

Example-based Segmentation of Swedish Compounds in a Swedish–English bilingual corpus and the possibility of Evaluating Compound Links based on that Segmentation

Markus Saers

Abstract

In this paper an algorithm for segmenting Swedish compounds in a linking material is presented. The algorithm does the segmentation by looking at the example set by the corresponding English compound. The idea that this kind of segmentation can be used to evaluate the link between the two compounds is also tested. This would be possible because links where the algorithm cannot find suitable Swedish elements for each English element could be considered partial and thus unwanted. In order to be able to do this, the concept of silhouettes is introduced.

Master's Thesis in Computational Linguistics
Språkteknologiprogrammet (Language Engineering Programme)
Department of Linguistics and Philology, Uppsala University

Last changed: 2005-01-31

Supervisor: Anna Sågwall Hein, Uppsala University

Contents

1	Introduction.....	3
1.1	Statement of purpose	3
1.2	Outline of the paper.....	4
2	Background.....	5
2.1	Word alignment.....	5
2.2	Computational analysis of compounds.....	6
2.2.1	Lexical compound morphology	6
2.2.2	Statistical compound analysis.....	7
2.2.3	Phonotactical compound analysis	7
2.3	String similarities	7
3	Data.....	9
3.1	Training data	9
3.2	Test data – technical text.....	9
3.3	Test data – literary text.....	9
3.4	Summary	9
4	Method.....	10
4.1	Basic concepts and sub-algorithms	10
4.1.1	Extracting words from strings	10
4.1.2	Silhouettes.....	11
4.1.3	Complex silhouettes	11
4.1.4	Silhouette comparison	11
4.2	The algorithm.....	12
4.2.1	Clarifications	12
4.2.2	The dictionary.....	12
4.2.3	Segmenting the English compounds.....	12
4.2.4	Segmenting the Swedish compounds – an example.....	13
4.2.5	Deciding whether to keep the link or not.....	16
4.3	Implementation	16
4.3.1	The Database	16
4.3.2	Phase 1: Reading the dictionary into the database	16
4.3.3	Phase 2: Sorting out compound candidates.....	17
4.3.4	Phase 3: Segmenting English compounds	17
4.3.5	Phase 4: Creating silhouettes for Swedish compounds.....	17
4.3.6	Phase 5: Segmenting Swedish compounds	18
4.3.7	The report	18
5	Training.....	19
5.1	Training references.....	19
5.1.1	A priori reference.....	19
5.1.2	A posteriori reference.....	19
5.2	Optimizing weights.....	20
5.3	Optimizing operator assignment	20
6	Evaluation and Results	24
6.1	Evaluation methodology	24
6.1.1	Creating a reference	24
6.1.2	Evaluating segmentation.....	24
6.1.3	Evaluating operator assignment	24
6.2	Results from testing	25
6.2.1	Technical text.....	25
6.2.2	Literary text.....	25
7	Conclusion.....	26
7.1	Segmentation	26
7.2	Link evaluation	26
8	Outlook	27
8.1	Improving the algorithm	27
8.2	Further uses of silhouettes	27

1 Introduction

A great asset in corpus linguistics is a corpus and its translation – a bilingual corpus. As this depicts how one language is translated into another; it has a great potential in, among other things, machine translation. In order to be of use, the bilingual corpus has to be aligned. This is a process that aims at identifying segments of the two texts that are in some sort of equivalence relation to each other. To align sentences with each other is a fairly developed and reliable process which is widely used. In machine translation, the knowledge of which sentence to translate into which is not of that much use, since the chance of a new material consisting of known sentences is minimal. To increase the possibilities of the bilingual corpus, one would want to align smaller segments than sentences. The obvious choice for sub-sentence alignment is word alignment, since a sentence is often perceived as consisting of syntactic words. When the words of a bilingual corpus are aligned there will be a number of links connecting the words in one language to the words in the other, the process of word aligning is thus sometimes called linking. The result of the linking process is a translation relation between one language and the other, resembling that of a bilingual dictionary. This is one of the major reasons for doing word alignment: the possibility of automatically generated bilingual lexica. I will refer to these translation mappings as linking materials.

One problem when linking English to other Germanic languages is the difference in compounding. Where Germanic languages in general produce one word compounds, English does not. This means that the English unit to link to the Germanic compound is not trivially found. In a Swedish–English linking material there is bound to be partial links, that is: links that have a Swedish compound linked to a number of English words that overlap the English compound, but not completely. The word *magnetventil* ‘solenoid valve’ might be linked to *solenoid*, <preceding word> *solenoid*, *valve*, *valve* <following word> or *solenoid valve*. A large portion of the errors in automatically generated Swedish–English lexica are due to these partial links.

In this paper I will present an algorithm that identifies compound elements in Swedish by looking at the English words at the other end of a link. By following the links back to Swedish; the algorithm can find other compounds containing translations of the same English words. The parts of the Swedish compounds that overlap should prove to be a viable translation of the English word. If we have the English word *valve* in the compounds *solenoid valve* and *pressure valve*, the translations of these should be *magnetventil* and *tryckventil*. These both words have the overlapping part *ventil*, which is a viable translation of *valve*. A way to extract these overlapping parts is also presented as silhouettes.

The presented algorithm will do well on well formed links, but will have problems with ill formed links such as partial ones. This can be exploited as a way to clean out bad links from a linking material. A way to exploit this is also presented and tested.

1.1 Statement of purpose

The purpose of this paper is to investigate if Swedish compounds in a bilingual corpus can be segmented accurately using the trivial segmentation of free-form English compounds, and whether the success of the segmentation process can be used to evaluate the link between those words.

To do this I will use only the bilingual corpus, or rather; a linking material that has been automatically generated from a corpus. In this linking material there is only information on orthographic representation and frequency of the link types. This is the only information that will be provided to the algorithm, and this sets it apart from most previous work, which has relied on some sort of verified lexica in the traditional sense (see e.g. Koskeniemi 1983, Karlsson 1992, Dura 1998, Åberg 2003 and Olsson 2004).

To limit the work somewhat I intend to look only at compounds in the linking material. I will identify these as links connecting a Swedish single word unit to an English multi word unit. This is the only automatic way to identify possible compounds since it addresses the fundamental problem of linking English to other Germanic languages: the fact that compounding is a morphological phenomenon in Germanic languages whilst a syntactic phenomenon in English.

The segmentation will be carried out with the introduced notion of silhouettes, and the evaluation of the link will in fact be an evaluation of how well the Swedish compound was segmented by the example set by the English compound. The notion will be that Swedish compounds can be segmented in the same way as the linked English compounds, that is: if we have a link between “tryckvakt” and “pressure monitor” then “tryckvakt” has to be segmentable into something translating into “pressure” and something translating into “monitor”. In this case it is segmented into “tryck” and “vakt”. Silhouettes make it possible to isolate “tryck” from e.g. “tryckvakt” and “tryckmätare”, and if they are linked to English compounds containing a common word, that word would be likely to have a meaning similar to “tryck”, in this case such a word would be “pressure” (they should be linked to “pressure monitor” and “pressure gauge” respectively).

1.2 Outline of the paper

The paper will start with a description of the research areas of interest; word alignment, compound analysis and string similarity measurements (section 2). After that comes a brief description of the data used (section 3). Then, in section 4.1, there will be a description of the concepts that make the process possible. Silhouettes are introduced, and explained: how to extract, combine and compare them. In section 4.2 the algorithm is explained, mainly through a comprehensive example: we get to follow the link “tryckvakt : pressure monitor” through out the whole process. Then, in section 4.3, the implementation of the algorithm is explained, with database diagrams and examples of the final report.

At this stage the implementation needs to be trained. To train the segmentation process; an a priori reference is needed (section 5.1.1) to find out where to set the five threshold values that need to be trained (section 5.2). To train the link evaluation process; an a posteriori reference is needed (section 5.1.2) to set two threshold values (section 5.3).

The next step is to evaluate how well the system performs. This is described in section 6.1 (evaluation methodology), and the results from this testing is described in section 6.2. In section 6.3 there is a summary of the tests.

My conclusions come after that (section 7), and then an outlook onto the future (section 8).

2 Background

The NLP-areas of main interest to this paper are word alignment, compound analysis and string comparison. These three areas will be briefly discussed in section 2.1, 2.2 and 2.3 respectively.

2.1 Word alignment

To use bilingual corpora in NLP-tasks one must somehow know what parts are related in some way. This is done by aligning the texts. The first step would be to align sentences, and Gale & Church (1991) describes a well-used algorithm for doing this. Once the sentences are neatly aligned one can begin to align at the sub-sentence level – word aligning. The reasons for aligning words can range from large lexicon creation to term extraction. Objectives for word aligning will be covered later in this section. There are a number of different ideas on how to align words, but most of them use some form of co-occurrence measurement. This measures how often a certain word is found on one side of the translation given that another certain word is found on the other. The Swedish word “ventil” would have a high co-occurrence value with the English word “valve”, as they have the same meaning, and is likely to be present on their respective side of the translation of a sentence dealing with valves. In other words they co-occur. There are a number of statistical co-occurrence measurements including t-score, Dice’s coefficient and mutual information.

When two corpora are aligned; there will be a number of mappings, or links, connecting segments in one language to segments in the other language. In this paper; these links will be presented as the two segments separated by a colon, like this:

ventil : valve

This should be read like this: “There seems to be some statistical certainty that the word ‘ventil’ in the source language can be substituted with the word ‘valve’ when translating from the source language into the target language.” That is; it should be read as a relation from source language to target language (going from left to right) and there is a degree of uncertainty as to the correctness of the link.

One can view the segments being linked as tokens. The links between these tokens will be called link instances. When one creates a word list from a corpus, all tokens that are equal are condensed into a type (usually coupled with a frequency count). In a similar way a translation mapping can be created from all link instances. This will result in a list of link types, a structure around which a translation lexicon can be built (Tiedemann 1999). Depending on the purpose of the alignment one can focus on either link instances or link types. If the objective is to mark up an entire corpus (e.g. for educational purposes) the focus would, naturally, be on link instances – finding correspondences to as much as possible. If the objective is to create a bilingual lexicon (e.g. for machine translation) the focus would be on link types – making as good lexicon-entries as possible. Term extraction would then be similar to lexicon creation, but with the emphasis on precision rather than a balance between precision and recall.

The problems that have to be overcome to align words correctly are numerous. To start with, different languages have different structures. What one language uses inflections to describe, another might use words. This shatters any hopes of finding a one-to-one correspondence between the two languages. This is especially true for Germanic languages paired with English or non-Germanic languages, since Germanic languages compound into orthographic words, rather than just writing words after each other. This yields a lot of one-to-many links e.g.:

varvtalsgivare : engine speed sensor

Using a co-occurrence measurement, this would make “engine”, “speed” and “sensor” share the same co-occurrence value to “varvtalsgivare”, in effect dividing the value by three. All too often, this leads to dead wrong or partial links. Instead of the one above we might get anyone of (but not limited to) these:

varvtalsgivare : engine speed
∅ : sensor

∅ : engine
varvtalsgivare : speed sensor

∅ : engine
varvtalsgivare : speed
∅ : sensor

Note that the zero-links might or might not appear depending on the objective of the alignment. There have been a number of suggestions in the literature on how to handle these one-to-many links and the partial linkage they spawn.

Jones & Alexa (1997) suggests trying all n-grams of sizes ranging from 1 to 5. Even though this is a rather naïve method it seems to work well in conjunction with Dice's coefficient on an English–German bitext; although no results are presented the authors seem to be hopeful.

Tiedemann (1999) uses a dynamic linking strategy that establishes only the most certain links. All established links are removed, and the process is repeated until everything has been linked. This strategy uses the principles of “baby-steps”, which are developed from Kay's (1980) proposal on how to approach machine translation:

The keynote will be modesty. At each stage, we will do only what we know we can do reliably. Little steps for little feet!

(Kay 1980:13)

Tiedemann (2003) improves the previous results by looking at clues. The idea is to use all knowledge of the processed text to better the linking process. Therefore, things like part of speech-tags, as well as language specific user-declared information, are considered and used in the linking process. This is, of course, also consistent with the principles of “baby-steps”.

Even with these conscious efforts to reduce partial linkage, this is still a problem. Ahrenberg et al (1998) has reported partial links specifically in the results from a word alignment between Swedish and English. In their tests on literary text, between 2.5 and 5.6 % of the errors were partial links. For technical text the partial linkage ranged from 2.1 % to 16.8 %. A higher precision/recall generally seems to give a higher amount of partial links, which might be due to the fact that partial links get a reduced score instead of being counted as wrong. This means that there is a lot to be gained if the problem with partial linkage can be successfully addressed.

2.2 Computational analysis of compounds

Unlike English, the Swedish compounds are written as one orthographic word. This makes compound analysis a morphological problem in Swedish rather than a syntactical. Traditionally, computational inflection morphology deals with a very limited number of affixes, and some rules as to how these are affixed onto words. There are three ways to deal with the problem of compound analysis in Swedish and other Germanic languages: lexical morphology, statistical and phonetical compound analysis. These will be discussed in section 2.2.1, 2.2.2 and 2.2.3 respectively.

2.2.1 Lexical compound morphology

In the same way that traditional inflection morphology uses a list of affixes; the lexical compound morphology would use a lexicon as a list of possible elements. And in the same way that rules are used to determine how the affixes are “glued” onto the inflected words; rules are used to determine how elements are compounded into compounds.

The problem with this method is that it overgenerates, since it generates all possible solutions. These collections of possible solutions are called cohorts (Karlsson 1992). The word *kulturkrock* ‘cultural clash’ could possibly be made up of different combinations of the words *kul* ‘fun/ball’ (as in ball-point pen), *kult* ‘sect’, *kultur* ‘culture’, *tur* ‘luck/turn/short trip’, *turk* ‘Turk’, *ur* ‘watch/primordial’, *krock* ‘clash/crash/collision’ and *rock* ‘rock music/coat’ (example from Karlsson 1992.) Choosing which of these readings to use is the real challenge for a compound analyzer.

Karlsson (1992) uses two preference rules in his SWETWOL system (SWEDISH TWO LEVEL MODEL – an implementation of Koskeniemi's (1983) TWOL – TWO LEVEL MODEL – for Swedish morphology); Compound Elimination Principle and Derivative Elimination Principle. These mean that a simple non-derived reading is chosen over a complex derived reading. If a token is found in the lexicon no further analysis is carried out. Thus *kultur|krock* would be the preferred reading of the above example since it has the fewest elements (Compound Elimination Principle).

A similar approach is taken by Dura (1998) and Olsson (2004). Dura adds weights (based largely on the principles put forward by Karlsson, 1992) to determine which analysis to choose. The paper describes a fast implementation for on-line applications.

Olsson (2004) describes (the Swedish part of) a proof-reading system called Scarrie, a Scandinavian version of the Corrie prototype. This formalism relies heavily on lexical information – each word is tagged with a feature that determines its “compoundability”. There are rules as how these lexical items can be combined, and the Compound Elimination Principle is enforced so that compounds with fewer elements are preferred.

Another paper that addresses the problem of choosing the right segmentation is Åberg (2003). Using the compound analysis of Swe.ucp (Sågvall Hein, 1983) as input, Åberg's implementation chooses the right analysis by enforcing the Compound Elimination Principle, but also by looking at the length of the elements. If it can not choose one analysis it will pass that word on for manual segmentation. The implementation has been tested on technical text. In this text the implementation managed to choose the correct analysis for 8,241 out of 8,662 words, which has to be regarded as good. Most of the errors were caused by the grammar (Swe.ucp); the implementation only made 86 bad choices.

2.2.2 Statistical compound analysis

This is the notion that compounds could be split by rules learned from corpora. In Koehn & Knight (2003) an algorithm for learning splitting rules from a parallel corpus is introduced. This algorithm works by splitting words into other words found in the same corpus. For example, German “aktionsplan” (which translates into “action plan”) could consist of the words “aktionsplan”, “aktion”, “aktionen”, “akt”, “ion” and “plan”, since all of these words have been found individually in the corpus. The correct combination of elements is arrived at by looking into a sentence aligned, parallel English–German text. Each element combination is ranked by how well it translates into English. If we are testing the elements “akt”, “ion” and “plan” we would expect to find translations of these words into English in the corresponding English sentence. In doing this a one-to-one correspondence between German compound elements and English content words were enforced. Needless to say this approach is dependent on a translation lexicon. To obtain this lexicon they used the Giza toolkit (Al-Onaizan et al, 1999). Using the parallel corpus and the extracted translation lexicon, they were able to achieve an accuracy of 98.6 % measured against a manually annotated gold standard. The paper also mentions a way of taking part of speech tags into account, which boosts the accuracy up to 99.1 %.

2.2.3 Phonotactical compound analysis

One way to do morphology is to look at the surface structure of words, and create rules constraining the way the morphemes may look. This is motivated by an intuition that the phonotax of spoken words is reflected in the orthographic representation of written words, and the assumption that a language has constraints on how a phonotactically well formed word may look. Brodda (1979) presents an algorithm for segmenting inflected Swedish words with this method. By defining lists of inflections and consonant clusters (a string of consonants surrounded by vowels or word-spacing characters) that may appear in different situations (morpheme initial, morpheme final or morpheme medial), words can be correctly segmented. To handle compounds there is a hierarchy of medial consonant clusters that range from “certainly containing a word divider” to “may contain a word divider”. The consonant cluster “ntst”, for example, can never occur inside a Swedish word – it has to contain a word divider (e.g. *kant|sten*, ‘corner stone’). This is (according to the author) a fairly accurate model for native Swedish words, but has inherent problems with imported words. The algorithm expects words to follow Swedish phonotax, but imported words can not be expected to do that – thus there are problems with non-native words.

2.3 String similarities

String similarity is a metric of how alike two strings are. The basic computational way to compare strings is to deem them either exactly identical or not. This is not a fine enough tool when one is dealing with natural language. In an NLP-application the need might arise to know how similar two strings are even when they are not identical. This is the ability that string similarity metrics provide.

The most basic way to calculate string similarity is editing distance (*eddist*). This is the notion of counting how many insertions, deletions and substitutions are required to rewrite one string into another. This actually calculates the difference between the two strings, the similarity measurement derived from editing distance is called *Longest Common Subsequence* (LCSS). The LCSS is defined as the length of the longer of the two strings minus the editing distance between them:

$$LCSS = \max(\text{length}(w_1), \text{length}(w_2)) - \text{eddist}(w_1, w_2)$$

This is often implemented using dynamic programming and achieves speeds of $O(nm)$ (where n and m is the length of the two strings respectively). The dynamic programming approach creates a table like that of figure 1, where the LCSS can be obtained by looking at the most bottom-right field.

	s	e	e		e	x	a	m	p	l	e
s	1	1	1	1	1	1	1	1	1	1	1
e	1	2	2	2	2	2	2	2	2	2	2
	1	2	2	3	3	3	3	3	3	3	3
e	1	2	3	3	4	4	4	4	4	4	4
x	1	2	3	3	4	5	5	5	5	5	5
e	1	2	3	3	4	5	5	5	5	5	5
m	1	2	3	3	4	5	5	6	6	6	6
p	1	2	3	3	4	5	5	6	7	7	7
e	1	2	3	3	4	5	5	6	7	7	8
l	1	2	3	3	4	5	5	6	7	8	8

Figure 1: Longest Common Subsequence calculated by dynamic programming, example from Tiedemann (1999).

The LCSS is often normalized as the Longest Common Subsequence Ratio (LCSR) by dividing the LCSS by the length of the longer of the two words:

$$LCSR = \frac{LCSS}{\max(\text{length}(w_1), \text{length}(w_2))}$$

In the example above we would have an LCSS of 8, and a longest word length of 11 ($\max(10, 11)$), which would give us an LCSR of:

$$\frac{8}{11} \approx 0,72$$

That is: the words have 72 % of the characters in common.

3 Data

I used three different corpora: one for training and two for testing. All three corpora have been aligned by Jörg Tiedemann using the Uppsala Word Aligner – uwa (Tiedemann 1999). The alignment process produced a number of type links. Among these type links those connecting a Swedish single word unit (swu) with an English multi word unit (mwu) were selected for further processing, as the theory is that these links constitute candidate compounds. The three corpora are described in sections 3.1–3.3. Section 3.4 contains a summary of the three corpora.

3.1 Training data

The training data consists of user manuals for Microsoft Access. The corpus is a part of the PLUG-corpus (Sågval Hein 1999), and contains about 163,000 words. When uwa was applied; a linking material containing 4,759 type links was obtained. These links link 2,872 English units to 3,185 Swedish units. Of these 4,759 type links, 838 fit my criterion for being considered as compounded. These links link 670 English multi word units to 687 Swedish single word units.

3.2 Test data – technical text

The test data consists of the MATS-corpus (Sågval Hein et al 2002), a corpus of about 100,000 words. After uwa was applied; a linking material containing 8,708 type links was obtained. The links connect 7,244 Swedish units to 5,841 English units. Of these links, 1,956 fit my criterion for being considered as compounded. These links connect 1,807 Swedish single word units to 1,388 English multi word units.

3.3 Test data – literary text

For testing on literary text I used a translation of “Viking P. for the Jewish Publication Society of America” by Saul Bellows which is a part of the PLUG-corpus (Sågval Hein 1999). It contains about 132,000 words. uwa produced a linking material from it containing 8,028 type links, connecting 5,645 English units to 6,457 Swedish units. Of these, 450 links fit my criterion for being considered as compounded; connecting 421 English multi word units to 431 Swedish single word units.

3.4 Summary

A comprehensive table of the three corpora in numbers can be found in table 1. Most noteworthy is the fact that the literary text contains so few possible compounds (less than 10 % of the types are possible compounds). This would further substantiate Åberg’s (2004) conclusion that compounds are very common in technical texts.

	Training data (technical)			Technical test data			Literary test data		
	Swe	Links	Eng	Swe	Links	Eng	Swe	Links	Eng
Types	3,185	4,759	2,872	7,244	8,708	5,841	6,457	8,028	5,645
Compounds	687	838	670	1,807	1,956	1,388	431	450	421
Compounds/type (%)	21.6	17.6	30.4	24.9	22.5	23.8	6.7	5.6	7.5

Table 1: Comparison of the corpora in number of types, number of possible compounds and percentage of compounds.

4 Method

The algorithm presented here addresses the problem of partial compound links in an automatically generated English–Swedish lexicon. It does this by segmenting the one-word Swedish compounds after an example set by the English free form compounds. If the link between “tryckvakt” and “pressure monitor” can produce a plausible segmentation of “tryckvakt” into something likely representing “pressure” and something likely representing “monitor”, then the link is valid. This should fail if there is something missing or something extra present. For example, if we have a link between “pinjonglagret” (the pinion bearing) and “the pinion”, then one should be able to conclude that the English word do not represent the whole of “pinjonglagret”, and therefore the link is flawed. This should work in the other direction as well. The link between “adr-bilar” (adr vehicles) and “adr vehicles fitting” should be recognized as flawed since there is a Swedish representation of “adr” and “vehicle” but not of “fitting”. This is my hypothesis, and here I intend to present a method for testing it.

4.1 Basic concepts and sub-algorithms

In the algorithm described below I will use a few notions, concepts and minor algorithms that I intend to describe here. The notion of silhouettes, how to extract, represent, combine and compare them, is introduced and explained.

4.1.1 Extracting words from strings

When extracting words from strings I will use an algorithm of my own device similar to that of LCSS. Instead of generating a sequence of characters this generates a set of substrings that are common to the compared strings. It generates a table similar to that of LCSS, but, instead of adding up the similarity so far, a “1” is inserted when the two considered characters are equal, and a “0” when they are not. See figure 2 for an example.

	s	e	e		e	x	a	m	p	l	e
s	1	0	0	0	0	0	0	0	0	0	0
e	0	1	1	0	1	0	0	0	0	0	1
	0	0	0	1	0	0	0	0	0	0	0
e	0	1	1	0	1	0	0	0	0	0	1
x	0	0	0	0	0	1	0	0	0	0	0
e	0	1	1	0	1	0	0	0	0	0	1
m	0	0	0	0	0	0	0	1	0	0	0
p	0	0	0	0	0	0	0	0	1	0	0
e	0	1	1	0	1	0	0	0	0	0	1
l	0	0	0	0	0	0	0	0	0	1	0

Figure 2: Strings extracted by dynamic programming similar to that of LCSS.

Here, every top–down, left–right diagonal represents a common substring (the darker ones are longer). The set of substrings in the example consists of {“se”, “e”, “e ex”, “mp”, “l”}. This gives us all unique substrings, it does not however make any difference between the “e”:s in “see” and the ones in “example”. One way to differentiate them is to create a *silhouette* describing where the string was found. This is done by following the diagonal all the way (and extend with zeros when necessary) along the word that the silhouette is made for. Assuming we are making silhouettes for “see example” we would end up with this set: {1100000000, 0100000000, 0010000000, 0011110000, 0000100000, 0000001100, 0000000010, 0000000001}, corresponding to the strings “se”, “e”, “e”, “e ex”, “e”, “mp”, “l” and “e”.

This may not be a very good way of comparing across languages, but that is not how it will be used. Instead it will be used to compare words of the same language to find, for example, elements of a compound. If we do a comparison between *tryckvakt* ‘pressure monitor’ and *nivåvakt* ‘level monitor’ we get the table in figure 3, and correspondingly a set of silhouettes looking like this (for *tryckvakt*): {100000000, 000001000, 000001111} corresponding to the strings “t”, “v” and “vakt” (which happens to be one of the compound elements).

	t	r	y	c	k	v	a	k	t
n	0	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	0	0	0
v	0	0	0	0	0	1	0	0	0
å	0	0	0	0	0	0	0	0	0
v	0	0	0	0	0	1	0	0	0
a	0	0	0	0	0	0	1	0	0
k	0	0	0	0	1	0	0	1	0
t	1	0	0	0	0	0	0	0	1

Figure 3: Silhouettes between “tryckvakt” and “nivåvakt”.

4.1.2 Silhouettes

A silhouette is the notion of a cardboard cut-out – that is; it shows part of a string and hides the rest. It is extracted using the algorithm above and thus modelled as a string of binary values of the same length as the string it is made for. I will use the expression “a silhouette *over* a string” to refer to the relation between the silhouette and the original string. For example:

```
tryckvakt (a string)
111110000 (a silhouette over tryckvakt denoting “tryck”)
```

When clarity demands shortening of a silhouette, I will replace zeros with “•” and ones with the character denoted. The above example would then be written as “tryck••••”.

4.1.3 Complex silhouettes

Several silhouettes can be combined to form a new, complex silhouette. The operator used is exclusive disjunction (XOR). This is the logical equivalent of the notion “A or B but not both”. Normally a logical operator takes two arguments. I will instead use a multi-argument XOR-operator. This means that it is the equivalent of the notion “any of A_1, A_2, \dots, A_n but only one of them”. As the operator is used to combine silhouettes that are made up of ones and zeros the operator works by adding all values of each column (made by putting the silhouettes on top of each other). The sum is then interpreted as zero if it is not equal to one. The ones and zeros now denote shown and hidden respectively.

```
111110000 tryck
000001111 vakt
===== XOR
111111111 tryckvakt
111111111 interpretation of this complex silhouette.
```

```
11111111100000000 överfalls
00000000111111110 sventile
000000000000000011 en
===== XOR
11111111211111121 a complex silhouette
11111111011111101 interpretation denoting “överfall•ventil•n”.
```

4.1.4 Silhouette comparison

In the algorithm an important step is comparing how similar two silhouettes are, even across languages. I do this by comparing coverage. For example: I wish to see how similar “tryck••••” (from “tryckvakt”) is to “pressure••••••••” (from “pressure monitor”). The first silhouette covers 5 out of 9 characters of the string, the second covers 8 out of 16 characters. The first step is to find a common denominator. Since I only want to use integers I will keep the cover values in fraction form:

$$\text{For “tryck”}: \frac{5}{9} \times \frac{16}{16} = \frac{80}{144} \quad \text{and for “pressure”}: \frac{8}{16} \times \frac{9}{9} = \frac{72}{144}$$

The difference between these two is then calculated:

$$\left| \frac{80}{144} - \frac{72}{144} \right| = \frac{8}{144} \approx 5.56\%$$

The difference between two complex silhouettes is calculated by summing up numerator and denominator separately for all simple silhouettes making up the complex silhouette, and for the complex silhouette's surface form. In other word; the average difference is calculated.

```

tryck..... 5/9      pressure..... 8/16
.....vakt 4/9      .....monitor 7/16
=====
tryckvakt 9/9      pressure·monitor 15/16

```

The difference would then be calculated as such:

$$\frac{|(5 \times 16) - (8 \times 9)| + |(4 \times 16) - (7 \times 9)| + |(9 \times 16) - (15 \times 9)|}{(9 \times 16) + (9 \times 16) + (9 \times 16)} = \frac{8 + 1 + 9}{144 + 144 + 144} = \frac{18}{432} \approx 4.17\%$$

4.2 The algorithm

Here I intend to show how Swedish compounds can be segmented in the same way as their corresponding English compounds are trivially segmented at white space.

4.2.1 Clarifications

As this is a rather coarse algorithm some words are used in a slightly different (looser or stricter) sense than usual. A candidate “compound” here is a link between a Swedish single word unit (swu) and an English multi word unit (mwu). The fact that there are English swu compounds is not acknowledged – nor the fact that there are swu–mwu links that are not compounds.

The only white space character is space (although it would be interesting to include some more such as hyphenation characters).

4.2.2 The dictionary

The dictionary is generated from the respective corpora using Uppsala Word Aligner (uwa) in static linking mode. Since compounds are of main interest, a new dictionary is created only containing links that have a Swedish swu entry and an English mwu entry (as the notion is that Swedish compounds are written as one word and English as several).

4.2.3 Segmenting the English compounds

The next step is to segment the English compounds. As I have mentioned earlier this is trivial. The English mwus are split at white space and the elements are stored in a new word list which is then linked to the word list of English compounds. Associated to the links are silhouettes of the elements over the compound. White space will be left out, so the silhouettes will not cover the whole compound. For example, the compound “pressure monitor” would be represented by two elements and their silhouettes over the original compound; as such:

```

pressure.....
.....monitor
=====
pressure·monitor

```

4.2.4 Segmenting the Swedish compounds – an example

The segmentation process starts with a Swedish compound to be segmented.

tryckvakt

All English entries linked to that compound are looked up, and treated one at a time.

pressure monitor

The next step is to look up all the elements included in the English compound, and then look up all the Swedish compounds linked to an English compound that includes that element. The single Swedish words that might be linked to the English compound as a single word are also included. So, in the example below; if “pressure” is linked to “tryck” as single words, then “tryck” would be included in the list of Swedish compounds (or rather segments) to be processed.

pressure -> {*pressure monitor, pressure valve, pressure gauge, pressure*}
-> {*tryckvakt, tryckventil, tryckmätare, tryck*}

monitor -> {*pressure monitor, steam monitor*} -> {*tryckvakt, ångvakt*}

We now have a list of all the Swedish compounds that are supposed to have an English translation containing the current English compound element. Next, the list of Swedish compounds is compared to the original Swedish compound (the one we are examining) and all substrings are collected (as a set of silhouettes over the original compound). These silhouettes denote possible elements in the Swedish compound. Of course, comparing a word to itself is meaningless, and therefore no word is compared to itself.

pressure:
tryckvakt & tryckventil -> {*tryckv•••, ••••••k•, ••••••t, ••••••••*}
tryckvakt & tryckmätare -> {*tryck••••, ••••••a•, ••••••k•, ••••••t, ••••••••*}
tryckvakt & tryck -> {*tryck••••, ••••••k•, ••••••t, ••••••••*}

Set of silhouettes to denote pressure:
{*tryckv•••, tryck••••, ••••••k•, ••••••t, ••••••a•, ••••••••*}

monitor:
tryckvakt & ångvakt -> {*t••••••••, ••••k••••, ••••vakt, ••••••••*}

Set of silhouettes to denote monitor:
{*t••••••~•, ••••k••••, ••••vakt, ••••••~•*}

Should there be nothing to compare with, and thus yield an empty set of candidate elements, an empty silhouette is inserted with a frequency of 1.

To get an idea of which of these are most likely to yield a good segmentation, a merit value is calculated. This value is a weighted average of frequency, cover ratio and likeness to corresponding English element.

The frequency count is normalized over the set representing one English element. That is; the normalized frequency total of e.g. “pressure” is 1. Assuming frequencies of 1 for “tryckvakt” and “tryckmätare”, the normalized frequencies for the set above would be calculated as such:

First the frequency count of each silhouette is calculated depending on how many words it appear in.

tryckv•••: gets 1 from “tryckventil”, total frequency: 1.
tryck••••: gets 1 from “tryckmätare” and 1 from “tryck”, total frequency: 2.
••••••k•: gets 1 from “tryckventil”, 1 from “tryckmätare” and 1 from “tryck”, total frequency: 3.
••••••t: gets 1 from “tryckventil”, 1 from “tryckmätare” and 1 from “tryck”, total frequency: 3.
••••••a•: gets 1 from “tryckmätare”, total frequency: 1.
••••••~•: gets 1 from “tryckventil”, 1 from “tryckmätare” and 1 from “tryck”, total frequency: 3.

Then the frequency count of each silhouette is divided by the total frequency count (13).

tryckv•••: $1/13 = 0.08$
tryck••••: $2/13 = 0.15$
••••••k•: $3/13 = 0.23$
••~•••t: $3/13 = 0.23$
••••••a•: $1/13 = 0.08$
••••••~•: $3/13 = 0.23$

The sum is now one ($0.08 + 0.15 + 0.23 + 0.23 + 0.08 + 0.23 = 1.00$).

The cover ratio is simply the number of characters covered divided by the length of the silhouette. For the set of “pressure” this would be calculated as such (rounded to two decimals):

$$\begin{aligned} \text{tryckv}\bullet\bullet\bullet: & 6/9 = 0.67 \\ \text{tryck}\bullet\bullet\bullet\bullet: & 5/9 = 0.56 \\ \bullet\bullet\bullet\bullet\bullet k\bullet: & 1/9 = 0.11 \\ \bullet\bullet\bullet\bullet\bullet t: & 1/9 = 0.11 \\ \bullet\bullet\bullet\bullet\bullet a\bullet\bullet: & 1/9 = 0.11 \\ \bullet\bullet\bullet\bullet\bullet\bullet: & 0/9 = 0.00 \end{aligned}$$

The likeness is calculated as one minus difference. For the set representing “pressure” this would be calculated as such (rounded to two decimals):

$$\begin{aligned} \text{pressure}\bullet\bullet\bullet\bullet\bullet\bullet \& \text{tryckv}\bullet\bullet\bullet \rightarrow 1 - |(72/144) - (96/144)| = 1 - (24/144) = 0.83 \\ \text{pressure}\bullet\bullet\bullet\bullet\bullet\bullet \& \text{tryck}\bullet\bullet\bullet\bullet \rightarrow 1 - |(72/144) - (80/144)| = 1 - (8/144) = 0.94 \\ \text{pressure}\bullet\bullet\bullet\bullet\bullet\bullet \& \bullet\bullet\bullet\bullet k\bullet\bullet\bullet \rightarrow 1 - |(72/144) - (16/144)| = 1 - (56/144) = 0.61 \\ \text{pressure}\bullet\bullet\bullet\bullet\bullet\bullet \& \bullet\bullet\bullet\bullet\bullet t \rightarrow 1 - |(72/144) - (16/144)| = 1 - (56/144) = 0.61 \\ \text{pressure}\bullet\bullet\bullet\bullet\bullet\bullet \& \bullet\bullet\bullet\bullet\bullet a\bullet\bullet \rightarrow 1 - |(72/144) - (16/144)| = 1 - (56/144) = 0.61 \\ \text{pressure}\bullet\bullet\bullet\bullet\bullet\bullet \& \bullet\bullet\bullet\bullet\bullet\bullet \rightarrow 1 - |(72/144) - (0/144)| = 1 - (72/144) = 0.50 \end{aligned}$$

As I mentioned these three values are combined as a weighted average. In this demonstration I will instead use an ordinary average, leaving the weighting for the training section where it really belongs. Calculating an average of these three values for the set associated with “pressure” we get the following merit values:

$$\begin{aligned} \text{tryckv}\bullet\bullet\bullet: & (0.08 + 0.67 + 0.83) / 3 = 0.53 \\ \text{tryck}\bullet\bullet\bullet\bullet: & (0.15 + 0.56 + 0.94) / 3 = 0.55 \\ \bullet\bullet\bullet\bullet\bullet k\bullet: & (0.23 + 0.11 + 0.61) / 3 = 0.32 \\ \bullet\bullet\bullet\bullet\bullet t: & (0.23 + 0.11 + 0.61) / 3 = 0.32 \\ \bullet\bullet\bullet\bullet\bullet a\bullet\bullet: & (0.08 + 0.11 + 0.61) / 3 = 0.27 \\ \bullet\bullet\bullet\bullet\bullet\bullet: & (0.23 + 0.00 + 0.50) / 3 = 0.24 \end{aligned}$$

For the set of silhouettes to represent “monitor”, the merit value would be calculated like this (just trust me on the numbers):

$$\begin{aligned} t\bullet\bullet\bullet\bullet\bullet\bullet: & (0.25 + 0.11 + 0.67) / 3 = 0.34 \\ \bullet\bullet\bullet k\bullet\bullet\bullet\bullet: & (0.25 + 0.11 + 0.67) / 3 = 0.34 \\ \bullet\bullet\bullet\bullet vakt: & (0.25 + 0.44 + 0.99) / 3 = 0.56 \\ \bullet\bullet\bullet\bullet\bullet\bullet: & (0.25 + 0.00 + 0.56) / 3 = 0.27 \end{aligned}$$

Now it is time to combine all possible combinations of candidate elements for “pressure” and “monitor”. The resulting complex silhouettes then represents different ways of segmenting “tryckvakt” based on the example set by “pressure monitor”. A new merit value is calculated for the complex silhouette, and the best is chosen. This merit value is calculated as a weighted average between an element merit summary and the cover ratio of the complex silhouette.

The merit summary of a complex silhouette is calculated by assigning the merit of each silhouette to all characters covered by it and zero to all characters not covered by it. The complex silhouette then gets a similar assignment to its characters, keeping the merit values from the silhouette covering it. Remember the exclusiveness of the combination operator, as it now resolves conflicts – if a character is covered by two silhouettes it is in fact not covered at all. For the combination of “tryckv” and “vakt” we get the schema depicted in figure 4.

t	r	y	c	k	v	•	•	•
0.53	0.53	0.53	0.53	0.53	0.53	0.00	0.00	0.00
•	•	•	•	•	v	a	k	t
0.00	0.00	0.00	0.00	0.00	0.56	0.56	0.56	0.56

t	r	y	c	k	•	a	k	t	$merit = \frac{(0.53 \times 5) + (0.56 \times 3)}{9} = 0.48$
0.53	0.53	0.53	0.53	0.53	0.00	0.56	0.56	0.56	

Figure 4: Merit summary of a complex silhouette containing the silhouettes of “tryckv” and “vakt” merited to 0.53 and 0.56 respectively.

The cover ratio for complex silhouettes is exactly the same as that of simple silhouettes. And, as with the weighted average of simple silhouettes, I will rely on the regular average of the merit summary and the cover ratio for complex silhouettes as well. For the example in figure 4 that would be:

$$merit = \frac{\left(\frac{(0.53 \times 5) + (0.56 \times 3)}{9}\right) + \left(\frac{8}{9}\right)}{2} \approx \frac{0.48 + 0.89}{2} = 0.685$$

Going through the same procedure for all possible combinations we get the following list (on the format complex silhouette {comprising simple silhouettes}: merit value):

•ryckv••• {tryckv•••, t••••••••}: 0.425
 tryc•v••• {tryckv•••, •••k••••}: 0.425
 tryck•akt {tryckv•••, ••••vakt}: 0.685
 tryckv••• {tryckv•••, ••••••••}: 0.510

•ryck•••• {tryck••••, t••••••••}: 0.344
 tryc•••• {tryck••••, •••k••••}: 0.344
 tryckvakt {tryck••••, ••••vakt}: 0.777
 tryck•••• {tryck••••, ••••••••}: 0.431

t••••••k• {••••••k•, t••••••••}: 0.148
 •••k••k• {••••••k•, •••k••••}: 0.148
 ••••va•t {••••••k•, ••••vakt}: 0.260
 ••••••k• {••••••k•, ••••••••}: 0.073

t••••••t {••••••t, t••••••••}: 0.148
 •••k••t {••••••t, •••k••••}: 0.148
 ••••vak• {••••••t, ••••vakt}: 0.260
 ••••••t {••••••t, ••••••••}: 0.073

t•••••a•• {•••••a••, t••••••••}: 0.145
 •••k•a•• {•••••a••, •••k••••}: 0.145
 ••••v•kt {•••••a••, ••••vakt}: 0.260
 •••••a•• {••~••a••, ••••••••}: 0.071

t•••••••• {••••••••, t••••••••}: 0.074
 •••k•••• {••••••••, •••k••••}: 0.074
 ••••vakt {••••••••, ••••vakt}: 0.433
 •••••••• {••••~••••, ••••••~••}: 0.000

Now all we have to do is select the best complex silhouette, and have it represent the English compound.

tryckvakt {tryck••••, ••••vakt}: 0.777

This complex silhouette is now subject of even more merit adjustment to ease the decision of whether it is indeed a good link that we want to keep or not. This is done by premiering complex silhouettes not containing empty silhouettes. The notion is that if the best available complex silhouette contains empty silhouettes then this is in fact a partial link. Take this example:

```
adr•••• : adr
•••bil : vehicle
•••••• : fitting
=====
adr-bil : adr vehicle fitting
```

The empty silhouette corresponding to “fitting” is definitely a clue that the link is in fact partial, and thus the merit score should be lowered. The merit modifier is calculated as such:

$$merit_{mod} = \frac{element_count - empty_element_count}{element_count}$$

And for the example above it would be:

$$\text{merit_mod} = \frac{3-1}{3} = \frac{2}{3} \approx 0.667$$

This means that the link in question is only assigned two thirds of its merit value for the future.

4.2.5 Deciding whether to keep the link or not

When the Swedish part of the link has been segmented by the example set by the English part of it, it is time to decide whether the link should be kept, discarded or sent on for manual checking. The choice of action is determined solely on merit value. The three actions possible are represented by “operators” borrowed from Perl/C which are inserted instead of the colon in the link representation. The operators are “==” for keep, “!=” for discard and “?=” for “pass on to manual check”. We thus have links looking like this:

```
setup directory != distribution
          adr-bil ?= adr vehicle fitting
pressure monitor == tryckvakt
```

The training process will yield acceptable threshold merit values for operator assignment, see section 5 Training.

4.3 Implementation

The implementation is divided into five phases, using a relational database to store information between the phases. The phases are: *reading the dictionary into the database*, *sorting out compound candidates*, *segmenting English compounds*, *creating silhouettes for the Swedish compounds* and *segmenting Swedish compounds*. I will describe the different phases one at a time (4.3.2–6), as well as the database in general (4.3.1) and the resulting report (4.3.7).

4.3.1 The Database

The database system I use is a freeware SQL database called MySQL. To model a bilingual dictionary I use word list tables to store monolingual types, and link list tables to link monolingual types to each other. This is a very flexible core that can be called upon to hold much more information than it does today (now it only holds an id-number, an orthographic representation and a frequency count for the words, and id-number, source–target id:s and frequency counts for the links). There is also a silhouette list that links for example English compounds to English compound elements (this holds an id-number, source–target id:s, silhouette and frequency). For convenience, the silhouette list also holds derivable information like coverage and difference. When illustrating the database I will use the convention that a box is a table, a diamond is a table that acts as a relation and a line is a connection between related tables. Furthermore, I will use a dashed line with an arrow to mean roughly “derived from”.

4.3.2 Phase 1: Reading the dictionary into the database

This phase does exactly what it sounds like. UWA produces lexicons like text files, and they have to be read and added correctly to the database. The database after phase one is illustrated in figure 5.

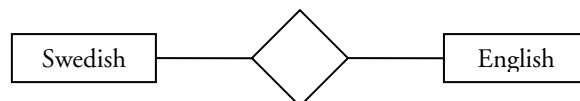


Figure 5: The database after phase 1.

4.3.3 Phase 2: Sorting out compound candidates

To sort out compound candidates the coarse definition “every link that has an *swu* on the Swedish side and an *mwu* on the English” is used. What this phase does is simply to go through all the links generated in phase one and sort out the ones that fit the description. The database after phase two is illustrated in figure 6.

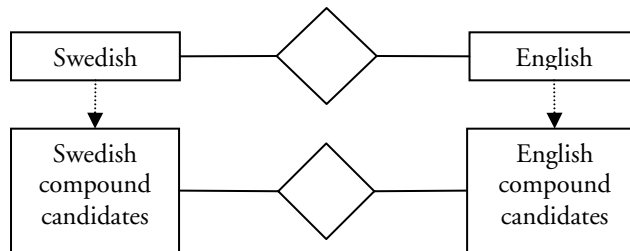


Figure 6: The database after phase 2.

4.3.4 Phase 3: Segmenting English compounds

At this phase each English candidate compound is split at white space and the elements are stored in a word list and linked with a silhouette to the original compound candidate. The database after phase 3 is illustrated in figure 7.

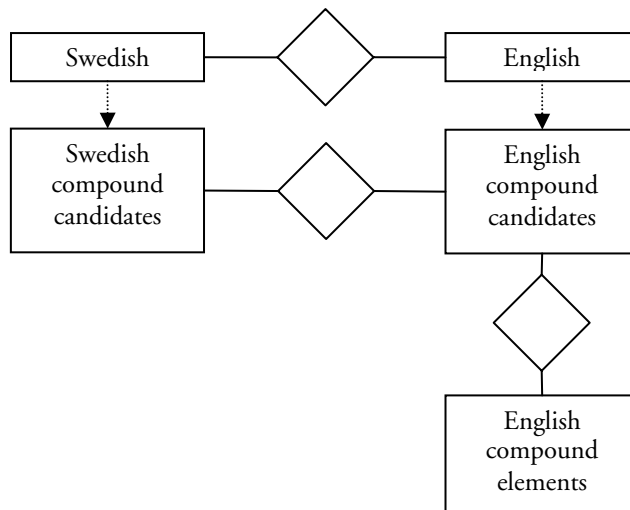


Figure 7: The database after phase 3.

4.3.5 Phase 4: Creating silhouettes for Swedish compounds

In this phase the goal is to establish a silhouette link between English compound elements and the Swedish compounds likely to include a translation of that element. This is done in the following steps:

1. For each Swedish compound: collect all linked English compounds.
2. For all those compounds: collect all elements they contain.
3. For all those elements: collect all English compounds that include the element in question.
4. For all those English compounds: collect all linked Swedish compounds.
5. Collect all Swedish words linked to the English element as single words.
6. These are then compared to the original Swedish compound and a silhouette link is established between the original Swedish compound and the English element in question.

What we have here is in fact candidate silhouettes (and therefore strings) of translations from English compound elements to substrings of Swedish compounds. The database after phase 4 is illustrated in figure 8.

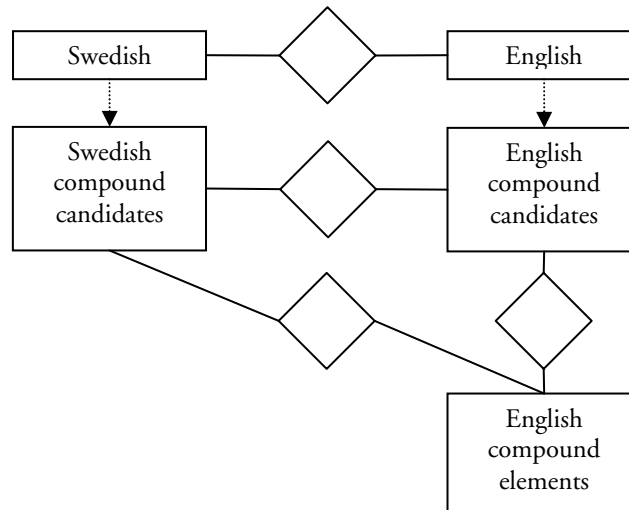


Figure 8: The database after phase 4.

4.3.6 Phase 5: Segmenting Swedish compounds

In this phase the goal is to establish a segmentation of the Swedish compounds, and to decide whether or not they are made up of the same words as the corresponding English compounds. This is done in the following steps:

7. For every Swedish compound: collect all English compounds it is linked to.
8. For all those English compounds: collect the elements that make it up.
9. For all those elements: collect all silhouettes between the element in question and the original Swedish compound. Calculate merit values for all silhouettes.
10. For all collections of silhouettes for all the elements making up the English compound: combine them in all possible permutations to complex silhouettes. Calculate merit values for all complex silhouettes, and sort them by this merit value.
11. Suggest the complex silhouette with the highest merit value to be the best segmentation of the Swedish compound by the example set by the English compound.
12. Modify the merit value of the chosen complex silhouette as described in the Method section
13. Check to see which action operator is appropriate for the selected link and assign it.
14. Construct a text file report of all suggestions, see below.

4.3.7 The report

The report consists of four columns; the merit value, the English compound, the action operator and the segmented Swedish compound. The merit value is printed as a percent value with four decimal places; the English compound is printed as is, although right justified; the action operator is printed as is and the Swedish compound is printed with segments marked by numbered brackets. Like this:

```

7.7909      setup directory != distribution
45.0919    adr vehicle fitting ?= [0adr0]-[1bil1]
77.1979     pressure monitor == [0tryck0][1vakt1]
  
```

This makes for quite intuitive reading as the action operators can be interpreted in a Perl/C kind of way to mean “not translatable”, “perhaps translatable” and “translatable”. This actually makes it readable like this (although the merit value is still just a series of non-sense digits):

```

"setup directory" is not translatable to "distribution"
"adr vehicle fitting" is perhaps translatable to "adr-bil"
"pressure monitor" is translatable to "tryckvakt"
  
```

5 Training

When conducting training of this system one can distinct three different tasks: the training reference and the actual training of both weights and operator thresholds. These three tasks will be covered in sections 5.1, 5.2 and 5.3 respectively.

5.1 Training references

To train this system I will need two kinds of references. First I will need a golden standard, or an a priori reference, which I will try to get as close as possible to. Then I will need to make a correction, or an a posteriori reference, from the report that got closest to the golden standard. Since I intend to make the golden standard somewhat fuzzy it will be able to evaluate compound segmentation but not operator assignment. The correction will be used to evaluate operator assignment, and similar corrections will be used to evaluate the test runs later. The a priori reference and the a posteriori reference will be dealt with in sections 5.1.1 and 5.1.2 respectively.

5.1.1 A priori reference

The a priori reference, or golden standard, will serve as a wish list of how a report should look like. The reference is created without any previous knowledge of how the system actually works, and must therefore be very general. This reference will therefore (unlike the a posteriori reference) be solely based on linguistic knowledge (and in some cases knowledge about the relevant domain). The main aim of the runs against the reference is to tune the systems ability to segment compounds correctly.

A strict regime based on the following rules will be implemented in constructing the reference:

1. The correspondences between an English word and a Swedish segment should have a semantic or syntactic equivalence.
2. For the keep operator (==) to be assigned there should be a correspondence between every word on the English side and the entire word on the Swedish side. Except if a character on the Swedish side corresponds to something that is not realized in English e.g. compounding-s.
3. For the discard operator (!=) to be assigned there should be no correspondence at all between the English and the Swedish words.
4. All links that are not assigned the keep or discard operators are assigned the maybe operator (?=).

Since the a priori reference will be used to determine which setting gives the best segmentation the fuzziness of the operators is tolerable. To create the reference, all of the compound candidates were analyzed (1,859 in total).

5.1.2 A posteriori reference

The a posteriori reference will be created using a strategy of minimal interference. This means that as little as possible is altered in the report to give a tolerable result. A tolerable result means that links that are kept are fit to be included in an automatically generated dictionary, the rest is discarded. Not much attention will be given to syntax at this stage; the focus will rather be on semantics. This means that “pinion bearing” can be linked to something that should really have been linked to “the pinion bearing” and vice versa.

To create this reference a sample of 499 links were selected. Of these; 390 turned out to be acceptable, leaving 109 that should be discarded.

5.2 Optimizing weights

The five weights described in the Method section can be divided into two groups that can be tested rather independently; the ones concerning simple silhouettes and the ones concerning complex ones. The simple silhouette weights are cover (s_cover), frequency (s_freq) and likeness (s_like) and the complex silhouette weights are merit summary (c_merit) and cover (c_cover). The training runs were guided in that I started with a base set of combination and then expanded the runs in the direction that seemed to give best results. The results can be found in table 2. The table is divided into four sections, where the first two sections are part of the base set, the third is the guided improvement, and the fourth was carried out to check that no further improvement could be achieved by making the weights heavier. As the table shows, the best setting seems to be: $c_cover=1$, $c_merit=1$, $s_cover=1$, $s_like=3$ and $s_freq=2$. My intuition was thus put to shame, as I expected cover to be more important than frequency. Then again, the important part of cover is probably expressed in the likeness value, which carries a heavy weight. I would also have expected that some weighting on the complex silhouette would have improved the results, but this seems not to be the case.

c_cover	c_merit	s_cover	s_like	s_freq	# correct segments	segment accuracy (%)
1	1	1	1	1	751	40.40
1	1	1	1	2	762	40.99
1	1	1	2	1	790	42.50
1	1	2	1	1	667	35.88
1	1	1	2	2	801	43.09
1	1	2	1	2	683	36.74
1	1	2	2	1	728	39.16
1	2	1	1	1	748	40.24
1	2	1	2	2	796	42.82
1	2	2	1	2	681	36.63
1	2	2	2	1	721	38.78
2	1	1	1	1	751	40.40
2	1	1	2	2	800	43.03
2	1	2	1	2	683	36.74
2	1	2	2	1	730	39.27
1	1	1	2	3	802	43.14
1	1	1	3	2	803	43.20
1	1	2	1	3	694	37.33
1	1	3	1	2	650	34.97
1	1	2	3	1	789	42.44
1	1	3	2	1	675	36.31
1	1	1	4	2	787	42.33

Table 2: Results from training runs with different weight settings. The best result is highlighted in bold digits. There was a total of 1859 segments in the material.

5.3 Optimizing operator assignment

The optimization process for operator assignment is not a training process as such, but rather a statistically asserted process of finding the right threshold merit values. To do this an a posteriori reference is created according to the principles drawn up above. Since there are no maybe-operators in this reference, it is able to tell exactly whether a link should be keep or not. To establish suitable threshold values, all possible values are tried. This is done by testing how the operators would be assigned if the threshold values were varied between 0 and 100 (in increments of 1 and never overlapping). This produces 5,151 different settings, and they are of course tested automatically. For each setting the number of correctly assigned operators is counted, as well as the number of maybe-operators in the report (there are none in the reference). If one treats the maybe-operators as a failure to deal with that link (as one indeed

could do as the meaning of it is merely “pass on to human inspection”), it becomes meaningful to calculate precision and recall. These would be calculated like this:

$$Precision = \frac{(keep_{ref} \cap keep_{train}) + (discard_{ref} \cap discard_{train})}{keep_{train} + discard_{train}}$$

$$Recall = \frac{(keep_{ref} \cap keep_{train}) + (discard_{ref} \cap discard_{train})}{keep_{ref} + discard_{ref}}$$

To weight precision and recall against each other; a harmonic mean is often created between them. This is called the f-score, and takes both precision and recall into account. The formula for calculating the f-score looks like this:

$$f - score = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

For individual operators I will instead focus on accuracy. This metric is calculated as the number of correctly assigned operators divided by the total number of that kind of operator. Mathematically it looks like this:

$$Accuracy_{op} = \frac{op_{ref} \cap op_{train}}{op_{train}}$$

To decide on one of the 5,151 cases, they were drawn onto a chart for visualization. That chart can be found as figure 9. As one can see; there are a lot of cases where the precision and the recall is lower than in another case. These can be automatically removed, and the result is a tracing of the upper boundary of the maximum performance. This tracing can be seen in figure 10, and represents the precision/recall trade off of this algorithm. When proceeding to choose to best setting, one must decide what the aims are. Emphasizing precision will give a good linking material that lacks in size; emphasizing on recall will give a large linking material with more errors. Another way of looking at this is to emphasize the keep or the discard operator. The keep operator is useful if the material can be manually processed, as a lot of verified cases will lower the amount of manual work needed. The discard operator is useful if there will be no manual processing of the material, and the main interest is to weed out bad links. To cater to all tastes, I will find the best setting for doing either. That means that I will find the case with the highest recall and the one with the highest precision. Furthermore I will find the cases with the highest keep-accuracy and the highest discard-accuracy. I will also calculate which case has the highest f-score, as it will represent a middle path, neither favouring precision nor recall. In doing this I will try to keep away from the extremes and only select cases with reasonable precision and recall. Generally both should be above 50 %. The selection of settings from the edge can be found in table 3.

The values in table 3 are interesting, as they show that there indeed is a great difference between different objectives. As expected, lowering the gap between the two thresholds increases recall, while precision benefits from a large gap. This is because the size of the gap determines the number of links that are assigned the maybe-operator. Since a high precision is easier to attain than a high recall, the f-score also benefits from a small gap.

As I mentioned above, one can use this algorithm to achieve two different goals. The first goal one can have is to verify a linking material that will be verified by hand, but a decrease in the amount of manual labour is desirable. The second goal is to weed out bad links from a linking material that will not be subject to manual processing.

In the first case, one would like to verify as much as possible, but the verified part must have a high precision, as it has to meet the standards of a human eye. The highest precision is attained by setting the discard threshold to 0 and the keep threshold to 75 (in the future I will use the notation 0/75 to describe thresholds). This setting does not throw away anything, but one could perhaps raise the discard threshold to 12, as suggested by the discard precision. Using the thresholds 12/75 is also a viable option. Using this setting would leave half of the material for manual processing. It might be valuable to see how much precision one would have to “pay” to increase the recall, and whether there is a setting that allows an acceptable precision loss at a valuable increase in recall.

In the second case; one would like to be able to throw away as much as possible. Table 3 suggests that a setting of 30/42–45 would allow us to throw away a lot of bad links. Looking at more than the discard section of the table would, however, suggest that the threshold be set to 27/42–45 instead. As for the keep threshold; it does not make much of a difference when we only want to throw away, the reason why only 42–45 have been suggested as keep threshold is probably because these are the only setting that belongs to the total precision/recall edge. It could therefore be set to a value that is most beneficial for the keep precision. This would yield the setting 27/75.

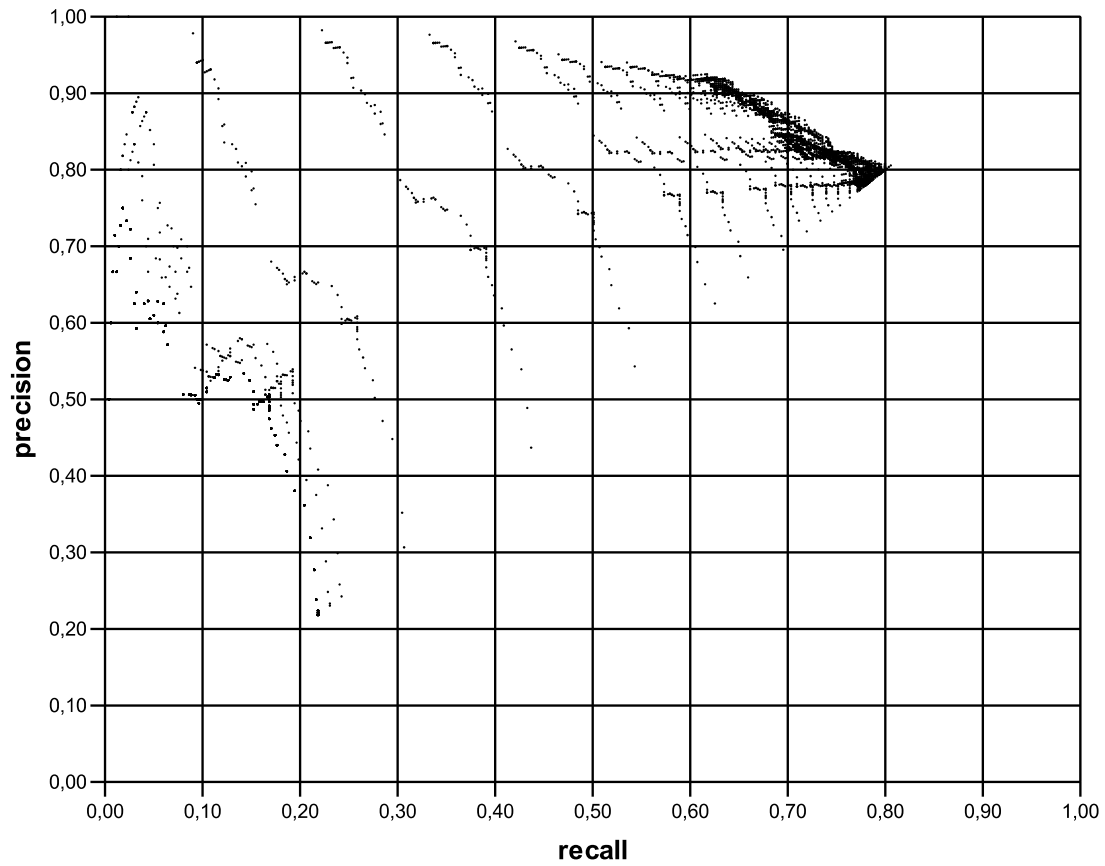


Figure 9: Each dot represents the precision and recall of one test run.

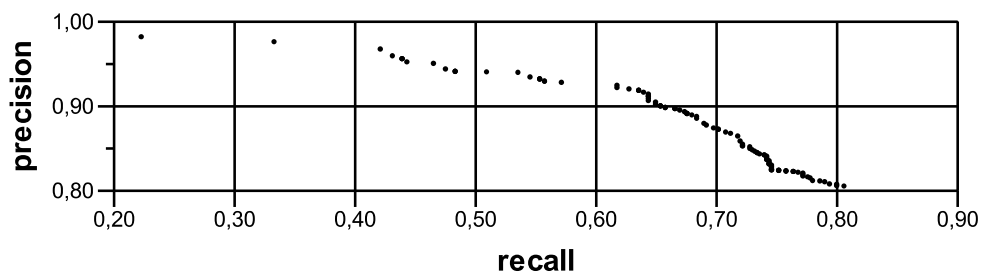


Figure 10: These dots represent the best of the previous graph – the edge.

Operator	Discard threshold	Keep threshold	Precision (%)	Recall (%)	f-score (%)
Total	0	75	94.07	50.90	66.06
	27	27	80.06	80.06	80.06
Keep	0	75	94.07	65.13	76.97
	15–16	25	82.02	95.90	88.42
	27	27	82.48	95.38	88.47
Discard	12–14	27, 29, 30, 50–52, 64, 74	75.00	8.26	14.88
	30	42–45	57.14	29.36	38.79

Table 3: Threshold settings to maximize precision/recall/f-score for keep operator, discard operator and all operators. The maximized value is marked by bold digits.

We now have four cases we wish to have a closer look at: 0/75, 12/75, 30/75 and 27/75. In addition, we would like to see if additional recall can be “purchased” for a slight decrease in precision. The threshold to do this is mainly 75, which could perhaps be lowered. Looking at the trade-off chart, one can see that there are “ledges”, which are suitable to choose candidate thresholds from. These represent the keep thresholds of 74, 72, 64 and 51. They have been summarized into table 4.

Thresholds	Precision	Recall	Precision difference to 0/75	Recall difference to 0/75	Recall/Precision difference ratio
0/75	94.07	50.90	0.00	0.00	0.00
0/74	94.01	53.51	0.06	2.61	43.50
0/72	92.83	57.11	1.24	6.21	5.00
0/64	92.49	61.72	1.58	10.82	6.85
0/51	89.73	66.53	4.34	15.63	3.60

Table 4: Summarization of “ledge edge” settings suitable to exchange precision for recall, along with their precision cost, recall gain and the ratio between the two.

Looking at table 4 it becomes obvious that the keep threshold should be lowered to 74. All that recall for almost no precision is a real bargain. If we want to exchange more precision; our best bet is to lower the keep threshold to 64, as it will get us 6.85 recall for every precision. We now have eight cases of interest; 0/74, 12/74, 30/74, 27/74, 0/64, 12/64, 30/64 and 27/64. These are summarized in table 5.

Thresholds		total			total			keep		
		precision	recall	f-score	precision	recall	f-score	precision	recall	f-score
0	74	94.01	53.51	68.20	94.01	68.46	79.23	n/a	n/a	n/a
12	74	93.24	55.31	69.43	94.01	68.46	79.23	75.00	8.26	14.88
27	74	89.46	59.52	71.48	94.01	68.46	79.23	62.50	27.52	38.22
30	74	87.94	59.92	71.28	94.01	68.46	79.23	57.14	29.36	38.79
0	64	92.49	61.72	74.04	92.49	78.97	85.20	n/a	n/a	n/a
12	64	91.88	63.53	75.12	92.49	78.97	85.20	75.00	8.26	14.88
27	64	88.71	67.74	76.84	92.49	78.97	85.20	62.50	27.52	38.22
30	64	87.40	68.14	76.58	92.49	78.97	85.20	57.14	29.36	38.79

Table 5: Summary of the eight best threshold settings, along with precision, recall and f-score for the keep operator, the discard operator and a combination of the two. Highest scores for each column is marked with bold digits.

Looking at table 5 one can observe that the best thresholds for precision maximization are still 0/74. Second place is held by 0/64 (as 12/74 provides less precision and less recall compared to 0/74). Third place would be 12/64 where one can increase the recall by 1.81 at the price of 0.61 precision. As for weeding out bad links; the best setting is to have the discard threshold at 30. The general performance of the algorithm is best measured by the thresholds 27/64, as they yield the best total f-score.

This reasoning has now yielded three settings; one for automatic verification followed by human verification (0/74), one for automatic weeding of bad links (30/64, 30/74) and one for general performance tests (27/64). I intend to test all of these three threshold settings on new linking material in section 6.

6 Evaluation and Results

The evaluation of the result of the implementation is based on the report it generates. This section will deal with evaluation methodology (section 6.1) and the results from testing (section 6.2).

6.1 Evaluation methodology

The algorithm will be run using the settings obtained through training, then compared to a reference key. When comparing to the reference key I will evaluate how good the implementation was at segmenting the compounds as well as how good it was at purifying the linking material (operator assignment). The creation of the reference key will be described in section 6.1.1. The evaluation of segmentation and operator assignment is described in section 6.1.2 and 6.1.3 respectively.

6.1.1 Creating a reference

The reference used to test the algorithm will be a posteriori references like those described in section 5.1.2 under Training. There will be one reference for the technical text and one for the literary text. The technical reference will be a sample of 500 links since it is too much work to correct all 1,950. The literary reference will comprise all 450 links in the literary material.

6.1.2 Evaluating segmentation

Since the reference will contain segmented link units, the evaluation of the report will be done by comparing the segments in the report to those in the reference. An accuracy score will be calculated by dividing the number of correct segments by the total number of segments.

6.1.3 Evaluating operator assignment

As stated in section 5.3, there will be three tests carried out: one to maximize the “purity” of verified links, one to discard as many bad links as possible and one to test general performance (maximizing the total f-score). These will be tested by setting the keep/discard thresholds differently. To maximize verification purity; the thresholds will be set to 0/74 (discard everything with a merit of less than 0 and keep everything with a merit of more than 74). To weed out bad links, the threshold will be set to 30/X (since the keep threshold is of no interest to the discard operator). The general performance will be measured as a maximized f-score between the total precision and the total recall. That value should be maximized with the threshold settings 27/64.

All of these three settings will be tested against the two test materials.

6.2 Results from testing

In this section I will report the results from the testing. I will do so in two sections: one for technical text (6.2.1) and one for literary text (6.2.2). For each of the two text types I will report how well the three tests went (purity, weeding and general performance – c.f. 6.1.3).

6.2.1 Technical text

Segmentation

The algorithm managed to find 665 of the 1079 segments in the reference material. This corresponds to an accuracy of 61.63 %.

Operator assignment

A comparison between the scores of the three test objectives can be found in table 6. As expected; having a certain objective produces a certain result. The first row shows the scores for the objective of getting a pure verified part. Here we also get the highest keep precision (although the general performance objective also yields a good result). The second row shows the score for the weeding objective. This setting performs well in the discard section – or rather the least bad. The third row shows scores for the general performance setting. Here we have maxed out the total f-score (which also leads to the highest recall due to the relation between precision and recall). It is also interesting to see what has happened in the keep section: the precision is only slightly lower, and the recall is significantly higher. In buy-in terms we have purchased 14.82 recall for a mere 0.7 precision, that means we are getting 21.17 recall for every precision.

Thresholds		total			keep			discard		
		precision	recall	f-score	precision	recall	f-score	precision	recall	f-score
0	74	96.47	54.60	69.73	96.47	60.40	74.29	n/a	n/a	n/a
30	n/a	n/a	n/a	n/a	n/a	n/a	n/a	23.08	25.00	24.00
27	64	87.72	70.00	77.86	95.77	75.22	84.26	22.73	20.83	21.74

Table 6: Comparison between test objectives for technical text. The highest value of each column is marked in bold digits.

6.2.2 Literary text

Segmentation

The algorithm managed to find 542 of the 936 segments in the reference material. This corresponds to an accuracy of 57.91 %.

Operator assignment

A comparison between the scores of the different test objectives can be found in table 7. The same pattern as for technical text can be seen. In the technical text we could buy 21.17 recall for every precision when altering the keep threshold from that of the purity objective to that of the general performance objective. In literary text we can buy 6.97 recall for 1.75 precision. This means we get 3.98 recall for every precision. At this rate it is not as evidently beneficial to lower the keep threshold as it was on the technical text.

Thresholds		total			keep			discard		
		precision	recall	f-score	precision	recall	f-score	precision	recall	f-score
0	74	97.33	32.44	48.67	97.33	37.73	54.38	n/a	n/a	n/a
30	n/a	n/a	n/a	n/a	n/a	n/a	n/a	18.82	55.56	28.11
27	64	57.06	45.78	50.80	95.58	44.70	60.92	18.33	52.38	27.16

Table 7: Comparison between test objectives for technical text. The highest value of each column is marked in bold digits.

7 Conclusion

In this conclusion I summarize how well the objectives of the study were met. In my statement of purpose I stated that the objectives would be to segment Swedish compounds and to evaluate the links that link them to English compounds. These are two objectives, and I will present how well they were met one at a time.

7.1 Segmentation

The segmentation objective was measured by how many of the manually identified segments that were found. To normalize this value; accuracy was used. The test materials did not differ that much on segmentation accuracy: for technical text 61.63 % of the segments were found, and for literary text 57.91 % were found. I do not think that this has too much with genre to do, but rather with the size of the material. I believe so because the training material was also a technical text, but smaller than the technical test material. The training material is closer to the literary test material in size, and it scored only 43.20 % accuracy.

When looking on these numbers, some things should be kept in the back of the head. First; the reference specifies all findable segments. This means that segments of partial links are marked in the reference. These are segments that the algorithm is, in an intrinsic way, designed not to find.

Secondly; the robustness of the algorithm can be questioned. It does produce a result for every link, but it does this by adding an empty silhouette to elements that are part of only the currently processed compound. This means that some links will have a bad segmentation because of data sparseness.

Even with these points being made; I do not think that the results are good enough. The approach taken by Koehn & Knight (2003) yields an accuracy of 99.1 %. It should, however, be said that their algorithm has the opportunity to look at a bigger picture, as it has access to the whole parallel corpus. The algorithm presented here only looks at the links created by aligning a parallel corpus.

The decision to use only the links and not the whole corpus was a conscious decision, as the algorithm is designed to improve a lexicon created by word alignment rather than be an integrated part of an alignment or translation system.

7.2 Link evaluation

The links that the algorithm has attempted to segment are evaluated as fit to keep, fit to discard or uncertain (which means that a human eye is needed in order to evaluate them). In the training process; three objectives for the link evaluation was set. The first was to verify which links to keep, assuming a human would look at the rest. This objective is reached by maximizing precision for the keep operator while keeping a reasonable recall. The second was to weed out as many as possible of the bad links. This objective does not count on a human to complete the job, and is achieved by maximizing precision and recall for the discard operator. The third objective was to test the general performance of the system, by maximizing the *f*-score. Only the two first objectives are practical ones, the third one has more of an academic interest. Thus the two first will be commented on more closely.

The second objective – the weeding out of bad links – does not live up to expectations. The *f*-scores are 24.00 % (technical text) and 28.11 % (literary text). These numbers are far to low to be of any practical use. It should be said though that some of the problems with this objective is due to the problems of data sparseness mentioned in the previous section. This is because of the fact that those English elements that do not have any Swedish units to extract possible Swedish elements from will get a merit of 0. This in turn will make it likely that they are evaluated as discardable even if they might be perfectly good compounds. I.e. some of the links evaluated as bad might be so because of data sparseness.

The first objective, on the other hand, seems to fare quite well. The precision of the keep operator is always above 95 %, even if the recall leaves some to be desired. One way to better the recall is to use the general performance settings instead. They yield slightly less in precision (but still above 95 %), and improves the recall. This recall improvement is very high for the technical text, but not so high for the literary text. Again this might have less with genre to do than with corpus size, as the training material is somewhere between the two test materials in terms of recall increase.

To summarize: the objective of verifying some links as part of a manual verifying job can be performed, whereas the objective of automatic weeding out still leaves a lot to be desired. It should be noted that there are a lot of things that could be done to improve these results. It is, however, beyond the scope of this paper.

8 Outlook

As seen in the previous section (Conclusions), there is really only one field in which the algorithm, as trained now, fares well. It could be used to speed up manual evaluation of links. I do, however, think that there are some ways in which the algorithm can be improved, and I also think that the concept of silhouettes can be used in other applications. I will therefore divide this outlook into two sections: improving the algorithm (8.1) and further uses of silhouettes (8.2).

8.1 Improving the algorithm

The most pressing need is to improve the robustness of the algorithm. This should ideally be done by incorporating some way to deal with data sparseness. One conceivable way to deal with one missing element in a compound would be to assign the leftovers to that element. One would then have to be fairly convinced that the other elements are correct. I think this should only be attempted on a training material without flaws, i.e. a constructed material containing compounds that are verified as correct. I also believe that training on such a material would yield better results in general.

Another thing that needs to be addressed is the fact that Swedish and other Germanic languages often uses some kind of connection between the compounded words rather than just stack them on top of each other. In Swedish this is mainly done by inserting a compounding-s. The words *information* ‘information’ and *tavla* ‘board’ are not compounded into **informationtavla*, but instead into *informationstavla*. In the algorithm presented here this compounding-s might stick to either or none of the words surrounding it. Needless to say: the accuracy of the segmentation would be better off knowing where to put the compounding-s. Another way to compound is to drop or alter a vowel at the end of a word. The words *mätare* ‘gauge’ and *rensare* ‘cleaner’ are not compounded into **mätarrensare*, but instead to *mätarrensare*, dropping the final *e* in *mätare*. The altering of vowel can be exemplified by compounding *olja* ‘oil’ and *fat* ‘barrel’. These words are compounded into *oljefat* instead of **oljafat*, altering the final *a* in *olja* to an *e*. One way to address these problems would be to build in this kind of knowledge into the algorithm, allowing it to do simple morphological transformations and see if they make for better compounds.

8.2 Further uses of silhouettes

In this paper the notion of silhouettes is introduced. This is a notion which I think have great potential in corpus linguistics. Since it allows extraction and piecing together of sub-word strings it has a number of potential uses.

One way to use silhouettes could be to identify inflection patterns of nouns. Take the word *broarna* ‘the bridges’ for example. This word should be analyzed as having the stem *bro* ‘bridge’, the second declination plural *ar* and the plural definite case *na*. If other forms of *bro* are present along with other second declination plurals and other plural definite cases, a program might be able to surmise that the word *broarna* is made up of the stem *bro* and is a second declination plural in definite case, through the use of silhouettes. One would probably have to define all possible inflection patterns beforehand and let the program try to assign one such pattern to each stem.

Yet another way to use silhouettes might be to identify linking segments before they are actually linked. This would be of use in the kind of algorithm that Jones & Alexa (1997) describe. Instead of trying all possible n-grams, one would analyze the corpus beforehand and have a limited set of possible link units. This could be achieved by treating words in the same way as the algorithm presented in this paper treats characters. Each sentence of the corpus would then be compared to all the other sentences, and silhouettes extracted. A limited number of models of how each sentence could be completely covered by potential link units could then be constructed.

References

- Ahrenberg, Lars, Mikael Andersson & Magnus Merkel (1998). "A simple hybrid aligner for generating lexical correspondences from parallel texts" in: *Proceedings of COLING-ACL '98*, pp. 29–35, Montreal, Canada.
- Al-Onaizan, Y., J. Curin, M. Jahr, K. Knight, J. Lafferty, D. Melamed, F.-J. Och, D. Purdy, N. A. Smith & D. Yarowsky (1999). *Statistical machine translation*. Final report, John Hopkins University Summer Workshop. Available at http://www.clsp.jhu.edu/ws99/projects/mt/final_report/mt-final-report.ps (November 24:th 2004).
- Brodda, Benny (1979). *Något om de svenska ordens fonotax och morfotax: iakttagelser med utgångspunkt från experiment med automatisk morfologisk analys*. Papers from the Institute of Linguistics 38, University of Stockholm.
- Dura, Elzbieta (1998). *Parsing words*. Doctoral Thesis, Data Linguistica 19, University of Göteborg, Sweden.
- Gale, William A. & Kenneth W. Church (1991) "A Program for Aligning Sentences in Bilingual Corpora" in *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL'91)*, pp 177–184
- Jones, Daniel & Melina Alexa (1997). "Towards automatically aligning German compounds with English word groups" in Jones, D.B. & H. L. Somers (eds) *New Methods in Language Processing*, pp 199–206, UCL Press.
- Karlsson, Fred (1992). "SWETWOL: A Comprehensive Morphological Analyser for Swedish" in *Nordic Journal of Linguistics*, 15, pp 1–45
- Kay, Martin (1980). "The Proper Place of Men and Machines in Language Translation" Xerox PARC Working Paper, reprinted in *Machine Translation 12 (1–2)*, 1997, pp. 3–23
- Koehn, Philipp & Kevin Knight (2003) "Empirical Methods for Compound Splitting" in *11th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2003)*, pp 187–194.
- Koskenniemi, Kimmo (1983) *Two-level Morphology: A General Computational Model for Word-form Recognition and Production*. Publications of the Department of General Linguistics, 11, University of Helsinki.
- Olsson, Leif-Jöran (2004). *Utveckling av lexikala resurser för ett språkgranskningssystem för svenska*. Master's Thesis, Uppsala University, Uppsala, Sweden.
- Sågvall Hein, Anna (1983). *A Parser for Swedish. Status Report for Sve.Ucp. February 1983*. Rapport nr UC DL-R-83-2. Uppsala universitet. Centrum för datorlingvistik.
- Sågvall Hein, Anna (1999) "The PLUG-project: Parallel Corpora in Linköping, Uppsala, Göteborg. Aims and achievements" in: *Working Papers in Computational Linguistics & Language Engineering 16*. Department of Linguistics, Uppsala Universitet.
- Sågvall Hein, Anna, Eva Forsbom, Jörg Tiedemann, Per Wijnitz, Ingrid Almqvist, Leif-Jöran Olsson & Sten Thaning (2002) "Scaling up an MT Pototype for Industrial Use. Databases and data flow" in: *Proceedings of LREC 2002. Third International Conference on Language Resources and Evaluation*. Volume 5, pp. 1759–1766.
- Tiedemann, Jörg (1999). "Word Alignment Step by Step" in: *Proceedings of the 12th Nordic Conference on Computational Linguistics*, University of Trondheim, Norway.
- Tiedemann, Jörg (2004). *Recycling Translations – Extraction of Lexical Data from Parallel Corpora and their Application in Natural Language Processing*. Doctoral Thesis, Acta Universitatis Upsaliensis, Studia Linguistica Upsaliensia 1, Uppsala, Sweden.
- Åberg, Stina (2003). *Datoriserad analys av sammansättningar i teknisk text*. Master's Thesis, Uppsala Universitet, Uppsala, Sweden.