means that translations serve as sense indicators for a particular source word, and make it possible to divide a given word into different senses.

The aim of this thesis work is to use spectral graph theory in connection with the method Semantic mirrors to group words into separate word senses.

This is done in the following way. The method Semantic mirrors can be interpreted as an algorithm for deriving a graph from a seed word, using translations back and forth between a source language and a target language. Every vertex of the graph is assigned a word that is related to the seed word. The graph obtained is partitioned using well-proven spectral graph partitioning methods. Then it follows that the words assigned to the vertices is grouped into separate word senses. Certain measures is used to decide if it is a satisfying partition.

1.2 Outline of the report

Chapter 2: In chapter 2 we give an detailed explanation of the method Semantic mirrors. Some linguistic assumptions are made, and an example is given to illustrate.

Chapter 3: In this chapter we present all the theory that we will need concerning graphs. We will describe different graphs and their properties, such as directed graphs, connected graphs, subgraphs and weighted graphs. Op-

erations that can be made on graphs are presented, e.g. edge deletion and vertex splitting. The spectral theory of graphs is presented, starting with the different matrices associated with graphs. From there we continue and describe the partitioning methods that later will be used.

Chapter 4: Chapter 4 describes how the computation of word senses are made practically. We start by describing how translations are made, and then continue by in detail describing how the implementation of the method Semantic mirrors are made. From there we derive an adjacency matrix of a graph, and then use the partitioning methods described in chapter 3 to make cuts in the graph. After we have partitioned the graph, the words belonging to the vertices of the graph are divided into sense groups. In the end of the chapter, some examples are presented.

Chapter 5: In chapter 5 we present the results from the evaluation that have been made of the method. Some discussions are made of the questions that have been observed during the work with this thesis.

Chapter 2

Semantic Mirrors

Helge Dyviks method Semantic Mirrors [7, 9] is partly about finding different meanings of a word, by using translations into a different language of that word. Given a lexicon, there is a lot of implicitly given information, that is hidden in the lexicon. By using this partially hidden information given in the lexicon, we can extract a intricate network of translational correspondences, bringing together the vocabularies of the two languages. This makes it possible for us to treat each language as the "semantic mirror" of the other language, in a way that will be described further on. In order for us to proceed, we need some assumptions [7]:

(1) Semantically closely related words tend to have strongly overlapping sets of translations.

(2) Words with a wide meaning tend to have a higher number of translations, then words with a more narrow meaning.

(3) Contrastive ambiguity, i.e. ambiguity[1] between two unrelated senses of a word, tend to be a historically accidental and idiosyncratic property. Therefore, we dont expect to find instances of the same contrastive ambiguity in other words, or by words in other languages.

(4) Words with unrelated meanings will not share translations into another language, except in cases where the shared words is contrastively ambiguous between the two unrelated meanings. By assumption **(3)** there should then be at most one such shared word.

By these assumptions, we should be able to distinguish contrastive ambiguity in the information given by the lexicon, and thereby find the senses of the original word.

In the explanation of the method Semantic Mirrors, we will use the Swedish word *rätt* as an example, and thereby hopefully make it a bit easier to understand. (Notice that the example is constructed to make it easy to understand, and does not reflect any dictionary accurately). So we start with *rätt* and look up the translations in the English language, as shown in figure 2.1.

[1] A word is called ambiguous if it can be interpreted in more than one way.

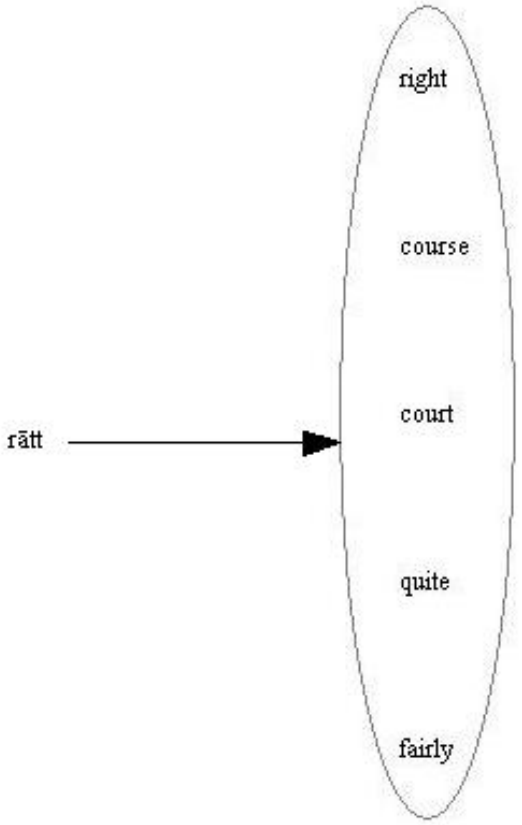


Figure 2.1: The first t-image of *rätt*.

This first set of words is refered to as the first t-image[2] of *rätt*. If we now take every word in the t-image of *rätt*, and look up its Swedish translations, we get the inverse t-image of *rätt*, shown in figure 2.2.

Every word is of course also translated back into our original word *rätt*, but this is excluded to make the picture easier to understand. We see that the translations of *quite* and *fairly* overlap by the word *ganska* (in addition to *rätt*), while there is no overlap from the words *right*, *course* and *court*. We therefore assume that *quite* and *fairly* are semantically related, while *right*, *course* and *court* are not. We then make a third move, and look up the first t-images of every word in the inverse t-image of *rätt*. This new set is refered to as the second t-image of *rätt*, and is shown, together with the words from the first t-image, in Figure 2.3.

As we can see, the second t-image can be divided into four groups of sets. The sets are being held together by overlap relations among the words in the first t-image. On the basis of these four groups, we can divide the first t-image of *rätt* into four sense partitions. This is shown in Figure 2.4. The conclusion

[2]Short for translational image.

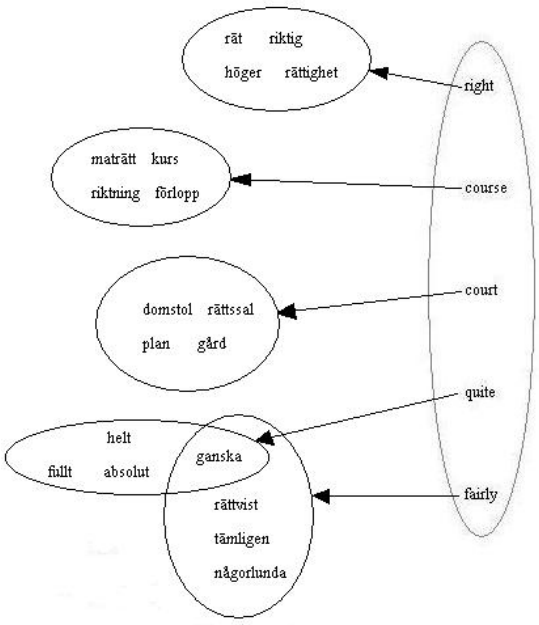


Figure 2.2: The inverse t-image of *rätt*.

is that that the senses of *rätt* represent unrelated meanings, and that *rätt* is ambiguous. This is how the main senses of lemmas are individuated.

Dyvik then derive semantic fields, give the words different features and derive hyperonym/hyponym relations among the words. A hyperonym is a superordinate concept, while a hyponym is a subordinate concept. An example is the relation between the words *human* and *child*. *Human* is a hyperonym, and *child* is a hyponym. A *child* is always a *human*, but a human does not have to be a *child*. However, these last operations is only a paranthesis for this thesis, and is not investigated further.

One can look at Dyvik's method Semantic mirrors as a way of creating a graph. Let the words be the vertices of the graph, and let an edge between two words be created if there is a translation shared by the two words. We will use this approach in the remainder of this thesis.

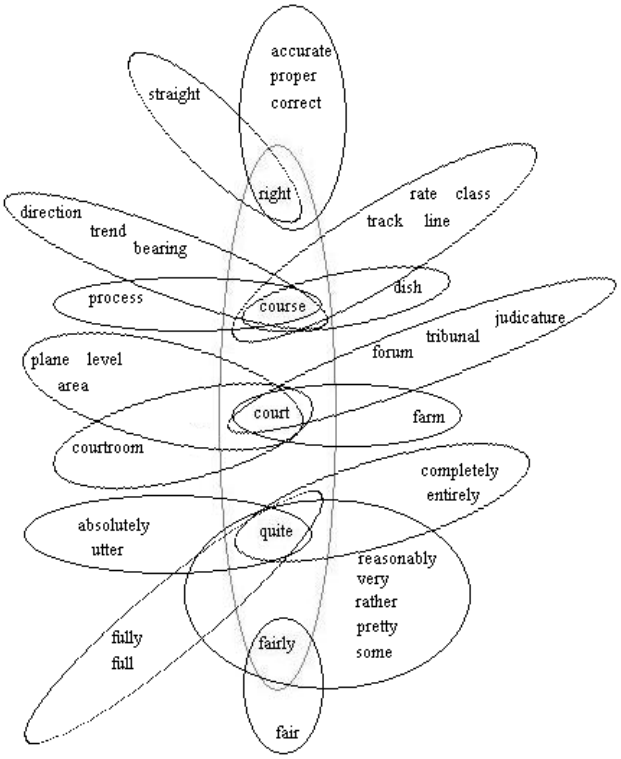


Figure 2.3: The second t-image of *rätt*.

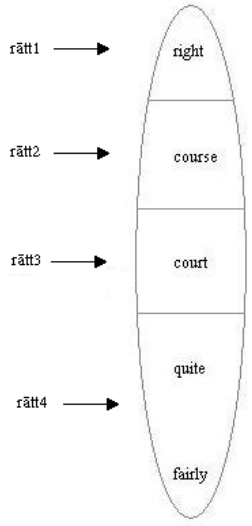


Figure 2.4: The sense partitions of *rätt*.

Chapter 3

Graph- and Spectral Theory

In this chapter, the necessary theory of graphs will be explained. In connection with graphs, some spectral theory is described, along with graph partitioning and connectivity. For further details, see [1, 3, 11].

3.1 Graph Theory

A graph $G = (V(G), E(G))$ is defined by a set $V(G)$ of vertices v_i, and a set $E(G)$ of edges e_j. An edge e_j may also be written $e(v_i, v_j)$ or even (v_i, v_j), where the vertices v_i and v_j are the ends of the edge. In that case, we say that v_i and v_j are adjacent. Sometimes, for simplicity, we write the vertices as i instead of v_i. In that case, an edge may be written as (i, j). We will also allow ourselves to write V instead of $V(G)$, and E instead of $E(G)$.

The vertices are denoted by a dot or a ring. The edges are denoted by a line. A directed graph is a graph in which every edge has a direction assigned. Such an edge is denoted by an arrow pointing in the given direction.

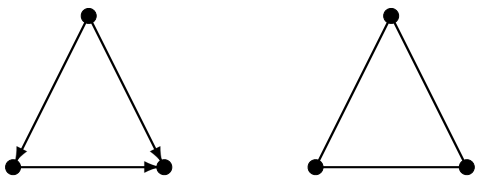


Figure 3.1: Directed (left) and undirected graph.

We will mostly work with undirected graphs. We will also allow loops in the graphs, because they play an important role. A loop is an edge with identical

ends, i.e. an edge that start and end at the same vertex. See Figure 3.2 for an illustration of a loop.

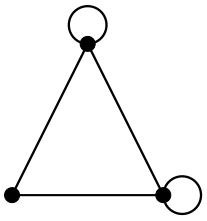


Figure 3.2: Graph with loops.

3.1.1 Connected and disconnected graph

Here we shall describe the necessary properties for a graph to be connected or disconnected. For a connected graph, we can also measure how well connected the graph is. This is done in chapter 3.2.2. First we need the definition of a walk. A *walk* in a graph G is an alternating sequence

$$v_0 e_1 v_1 \dots v_{l-1} e_l v_l \tag{3.1}$$

of vertices and edges, such that v_{i-1} and v_i are the ends of $e_i, 1 \leq i \leq l$. If $v_0 = x$ and $v_l = y$, we say that the walk (3.1) *connects* x to y. We are now ready for the definitions of a connected graph. We say that a graph G is *connected* if for every pair of vertices $\{x, y\} \in G$, there exists a walk that connects x to y. Then we also say that G consists of one *connected component*. If there exists a pair of vertices $\{x, y\} \in G$ with no such walk, then we say that G is *disconnected*. In that case, G consists of more than one connected component. In Figure 3.3 below we see a connected graph G, and a disconnected graph H. The graph H consists of three connected components, while G only consists of one connected component.

3.1.2 Subgraphs

Later on we are going to cut graphs into smaller graphs. Therefore we define subgraphs:

A graph H is a *subgraph* of a graph G if

$$V(H) \subseteq V(G), E(H) \subseteq E(G).$$

In Figure 3.3, the graph H is a subgraph of G. Any subgraph can be obtained by repeatedly applying the following two operations.

Figure 3.3: Connected graph (G), and disconnected graph (H).

- **Edge deletion**: Let G be a graph with m edges. Then, if $e \in E$, we may obtain a graph on $m-1$ edges by deleting e from G but leaving the vertices and the remaining edges intact. The resulting subgraph is denoted by $G\backslash e$.

- **Vertex deletion**: Let G be a graph with n vertices. Then, if $v \in V$, we may obtain a graph on $n-1$ vertices by deleting from G the vertex v together with all the edges incident with v. The resulting subgraph is denoted by $G - v$.

An example of the two operations is shown in Figure 3.4.

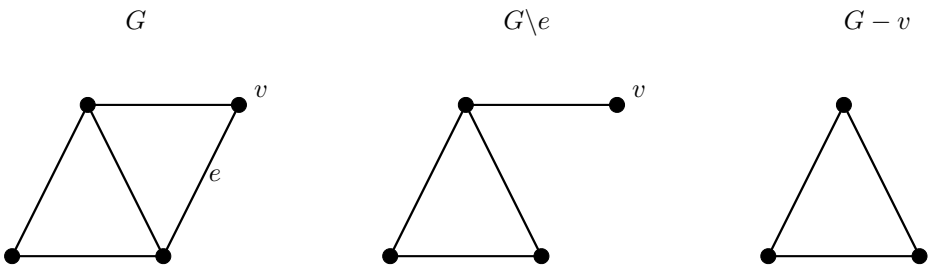


Figure 3.4: G with some subgraphs.

Another important operation that is possible to perform on a graph is Vertex splitting [3].

- **Vertex splitting**: To split a vertex $v \in V$, replace v by two adjacent vertices v' and v''. Let every edge incident with v instead be incident to either v' or v'', and let the other end of every such an edge remain unchanged. Notice that a vertex v with more than one edge incident to v, can be split in several ways, so the graph obtained is not unique.

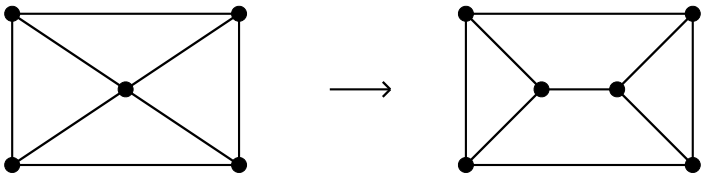


Figure 3.5: Vertex splitting

Vertex splitting is illustrated in Figure 3.5.
The next step is to give the graph some properties. We do this by assigning a weight function $w : V \times V \to \mathbb{R}$, to the edges and vertices. The weight for an edge between the vertices v_i and v_j is denoted by $w(v_i, v_j)$. The weight for a vertex v_i is denoted $w(v_i, v_i)$. The function satisfies

$$w(v_i, v_j) = w(v_j, v_i), \tag{3.2}$$

and

$$w(v_i, v_j) \geq 0. \tag{3.3}$$

It follows that if the edge $(v_i, v_j) \notin \mathrm{E}$, then $w(v_i, v_j) = 0$. The weight on a vertex can also be interpreted as the vertex having a loop, and the loop having the weight. However, this won't make any difference in the following derivations, and therefore we say that the weight is on the vertex. A graph with weights is illustrated in Figure 3.6.

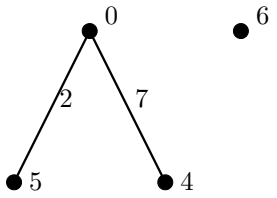


Figure 3.6: A graph with weights.

3.1.3 Graphs and matrices

In connection with graphs we can associate some matrices. The first one, the adjacency matrix A, is defined as:

$$A(i,j) = \begin{cases} w(v_i, v_j), & \text{if } v_i \text{ and } v_j \text{ are adjacent,} \\ 0, & \text{otherwise.} \end{cases}$$

The elements a_{ii} on the diagonal show the weight of the vertex i, and the elements $a_{ij}, i \neq j$ outside the diagonal show the weight on edge (v_i, v_j). In (3.4) we see the adjacency matrix to the graph in Figure 3.6.

$$\begin{pmatrix} 0 & 2 & 7 & 0 \\ 2 & 5 & 0 & 0 \\ 7 & 0 & 4 & 0 \\ 0 & 0 & 0 & 6 \end{pmatrix} \tag{3.4}$$

Observe that $a_{ij} = a_{ji}$, i.e. that A is symmetric. The unweighted case is treated in a similar way, with all the edges having weight $w = 1$ and the vertices having weight $w = 0$.

The next matrix we can associate with a graph is the Laplacian. To do this, we first need some definitions. The degree $d(v_i)$ of a vertex v_i is defined as:

$$d(v_i) = \sum_{v_j} w(v_i, v_j). \tag{3.5}$$

This is also equal to the i:th row sum of the adjacency matrix, that is,

$$d(v_i) = \sum_{j} A(i,j).$$

Using the degree we define the diagonal matrix D to be:

$$D(i,j) = \begin{cases} d(v_i), & \text{if } i = j, \\ 0, & \text{otherwise.} \end{cases}$$

Below in (3.6) is an example of the matrix D belonging to the graph in Figure 3.6.

$$\begin{pmatrix} 9 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 \\ 0 & 0 & 11 & 0 \\ 0 & 0 & 0 & 6 \end{pmatrix} \tag{3.6}$$

We can now define the Laplacian L of a graph to be:

$$L = D - A. \tag{3.7}$$

In other words, we have

$$L(v_i, v_j) = \begin{cases} d(v_i) - w(v_i, v_i), & \text{if } i = j, \\ -w(v_i, v_j), & \text{if } v_i \text{ and } v_j \text{ are adjacent,} \\ 0, & \text{otherwise.} \end{cases}$$

The Laplacian L of the graph in Figure 3.6 is seen in (3.8).

$$\begin{pmatrix} 9 & -2 & -7 & 0 \\ -2 & 2 & 0 & 0 \\ -7 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{3.8}$$

The last matrix that we want to associate with a graph is closely related to the Laplacian L. It is the normalised Laplacian, that we denote $\mathcal{L}$.

$$\mathcal{L} = D^{-1/2} L D^{-1/2}. \tag{3.9}$$

We use the convention that $D^{-1}(v_i, v_i) = 0$ if $d(v_i) = 0$. In other words

$$\mathcal{L} = \begin{cases} 1 - \frac{w(v_i,v_i)}{d(v_i)}, & \text{if } i = j \text{ and } d(v_i) \neq 0, \\ -\frac{w(v_i,v_j)}{\sqrt{d(v_i)d(v_j)}}, & \text{if } v_i \text{ and } v_j \text{ are adjacent,} \\ 0, & \text{otherwise.} \end{cases}$$

The normalised Laplacian will be the tool for our purposes in the chapter to come. Some properties of $\mathcal{L}$ will follow next.

3.2 Spectral Theory

We start this section by recalling the definition of eigenvector and eigenvalue. A vector v is an eigenvector with eigenvalue λ of the matrix M if

$$Mv = \lambda v.$$

All the eigenvalues of $\mathcal{L}$ are non-negative, and the smallest eigenvalue of $\mathcal{L}$ is always zero. To see this, let the eigenvalues of $\mathcal{L}$ be denoted

$$\lambda_0 \leq \lambda_1 \leq \ldots \leq \lambda_{n-1}.$$

Now, consider the Rayleigh quotient, $R(\mathcal{L}, g)$ defined as

$$R(\mathcal{L}, g) = \frac{g^T \mathcal{L} g}{g^T g}. \tag{3.10}$$

$\mathcal{L} \in \mathbb{R}^{n \times n}$, and $g \in \mathbb{R}^{n \times 1}$ is a vector. The minimum value of $R(\mathcal{L}, g)$ is equal to the smallest eigenvalue λ_0 of $\mathcal{L}$, when g is the corresponding eigenvector. Similarly, the maximum value of $R(\mathcal{L}, g)$ is equal to the largest eigenvalue λ_{n-1} of $\mathcal{L}$, when g is the eigenvector corresponding to λ_{n-1}. For further details, see [12]. Let $g = D^{\frac{1}{2}} f$, and let f denote an arbitrary function which assigns to each vertex v_i a real value $f(v_i)$. Then we have

$$\begin{aligned} R(\mathcal{L}, g) &= \frac{g^T \mathcal{L} g}{g^T g} \\ &= \frac{g^T D^{-\frac{1}{2}} L D^{-\frac{1}{2}} g}{g^T g} \end{aligned}$$

$$= \frac{f^T L f}{f^T D f} \tag{3.11}$$

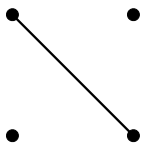
Now let $G_{1,2}$ be an unweighted graph on two vertices. The Laplacian $L_{G_{1,2}}$ to the graph $G_{1,2}$ is then

$$L_{G_{1,2}} = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}.$$

From this, using the vector $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, it's obvious that

$$x^T L_{G_{1,2}} x = (x_1 - x_2)^2. \tag{3.12}$$

Let $G_{u,v}$ be a graph on n vertices and only one edge between vertices u and v. Its Laplacian, $L_{G_{u,v}}$ is a n-by-n matrix whose only non-zero entries are in the intersections of rows and columns u and v. For example, the graph $G_{1,4}$ in Figure 3.7 has the Laplacian shown in (3.13).



$G_{1,4}$

Figure 3.7:

$$\begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix} \tag{3.13}$$

For a weighted graph G, we now define

$$L_G = \sum_{(u,v)\in E} w(u,v) L_{G_{u,v}}. \tag{3.14}$$

This definition is equivalent to the definition (3.7). Combining (3.14) and (3.12), we see that

$$x^T L_G x = \sum_{(u,v)\in E} w(u,v)(x_u - x_v)^2.$$

Continuing on the derivation in (3.11), we get

$$\frac{f^T L f}{f^T D f} = \frac{\sum_{(v_i,v_j)\in E} (f(v_i) - f(v_j))^2 w(v_i, v_j)}{\sum_i f(v_i)^2 d(v_i)} \geq 0. \tag{3.15}$$

If we let $f = \mathbf{1}$ in (3.15), where $\mathbf{1}$ is the all-ones vector, then we get equality in the last step. The corresponding eigenvector is then $g = D^{\frac{1}{2}}\mathbf{1}$. So we have now shown that λ_0 is always equal to zero. What is interesting here is that the multiplicity of zero among the eigenvalues is equal to the number of connected components in the graph [18]. If $\lambda_1 > 0$, then the graph is connected. If $\lambda_1 = 0$, the graph is disconnected. Also, the further λ_1 is from zero, the "better" connected the graph is. λ_1 is often called *the Fiedler value*. The eigenvector corresponding to the Fiedler value is called *the Fiedler vector*. An example of some graphs with Fiedler values is illustrated in Figure 3.8.

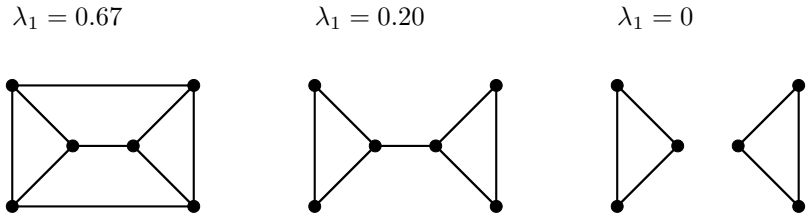


Figure 3.8:

There exists some interesting properties of the Fiedler vector that are worth mentioning. The Fiedler vector describes symmetry of the graph that it is related to. Look at the graph in the middle of Figure 3.8. The graph clearly is symmetric. Now look at the Fiedler vector, it is

$$\begin{pmatrix} -0.4451 \\ -0.4451 \\ -0.3220 \\ 0.3220 \\ 0.4451 \\ 0.4451 \end{pmatrix}$$

The vertices are numbered from top to bottom, left to right. Vertices that is equally related to the rest of the graph gets the same value in the vector. Though sometimes, they differ in the sign, as for vertices 3 and 4 above. Let us take another example. Look at the graph in Figure 3.9, and let the vertices be numbered as in the figure.

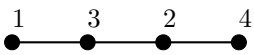


Figure 3.9:

The Fiedler vector to this graph is

$$\begin{pmatrix} -0.5774 \\ 0.4082 \\ -0.4082 \\ 0.5774 \end{pmatrix}$$

If we sort the rows of the vector in ascending order, the order of the rows is 1-3-2-4. Just as the vertices are ordered in the graph. From the magnitude of the entries in the Fiedler vector, we can also see that vertices 1 and 4 are positioned furthest away from the center of the graph, while vertices 3 and 2 are positioned close to the center. Symmetry is also shown in this example.

3.2.1 Spectral partitioning

The idea of spectral partitioning is to use the *Fiedler vector* to find a partition. A *partition* of a graph G is a division of the vertices in V into two disjoint subsets, S and $\bar{S}$, where $\bar{S}$ is the complement of S, i.e.

$$\bar{S} = V - S. \tag{3.16}$$

Now let $\bar{u} = (u_1, ..., u_n)$ be the *Fiedler vector* of the normalised Laplacian of a graph G. Then, for every i, let the vertice v_i correspond to u_i. We then find a value s, splitting the vertices into two subsets such that $v_i \in S$ if $u_i > s$, and $v_i \in \bar{S}$ if $u_i \leq s$. A few ways of choosing s are:

- **bisection**: s is the median of $\{u_1, ..., u_n\}$.
- **sign cut**: $s = 0$.
- **gap cut**: s is a value in the largest gap in the sorted Fiedler vector.

To illustrate the method of spectral partitioning we will give an example. Consider the graph in Figure 3.10. The edge weights are seen in the figure in connection to the edges. The vertex weights are all zero. The Fiedler vector to the graph is seen in (3.17).

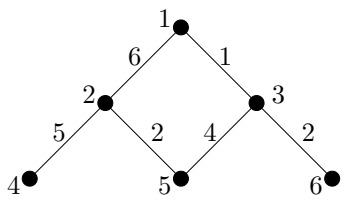


Figure 3.10: A graph with edge weights to be partitioned.

$$\begin{pmatrix} -0.2523 \\ -0.4183 \\ 0.5701 \\ -0.3603 \\ 0.3571 \\ 0.4232 \end{pmatrix} \tag{3.17}$$

If using sign cut, the cut is made along edges $(1, 3)$ and $(2, 5)$. This seems reasonable, since we want the cut to be made along the weakest edges. The result is seen in Figure 3.11.

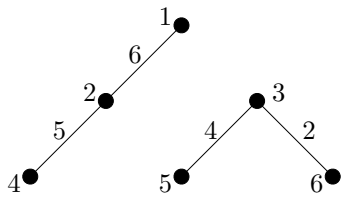


Figure 3.11: A graph after the partitioning.

We know that the eigenvector to the smallest eigenvalue is $D^{\frac{1}{2}}\mathbf{1}$. Since the normalised Laplacian is symmetric, the eigenvectors are orthogonal. All the entries of $D^{\frac{1}{2}}\mathbf{1}$ are non-negative, so then the Fiedler vector must have both positive and negative entries. This is the reason that sign cut often is used.

Where in the graph the cut is made depends very much on the weights of the edges and vertices. Take for example the graph to the left in Figure 3.12. Assume that the weights on all the edges are one, and that there is no weight on the vertices. Then the cut is made on edge $(5, 7)$, seen to the right in Figure 3.12. (Here we have used sign cut). The reason why the cut is made on $(5, 7)$ and not on $(3, 5)$, in this symmetric graph, is because of how we above define the vertices to be divided into subsets. The equality sign makes the difference.

If we now increase the weight on edge $(5, 7)$, so that $w(5, 7) = 2$, then the cut is instead made on edge $(3, 5)$. This is illustrated in Figure 3.13.

When it comes to the vertex weights, a similar reasoning can be made. Let us return to the graph to the left in Figure 3.12, where all the edges have weight one. Also let all the vertices have weight one. Making a cut now gives us the same result as in Figure 3.12, that is, the cut is made on edge $(5, 7)$. If we instead increase the weight on one of the vertices, e.g. $w(1, 1) = 2$, then the cut is made on edge $(3, 5)$. One can imagine the graph to balance on its equilibrium point, and the cut to be made close to that point. If increasing the weight on the first vertex even more, say $w(1, 1) = 25$, then the equilibrium point is clearly close to that vertex. And as one might have guessed, this time the cut is made on edge $(1, 3)$, seen in Figure 3.14.

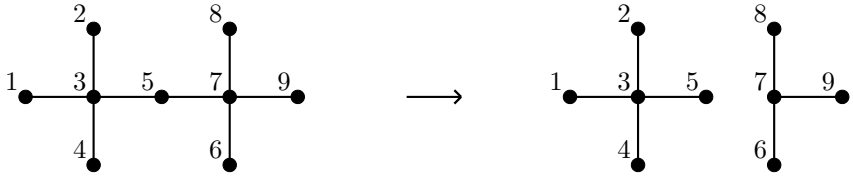


Figure 3.12: A graph with edge weights equal to one, and vertex weights equal to zero.

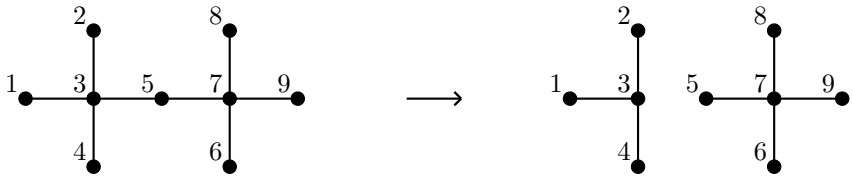


Figure 3.13: A graph with edge weights equal to one, except for edge $(5, 7)$, which has weight equal to two. The vertex weights is zero.

3.2.2 Connectivity

For further details and proofs on this section, consult [4, 18]. When we have partitioned the vertices of a graph G into two subgraphs, S and $\bar{S}$, we want to find out if it is a good partitioning. For this we use the conductance. The conductance $\phi(S)$, of a set of vertices S is defined as

$$\phi(S) = d(V)\frac{|E(S,\bar{S})|}{d(S)d(\bar{S})}, \tag{3.18}$$

where

$$d(S) = \sum_{v_i \in S} d(v_i),$$

and $d(v_i)$ is the degree of v_i as defined earlier in (3.5). $E(S,\bar{S})$ is the set of edges with one end is S and the other end in $\bar{S}$. $|E(S,\bar{S})|$ is the total weight of these edges, that is

$$|E(S,\bar{S})| = \sum_{v_i \in S, v_j \in \bar{S}} w(v_i, v_j).$$

It is obvious that $\phi(S) = \phi(\bar{S})$. Another measure that will be used is the sparsity $sp(S)$, of a set of vertices S. It is defined as:

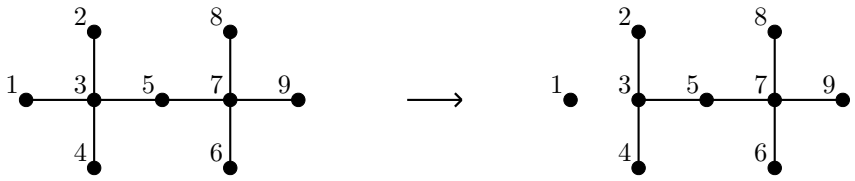


Figure 3.14: A graph with edge weigths equal to one, and vertex weights equal to one, except for vertex 1, which has weight 25.

$$sp(S) = \frac{|\,E(S,\bar{S})\,|}{\min(d(S), d(\bar{S}))} \tag{3.19}$$

Without loss of generality we may assume that $d(S) \leq d(\bar{S})$. Also assume that S and $\bar{S}$ are nonempty, and that G is connected. Then, since

$$1 < \frac{d(V)}{d(\bar{S})} \leq 2 \tag{3.20}$$

we have the following relation between the conductance and the sparsity,

$$\phi(S) < sp(S) \leq \frac{\phi(S)}{2}.$$

Now, the conductance of a graph G is the minimum of conductance over all partitionings. In other words,

$$\phi_G = \min_S \phi(S)$$

The further $\phi(G)$ is from 0, the better the graph is connected, and the harder it is to find a good partitioning. In the same way as for conductance of a graph, we get the sparsity of a graph G by:

$$sp_G = \min_S sp(S)$$

The relation between the sparsity and the Fiedler value λ_1 can be seen using the Cheeger inequality

$$2sp_G \geq \lambda_1 \geq \frac{sp_G^2}{2}. \tag{3.21}$$

The corresponding inequality for the conductance is

$$2\phi_G > \lambda_1 \geq \frac{\phi_G^2}{8}. \tag{3.22}$$

The inequalities tells us that when λ_1 is small, then there exists a cut of small sparsity or conductance. The second inequality can easily be deduced from the first one using (3.20). For the first one, see [4, p.26].

Chapter 4

The Computation of Word Senses

In this chapter, we use the idea of semantic mirroring described in chapter 2, to create a graph of words. We then use the graph partitioning methods from chapter 3.2.1 to partition the graph and make sense groups to a seed word. Last in the chapter is an example illustrating the method.

4.1 The translation matrix

In the experiments, we have worked with an English-Swedish lexicon consisting of 14 850 English adjectives, and its corresponding Swedish translations. To be able to work with the lexicon in an easily manner, the translation matrix $\mathbf{B}$ was created. Along with $\mathbf{B}$ we created two lists, one for the English and one for the Swedish words in the lexicon. The rows of $\mathbf{B}$ correspond to the English words, i.e. if a English word is found at position j in the English list, then row j in $\mathbf{B}$ coresond to that word. In the same way, every column in $\mathbf{B}$ correspond to a Swedish word k. If in the lexicon a English word j is translated to a Swedish word k, then $\mathbf{B}_{jk} = 1$, where $\mathbf{B}_{jk}$ is the value at position (j, k) in $\mathbf{B}$. If a English word j does not translate to a Swedish word k, then $\mathbf{B}_{jk} = 0$.

$\mathbf{B}$ has the dimension 14850×20714, i.e. $\mathbf{B} \in \mathbb{R}^{14850 \times 20714}$. Since every English word translates only to a few Swedish words, most of the elements in $\mathbf{B}$ are equal to 0. Such a matrix is called a *sparse* matrix. In $\mathbf{B}$, only about 0.016 % of the elements are nonzero. When storing such a matrix in MATLAB, only the nonzero elements are stored, along with their positions.

4.2 Translation

Since this thesis is largely about translation, we need a good way of performing translations. This is where we will have good use of our translation matrix $\mathbf{B}$. To mathematically express a English word j, we create a vector $\bar{e}_j$ defined by

$$\bar{e}_j(i) = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j, \end{cases}$$

where $\bar{e}_j(i)$ is element i in the vector $\bar{e}_j \in \mathbb{R}^{14850\times 1}$. The corresponding vector for a Swedish word k is defined in the same way, and is denoted by $\bar{s}_k \in \mathbb{R}^{20714\times 1}$. To translate a Swedish word k into English, we perform the matrix multiplication $\mathbf{B}\bar{s}_k$. The result is a vector $\mathbf{B}_{\cdot k}$, i.e. the kolumn k in $\mathbf{B}$. This vector represents the possible transations of k. If we instead want to translate from English to Swedish, the matrix multiplication $\mathbf{B}^T\bar{e}_j = \mathbf{B}^T_{j\cdot}$ is performed. $\mathbf{B}^T$ stands for the transpose of $\mathbf{B}$, and $\mathbf{B}_{j\cdot}$ is the j:th row in $\mathbf{B}$.

When translating several words at the same time, we make a matrix with columns consisting of the words corresponding vectors. Put $\mathbf{S} = (\bar{s}_{k_1}\bar{s}_{k_2}\dots\bar{s}_{k_m})$, and perform the multiplication $\mathbf{BS}$. The result will be a matrix $\mathbf{E} = (\bar{e}_{j_1}\bar{e}_{j_2}\dots\bar{e}_{j_m})$ with the same number of columns as $\mathbf{S}$.

4.3 The Computation of Word Senses

The following pages will explain in detail the method used in this thesis. We are going to work with the English language, and take help by some examples. So we start with an English word, called *eng1*[1]. We look up its Swedish translations. Then we look up the English translations of each of those Swedish words. We have now performed one "mirror-operation". In mathematical notation:

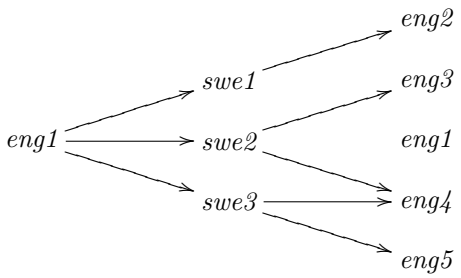
$$\mathbf{B}\mathbf{B}^T\bar{e}_{eng1}.$$

These words, i.e. the words in the invers t-image of *eng1*, are the words that are semantically close to *eng1*. But there is one problem. The original word should'nt be here. Therefore, in the last translation, we modify the matrix $\mathbf{B}$, by replacing the row in $\mathbf{B}$ corresponding to *eng1*, with an all-zero row. Call this new modified matrix $\mathbf{B}_{mod1}$. So instead of the matrix multiplication performed above, we start over with the following one:

$$\mathbf{B}_{mod1}\mathbf{B}^T\bar{e}_{eng1}. \tag{4.1}$$

To make it more clear in a linguistic perspective, maybe the following figure[2] will help.

(4.2)



The words to the right in the picture above (*eng2,...,eng5*) are the words we want to divide into senses. To do this, we need some kind of relation between the words. Therefore we continue to translate, and perform a second "mirror operation". To keep track of what each word in the inverse t-image are being translated to, we must first make a small modification. We have so far done

[1] Short for English1. We will use swe for Swedish words.
[2] The arrows indicates translation.

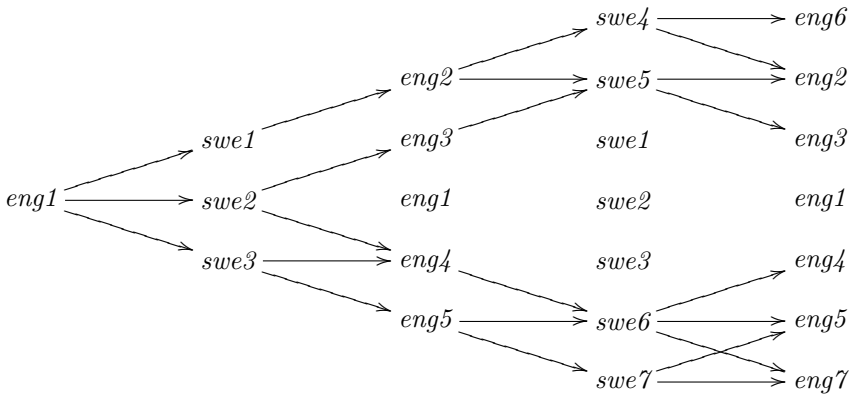
the operation (4.1), which gave us a vector, call it $e \in \mathbb{R}^{14850\times 1}$. The vector e consists of nonzero integers in the places corresponding to the words in the invers t-image, and zeros everywhere else. We make a new matrix $\mathbf{E}$, with the same number of rows as e, and the same number of columns as there are nonzeros in e. Now go through every element in e, and when finding a nonzero element in row i, and if it is the j:th nonzero element, then move it to row i and column j in $\mathbf{E}$. The procedure is illustrated in (4.3).

$$\begin{pmatrix} 1 \\ 0 \\ 2 \\ 1 \\ 0 \\ 3 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.3}$$

When doing our second "mirror operation", we don't want to translate through the Swedish words *swe1,...,swe3*. We once again modifies the matrix $\mathbf{B}$, this time replacing the colums of $\mathbf{B}$ corresponding to the Swedish words *swe1,...,swe3*, with zeros. Call this second modified matrix $\mathbf{B}_{mod2}$. We create a new matrix $\mathbf{E}$, as described in (4.3), and then performs:

$$\mathbf{B}_{mod2}\mathbf{B}_{mod2}^{T}\mathbf{E} \tag{4.4}$$

Continuing on (4.2), what we have done looks something like this:



Now we have got the desired relation that we wanted within *eng2,...eng5*. If in (4.4), taking only the rows corresponding to *eng2,...eng5*, we get a symmetric matrix $\mathbf{A}$, also known as the adjacency matrix. To this matrix there is related a graph. The adjacency matrix and the graph to our example is illustrated below.

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix} \tag{4.5}$$

The adjacency matrix should be interpreted in the following way. The rows and the columns corresponds to the words in the inverse t-image. Following our

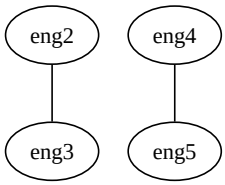


Figure 4.1:

example, *eng2* corresponds to row 1 and column 1, *eng3* corresponds to row 2 and column 2, and so on. The elements on position (i, i) are the vertex weights (as in (3.2) and (3.3)). The vertex weight associated with a word, describes how many translations that word has in the other language, e.g. *eng2* translates to *swe4* and *swe5* that is translated back to *eng2*. So the vertex weight for *eng2* is 2, as also can be seen in position $(1, 1)$ in (4.5). The vertex weight is highly related to assumption **(2)** on page 3. That is, words with a wide meaning tend to have a high number of translations, which is the same as saying that the vertex weight is high.

The elements in the adjacency matrix on position $(i, j), i \neq j$ are the edge weights. These weights are associated with two words, and describes how many words in the other language that both word i and j are translated to. E.g. *eng5* and *eng4* are both translated to *swe6*, and it follows that the weight, w(*eng4,eng5*) = 1. If we instead would take *eng5* and *eng7*, we see that they both translate to *swe6* and *swe7*, so the weight between those words, w(*eng5,eng7*) = 2. (But this isn't shown in the adjacency matrix, since *eng7* is not a word in the inverse t-image). The edge weight is highly related to assumption **(1)** on page 3. Semantically closely related words tend to have strongly overlapping sets of translations, which is the same as saying that the edge weight between two words is high.

The example illustrated in Figure 4.1 gave as a result two graphs that are not connected. Dyvik argues that in such a case the graphs represent two groups of words of different senses. In a more realistic example one is likely to obtain a much larger, and connected, graph. In that case, it may be reasonable to partition the graph into two subgraphs without breaking more than a small number of edges. Our hopes is that the two subgraphs are going to represent different senses.

By using the spectral graph partitioning methods described in section 3.2.1, partition the graph into two subgraphs. To find the splitting value s, we use a slightly different approach than in section 3.2.1. Take the sorted Fiedler vector of the normalised Laplacian $\mathcal{L}$ (3.9). For each of the $n - 1$ possible cuts in the sorted Fiedler vector, calculate the conductance, and choose the cut which produces the lowest conductance. This way we often get a better cut than we would with sign cut, for example.

When the splitting value s is found, delete all the edges in the graph with one end in S and one end in $\bar{S}$ (3.16). This is practically done in the following

way. In the adjacency matrix A, replace the elements A_{ij} and A_{ji} with zeros, where $v_i \in S$ and $v_j \in \bar{S}$. The procedure is illustrated in Figure 4.2 and in (4.6).

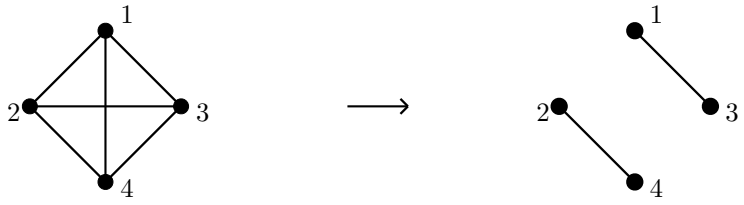


Figure 4.2:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 3 & 1 \\ 1 & 1 & 1 & 4 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \\ 1 & 0 & 3 & 0 \\ 0 & 1 & 0 & 4 \end{pmatrix} \tag{4.6}$$

If we want to, we can continue and partition one of the subgraphs even more. Choose the largest (in number of vertices) of the subgraphs, and derive a new adjacency matrix A_{new} by in the adjacency matrix A keeping only the rows and columns corresponding to the vertices in the largest subgraph. Let us show this by an example. If in Figure 4.2 we want to derive a new adjacency matrix to the vertices v_2 and v_4, then the procedure is illustrated in (4.7).

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \\ 1 & 0 & 3 & 0 \\ 0 & 1 & 0 & 4 \end{pmatrix} \longrightarrow \begin{pmatrix} 2 & 1 \\ 1 & 4 \end{pmatrix} \tag{4.7}$$

If in this new graph (a subgraph to the original graph) we perform a cut and thereby delete edges, then the operation of replacing elements with zeros (4.6), is performed on the original adjacency matrix A, and not on A_{new}. This is done to facilitate the calculations of finding the largest subgraph. The procedure of partition the subgraphs can then be continued for each of the subgraphs obtained. When to stop the partition is decided of some kind of stopping criteria. Different stopping criteria is discussed in section 5.2.3.

4.4 Examples

Below follows some real examples. A short example of the word *slithery* is illustrated with the associated graphs. For larger examples we only illustrate the words and the groups they are divided into.

Slithery

After performing two mirroring operations (4.4), the graph in Figure 4.3 is obtained. The words, and which vertex they belong to are presented in Table 4.1. Note that vertices v_6 and v_{11} are disconnected from the rest of the graph. The interested reader can find the edge and vertex weights in (4.8), but that information is precluded from the graph for simplicity.

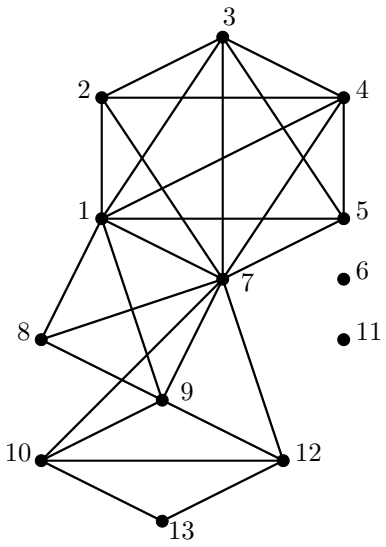


Figure 4.3: The graph to *slithery*.

After one partitioning of the connected component of the graph, we obtain the subgraphs illustrated in the left of Figure 4.4. The conductance of this first cut is $\phi = 0.1157$. We then partition the largest of those subgraphs, and obtain the graph to the right in Figure 4.4. The conductance in the last cut was $\phi = 0.3233$. The word senses of slithery after two partitionings is shown in Table 4.2.

vertex	word
1	smooth
2	slick
3	lubricious
4	slippery
5	glib
7	sleek
8	slimy
9	smooth-faced
10	oleaginous
12	oily
13	greasy
6	saponaceous
11	slippy

Table 4.1: The words in the inverse t-image of *slithery*.

slimy	glib	oleaginous	saponaceous	slippy
smooth-faced	slippery	oily		
smooth	lubricious	greasy		
sleek	slick			

Table 4.2: The word groups obtained after two partitionings.

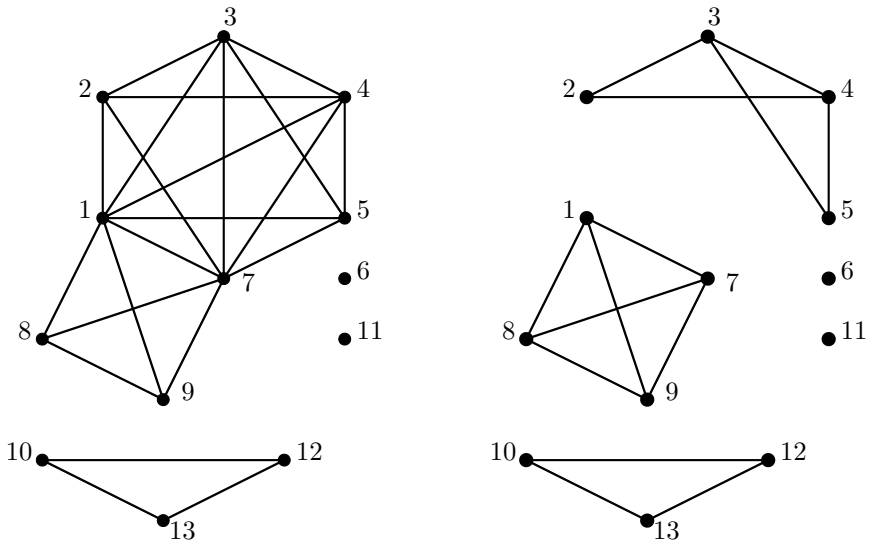


Figure 4.4: The graph to *slithery* after one (left) and two (right) partitionings.

$$\begin{pmatrix}
28 & 2 & 1 & 1 & 1 & 0 & 5 & 1 & 2 & 0 & 0 & 0 & 0 \\
2 & 14 & 1 & 1 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 4 & 2 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 2 & 3 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 5 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
5 & 2 & 1 & 1 & 1 & 0 & 15 & 1 & 2 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 5 & 1 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 0 & 2 & 1 & 5 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 6 & 0 & 3 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 3 & 0 & 6 & 3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 3 & 14
\end{pmatrix} \tag{4.8}$$

Global

We start with the word *global*, and after the mirroring operation (4.4), we get the words in Table 4.3.

absolute	international
aggregate	mondial
all-out	one-piece
clear	outright
complete	round
entire	teetotal
full	total
full-length	unbroken
full-scale	universal
intact	utter
integral	whole
integrate	worldwide

Table 4.3: The words from *global* after the mirroring operation. (Only one group).

After partitioning the graph six times, we get the groups of words illustrated in Table 4.4.

intact whole full-length one-piece clear round full	absolute utter complete all-out teetotal outright entire total	integrate unbroken integral	aggregate full-scale
worldwide international	mondial	universal	

Table 4.4: The word senses of *global*.

Chapter 5

Results, observations and discussions

In this chapter an evaluation of the method in chapter 4 have been made, and the results are displayed. Some discussions are made about questions that have been brought up during the work.

5.1 Results

To evaluate the method presented in section 4.3, 10 examples was derived. To simplify the evaluation for the evaluators, which is Swedish, the choice was made that the examples would be in Swedish. The derivation of the examples was made in the following way. A 10×1 array with entries being integers randomly choosen in the interval [1,20714] was derived. Then the method in section 4.3 was applied to the Swedish words corresponding to the entries in the array. If the graph obtained from each word contained at least 8 vertices, then that word was subject to evaluation. To find 10 words, with at least 8 vertices in the graph, five arrays with a total of 50 words was derived. The 10 Swedish words used in the evaluation is presented, with a possible translation, in Table 5.1.

bamsig	bonny, bouncing
borterst	furthest
kryddad	spiced, spicy
moraliskt_fördärvad	corrupt
proppfull	jampacked, tight
slutlig	final, ultimate
soft	soft
starkt_kuperad	rough (terrain)
vid_god_hälsa	healthy
välförtjänt	well-deserved

Table 5.1: The 10 words used in the evaluation set.

Two evaluators did the evaluation individually and later conferred to come to a

joint decision. Criteria for determining whether two word candidates belonged to the same sense were:

- Candidates can be used interchangeably but still preserving the core sense.

- Candidates can all be used with the same kind of head word and still preserving the core sense.

Method used for evaluation:

1. No. of sense groups.
   Count the number of suggested sense groups that hold at least two candidate words, disregarding two-word phrases.

2. No. of consistent sense groups. Decide for each group whether it contains one single sense.

   a. If all candidate words in the group are judged to have the same meaning (sense), the evaluation score is set to 1 for that group.

   b. If at least 2/3 of the candidates have the same meaning (sense), the evaluation score is set to 0.5 for that group.

3. No. of "seed-word-synonymous" sense groups. Decide whether the group is synonymous with the seed word. The group is regarded as being synonymous with the seed word, if at least 2/3 of the candidate words in the group can be regarded as being synonymous with the seed word. If the group is judged to be synonymous with the seed word, the evaluation score is set to 1, otherwise to 0.

Four methods were subject to evaluation. In method 1 we used conductance (3.18) as a measure. In method 2 we used sparsity (3.19). For method 3 and 4 we did not take into account the vertex weights, so all the vertex weights were set to zero, as described in section 5.2.2. In method 3 we used conductance, and in method 4 we used sparsity.

Below follows an example of how the evaluation is made. The word is *välförtjänt*, and method 1 is used. After we have partitioned the graph to the seed word *välförtjänt* six times, we get the groups seen in Table 5.2.

Group1	**Group2**	**Group3**	**Group4**	**Group5**
berättigad	rimlig	rättfärdig	noggrann	rätt
välgrundad	rättrådig	rättskaffens	exakt	riktig
befogad	skälig		precis	
	rättvis			

Table 5.2: The sense groups of *välförtjänt*.

Five groups found, illustrated in Table 5.2.

- **Group1** contains three consistent word candidates belonging to the same sense. All of the candidates are consistent with the seed word *välförtjänt*. → 1 for consistency score and 1 for synonym score.
- **Group2** contains three words consistent with each other (*rimlig, skälig, rättvis*). The candidate word *rättrådig* is however not synonymous with the other three. However, the three consistent word candidates are synonymous with the seed word. → 0.5 for consistency score and 1 for synonym score.
- **Group3** contains two internally consistent word candidates, but they are not synonymous with the seed word. → 1 for consistency score and 0 for synonymous score.
- **Group4** contains three internally consistent word candidates, but they are not synonymous with the seed word. → 1 for consistency score and 0 for synonymous score.
- **Group5** contains two internally consistent word candidates, but they are not synonymous with the seed word. → 1 for consistency score and 0 for synonymous score.

In total 5 identified sense groups. If we sum up the scores, we get a 4.5 consistency score and a synonymous score of 2. The 10 evaluated words and their respective scores are shown in Table 5.3.

Seed word	**Method used for evaluation (see page 30)**	**Method 1. With vertex weights and conductance**	**Method 2. With vertex weights and sparsity**	**Method 3. Without vertex weights and conductance**	**Method 4. Without vertex weights and sparsity**
bamsig	1	3	**4**	**4**	**4**
	2	2.5	**4**	**4**	**4**
	3	1	**1**	**1**	**1**
borterst	1	2	4	4	**5**
	2	1	2	2	**2.5**
	3	1	1	1	**2**
kryddad	1	5	6	**7**	7
	2	1.5	2	**3.5**	2
	3	0	0	**0**	0
moraliskt_ fördärvad	1	**5**	**5**	7	7
	2	**3.5**	**3.5**	3.5	3
	3	**1**	**1**	1	1
proppfull	1	5	7	**6**	**6**
	2	2	3.5	**3.5**	**3.5**
	3	1	2	**2**	**2**
slutlig	1	5	**7**	6	6
	2	4.5	**7**	5.5	5.5
	3	1	**2**	2	2
soft	1	4	6	**5**	5
	2	3.5	4.5	**5**	4.5
	3	2	3	**3**	4
starkt_ kuperad	1	6	**8**	5	5
	2	2	**3.5**	1.5	1
	3	0	**0**	0	0
vid_god_ hälsa	1	6	**6**	7	7
	2	4	**4.5**	4.5	5.5
	3	1	**2**	1	1
välförtjänt	1	5	5	**6**	7
	2	4.5	4	**5.5**	5.5
	3	2	2	**2**	2

Table 5.3: The score of the evaluated words.

It is difficult to see any clear pattern regarding which method gives the best performance. One criterion can be the proportion of generated well-formed, consistent sense groups against the total number of sense groups. For example, for the last word "välförtjänt", we see that it is the third method that has the highest score of the four (5.5/6=0.917), compared to 4.5/5, 4/5 and 5.5/7. This

will only give one dimension, as the evaluation score is relatively coarse, and does not take into consideration of how large the groups are. However, it is an indication.

The number of synonymous groups (relative to the seed word) is more of a side issue. This only tells the proportion of true synonym groups in relation to the starting word. All well-formed senses groups are to be seen as related synonym sets that has an indirect relation to the seed word, but they are not synonyms.

5.2 Observations and diskussions

5.2.1 Vertex splitting

One possible outcome that have been noted is when a connected graph have been partitioned, and when looking at the words, there seems to exist a word that could fit in both of the accuired word senses. For example, take the group of words[1] in Table 5.4. Dividing the words once gives the groups in Table 5.5.

easy
floaty
light
not_heavy
plain
primitive
simple

Table 5.4: A group of words to be divided.

easy	easy
floaty	plain
light	primitive
not_heavy	simple

Table 5.5: The words in Table 5.4 divided into two groups.

We see that the word *easy* is part of both groups. In the method presented in chapter 4, there is not possible to let a word be part of two or more groups. We could also look at this in a graph-theoretical approach. Consider the graph to the left in Figure 5.1. The partitioning methods performs a cut, dividing the graph into subgraphs, as shown to the right in Figure 5.1, (using sign cut).

But when looking at the edges adjacent to vertex 4 before the cut, one could think that the vertex just as well could belong to the other subgraph after the cut. To verify this, one way is to check the conductance of the two possible cuts around the vertex in question. If the difference is small enough, then maybe we should draw the conclusion that the word assigned to the vertex should belong

[1] This example is constructed, and the partition of the words has not been made automatically.

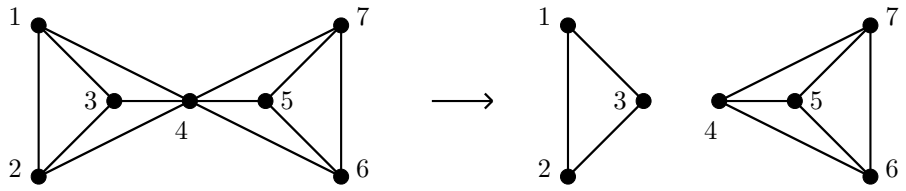


Figure 5.1: Partition of a graph. Edge weights are equal to 1, and vertex weights are equal to 0.

to both word senses. One possible solution could be the following. Let's undo the cut, and before making a new cut, split the vertex 4 using the operation vertex splitting described in chapter 3.1.2. Let the edges with one end in vertices $\{1, 2, 3\}$ and the other end in vertex 4 still have one end in vertices $\{1, 2, 3\}$ but the other end in vertex $4'$. Similarly, let the edges with one end in vertices $\{5, 6, 7\}$ and the other end in vertex 4 still have one end in vertices $\{5, 6, 7\}$ but the other end in vertex $4''$. Also, let the new edge $e = (4', 4'')$ have a low weight, that is $w(4', 4'') \leq 1$. The graph obtained is illustrated to the left in Figure 5.2. Making a new cut now will partition the graph according to Figure 5.2. The word assigned to vertex 4, and thus also assigned to vertices $4'$ and $4''$, is now part of two word senses.

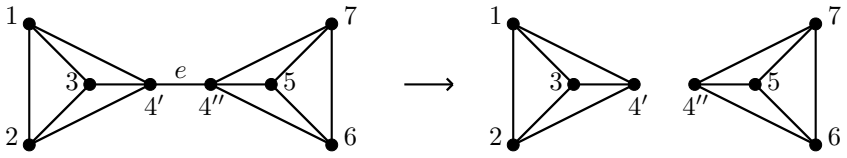


Figure 5.2: Partition of a graph after splitting a vertex.

5.2.2 Vertex weights

During the work of this thesis the question was raised, whether vertex weights should be taken into consideration when calculating a partition of a graph.

Vertex weights gives us no information of how the words in the graph are connected to each other. What the vertex weights do tell us is how many translations the word associated to the vertex, have in the other language. This was described more thoroughly in section 4.3. The weight also gives us an indication of how wide meaning the word have. For example, take the words *animal* and *cat*. *Animal* has a wide meaning, and probably a high vertex weight.

Cat, on the contrary, has a more narrow meaning, and therefore also a smaller vertex weight.

When partitioning a graph it is the connections between the vertices that have the largest effect in where the cut is made. These connections are the edges, and their weights. Every edge, and edge weight, are connected to two words. Let α be the set of translations of the first word, and β be the set of translations of the second word. Then the edge weight w is equal to the number of words in the intersection of α and β, i.e. $w =\mid \alpha \cap \beta \mid$.

Since the vertex weights effect how the partitioning is made, which has been shown in section 3.2.1, one might wonder whether the vertex weights really should be taken into account. To make a desicion in this question, more evaluation than is presented in the results in section 5.1 need to be done.

5.2.3 Stopping criteria for graph partitioning

In section (4.3) we partitioned a graph into subgraphs by making a cut in the largest subgraph. This procedure was then carried out a number of times. When to stop has so far been decided manually, by looking at the words in the groups. It would be very interesting to find a automatic way of deciding when to continue partition the graph, and when to stop. But there is no easy solution to the problem. Imagine two graphs with disjoint set of words, and let them, in graph theoretical terms, be very similar to each other. Then they would be partitioned in a similar way. But since it is how the words are being grouped together that decides if the partition is good, and the two graphs consists of disjoint set of words, it follows that one graph can get a good partition, while the other one can get a bad partition.

One possible solution to this could be to use the Fiedler value, λ_1. According to the Cheeger inequalities (3.21) and (3.22), we know that when the Fiedler value is small, it is possible to find a cut with small conductance and sparsity. So by looking at the Fiedler value, maybe one could decide if the graph should be partitioned or not. Another important factor is the size of the graph. A small graph, for example one with less then 10 vertices, should probably only be partitioned a few times. However, a large graph could be partitioned very many times, with a good result. (The largest graph in the experiments consists of 672 vertices, and is obtained from the word *heavy*.)

5.3 Summary and conclusion

In this report we have paired the method Semantic mirrors with spectral graph partitioning methods, to compute separate word senses.

To achieve this we created an algorithm that follows the Semantic mirrors method to obtain a graph. The vertices of the graph was assigned words that are semantically related to a seed word. From the adjacency matrix obtained from our algorithm, we created the normalised Laplacian. The eigenvalues and eigenvectors of the normalised Laplacian was computed, and we used the sorted eigenvector belonging to the second smallest eigenvalue to create cuts in the graph. For each such cut we calculated the conductance or sparsity, and choosed the cut that gave the lowest value. When the graph was partitioned, the words

assigned to the vertices was divided into groups, describing the word senses of the seed word.

The method proposed in this report seem to work quite well. The groups of words, obtained after the partition of the graph, are most often consistent. That is, the words in the group describes similar meanings. However, the groups are not so often synonymous with the seed word. The method also fails when it comes to words that have a one-to-one correspondence in the lexicon used, for example *geopolitical* - *geopolitisk*. In that case, there is no graph obtained at all. To sum up, we can say that the method is good at creating word senses, although they are not always synonymous with the seed word.

5.4 Future work

There is room for many more investigations of the approach outlined in this report. One is to explore the possibility to have a vertex (word) belong to multiple synsets, as discussed in section 5.2.1. In the present solution a vertex belongs to only one partition of a graph, making it impossible to have the same word belong to several synsets. Another one is to investigate the properties of graphs to see whether it is possible to automatically measure how close a seed word is to a particular synset. Furthermore, more thorough evaluations of larger data sets would give more information on how to combine similar synsets which were generated from distinct seed words. It would also be interesting to test the method on other lexica, and perform more experiments with the different tolerances involved.

Bibliography

[1] A. Björn, B. O. Turesson. *Diskret Matematik*. Matematiska Institutionen, Linköpings Universitet, 2001.

[2] M. Bolander. *Funktionell svensk grammatik*. Liber. 2005.

[3] J. A. Bondy, U. S. R. Murty. *Graph Theory*. Springer, 2008.

[4] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, Providence, Rhode Island. 1997.

[5] M. Diab, P. Resnik, *An Unsupervised Method for Word Sense Tagging using Parallel Corpora*, Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics.(2002) 255-262.

[6] H. Dyvik. *A Translational Basis for Semantics*. In: Stig Johansson and Signe Oksefjell (eds.): Corpora and Crosslinguistic Research: Theory, Method and Case Studies, pp. 51-86. Rodopi. 1998.

[7] H. Dyvik. *Translations as a Semantic Knowledge Source*. 2003.

[8] H. Dyvik. *Translations as a Semantic Knowledge Source*. Proceedings of the Second Baltic Conference on Human Language Technologies, Tallinn. 2005.

[9] H. Dyvik. *Translations as Semantic Mirrors: From Parallel Corpus to Wordnet*. Section for Linguistic Studies, University of Bergen. 2002.

[10] http://www.stanford.edu/∼ dgleich/

[11] C. Godsil, G. Royle. *Algebraic Graph Theory*. Springer-Verlag New York, Inc. 2001.

[12] G. H. Golub, C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore and London. 1996.

[13] N. Ide, *Cross-lingual sense determination: Can it work?*, Computers and the Humanities: Special issue on SENSEVAL, (2000) 34:223–234.

[14] U. Janfalk. *Linjär algebra*. Matematiska Institutionen, Linköpings Universitet. 2005.

[15] D. Jurafsky, J. H. Martin. *Speech and Language Processing, Second Edition*. Prentice Hall. 2009.

[16] T. Oetiker, H. Partl, I. Hyna, E. Schlegl. *The Not So Short Introduction to LaTeX 2ε*. 2006.

[17] P. Resnik , D. Yarowsky, *Distinguishing systems and distinguishing senses: new evaluation methods for Word Sense Disambiguation*, Natural Language Engineering, v.5 n.2, p.113-133, June 1999

[18] D. A. Spielman *Spectral Graph Theory.* Lecture notes. 2009.

[19] D. A. Spielman, S. -H. Teng, *Spectral partitioning works: Planar graphs and finite element meshes*, Linear Algebra and its Applications Volume 421, Issues 2-3, (2007), pp. 284-305.

[20] J. Tiedemann. *Recycling Translations.* Uppsala Universitet. 2003.