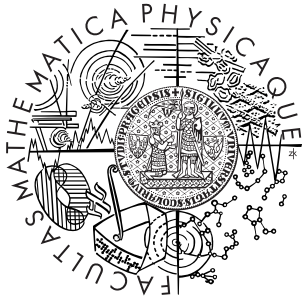
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jana Kravalová

Automatický word alignment
Automatic word alignment

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Pavel Pecina
Studijní program: Informatika, Obecná informatika

2007

Ráda bych poděkovala vedoucímu své práce Mgr. Pavlu Pecinovi za odborné vedení během mé práce. Jeho optimismus, podpora a velký rozhled v oblasti strojového zpracování textu mě přivedly ke studiu tohoto oboru.

Také bych chtěla poděkovat Milanu Strakovi za užitečné rady k optimalizaci programu.

V neposlední řadě bych velmi ráda poděkovala svým rodičům, bez jejichž ochoty a trpělivosti bych nemohla tuto práci dokončit.

Prohlašuji, že jsem svou bakalářskou práci napsala samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 24. 5. 2007 Jana Kravalová

Contents

Název práce: Automatický word alignment
Autor: Jana Kravalová
Katedra (ústav): Ústav formální a aplikované lingvistiky
Vedoucí bakalářské práce: Mgr.Pavel Pecina
e-mail vedoucího: pecina@ufal.mff.cuni.cz

Word alignment (párování slov) je klíčovou komponentou moderních systémů statistického strojového překladu. Vstupem je věta ve dvou jazycích a úkolem automaticky spárovat slova v obou jazycích tak, aby se nalezly nejpravděpodobnější překladové ekvivalenty. Jako varianta klasického generativního přístupu (IBM modely) se dnes prosazují i diskriminativní přístupy, které tuto úlohu řeší jako hledání maximálního hranového pokrytí v úplném ohodnoceném bipartitním grafu. Vrcholy grafu jsou tvořeny slovy v jednom a v druhém jazyce, hrany jsou ohodnoceny mírou asociace odhadnutou z trénovacích dat. Práce se zaměřuje na efektivní implementaci algoritmu pro hledání maximálního pokrytí bipartitního grafu, implementaci výpočtu ohodnocení hran bipartitního grafu a provedení základních experimentů.

Klíčová slova: párování slov, maximální hranové pokrytí

Title: Automatic word alignment
Author: Jana Kravalová
Department: Institute of Formal and Applied Linguistics
Supervisor: Mgr.Pavel Pecina
Supervisor's e-mail address: pecina@ufal.mff.cuni.cz

Word alignment is a crucial component of modern machine translation systems. Given a sentence in two languages, the task is to determine which words from one language are the most likely translations of words from the other language. As an alternative to classical generative approach (IBM models) new methods based on discriminative training and maximum-weight bipartite matching algorithms for complete bipartite graphs have been proposed in recent years. The graph vertices represent words in the source and target language. The edges are weighted by measures of association estimated from parallel training data. This work focuses on the effective implementation of maximum weight bipartite matching algorithm, implementation of scoring procedures for graph vertexes, and basic experiments and their evaluation.

Keywords: word alignment, maximum-weight edge cover

Chapter 1

Introduction

Automatic word alignment is a problem from the area of computational linguistic usually performed as the first step in most statistical machine translation systems. Given a parallel sentence (pair of sentences in source and target language, e.g. English and French), the goal is to find translation equivalents, that is, to find the corresponding pairs of words or expressions. For example, word alignment in the sentence pair "They work with them ." and "Elles travaillent avec eux ." is shown in Figure 1.1.

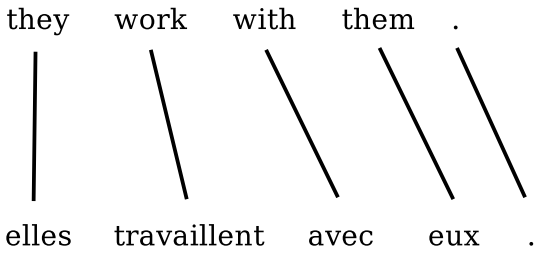


Figure 1.1: Word alignment example

In this work, we describe and implement automatic word alignment method based on discriminative approach using bilingual lexical association statistics, as it is proposed in [9] and [5]. The word alignment problem is reduced to a maximum-weight edge cover problem in a complete bipartite graph where the graph nodes represent words of the two sentences, one partity for the source language and one partity for the target language. All nodes of different partities are connected by edges, thus creating a complete bipartite graph with edges corresponding to all possible word pairs. The edges are weighted according to the strength of translation association (desirability) of this word connection. The weights are estimated using a relatively large sentence-aligned parallel corpus and a small sample of word-aligned data. The edge cover, obtained by the maximum-weight edge cover algorithm for bipartite graphs is interpreted as a set of word pairs satisfying the word alignment.

Chapter 2

Data and Methodology

2.1 Overall process

The overall process of building our word alignment system is shown in Figure 2.1 and includes these phases:

1. **Association measures estimation.** For each word pair (bigram) occurring in the sentence-aligned parallel corpus (SA training data), we estimate its possible strength of association by computing scores of various association measures. This phase is described in Section 2.3.
2. **Feature coefficients training.** Using scores of association measures from the first phase and other linguistic (orthographic) features we represent each possible word pair from WA training data (an edge in the bipartite graph) as a feature vector $\vec{f} = (f_1, \ldots, f_n)$ and (by SVM principle) find feature coefficients $\vec{a} = (a_1, \ldots, a_n)$ maximizing linear combination $\vec{a} \times \vec{f}$ for edges from the alignment and minimizing it for others. This part is described in Section 2.5.
3. **Null coefficient training.** For those words, which cannot be aligned to any word, a special null word is inserted in the sentence. The weight of graph edge between words and the special null word must also be trained. This part is described in Section 2.6.
4. **Automatic word alignment using maximum-weight edge cover.** We create a bipartite graph for each parallel sentence in WA testing data and weight the edges between each word pair e_i, f_j using the linear combination

$$\sum_{k=1..n} a_k \cdot f_k(e_i, f_j)$$

where f_k is feature value estimated in the first phase or a linguistic (orthographic) feature and a_k is corresponding linear coefficient trained in the second phase. Then we

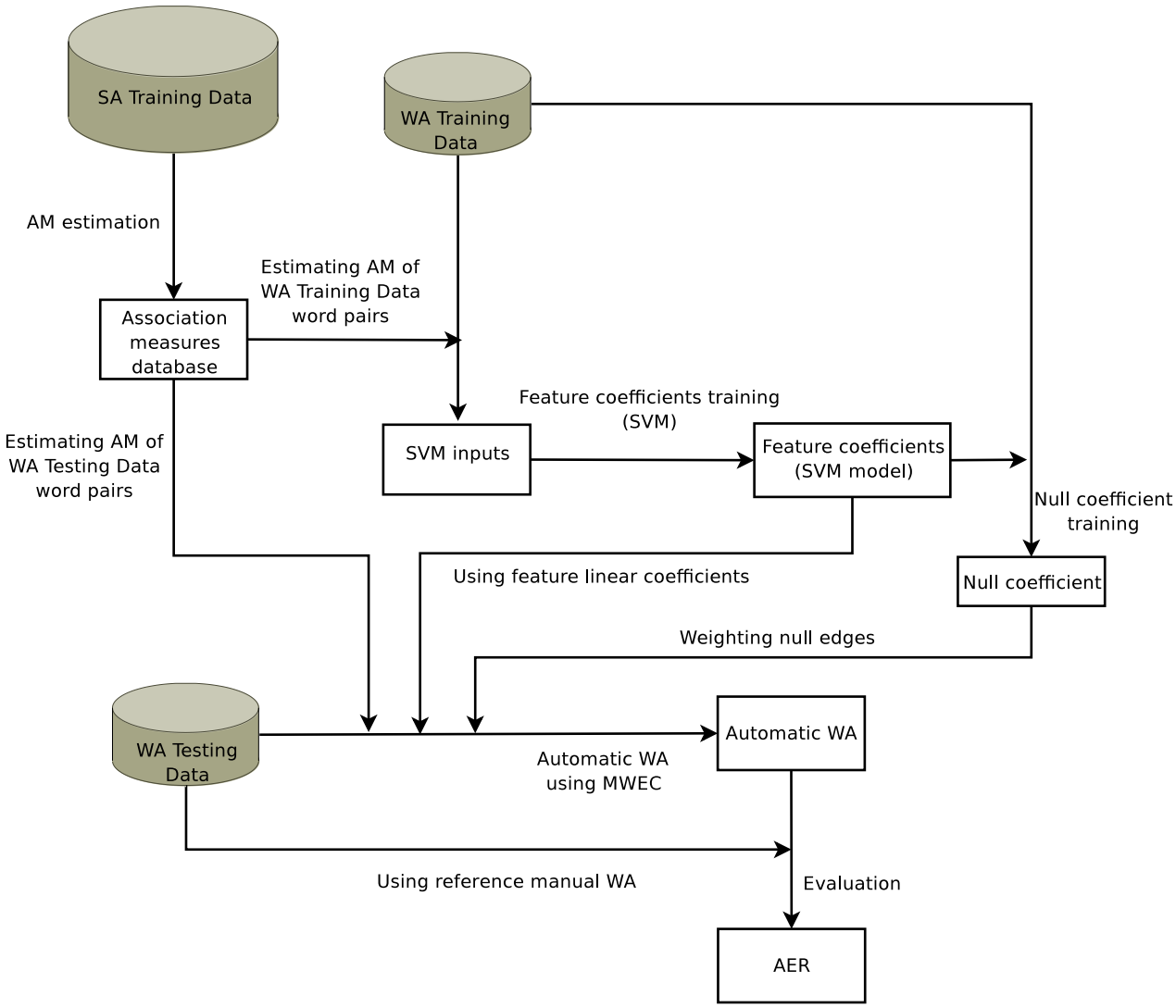


Figure 2.1: Overall process

run the maximum-weight edge cover algorithm on the graph and obtain the automatic word alignment. This phase is described in Section 2.7.

5. **Evaluation.** We use manual reference word alignment from WA testing data to evaluate the results of the automatic word alignment. This phase is described in Section 2.8.

2.2 Data

The experiments were carried out using data from the Canadian Hansard Corpus. We divided the data in three parts (see Figure 2.1):

- **SA training data.** We used $1,130,104$ English-French pairs of aligned sentences from the training section to estimate scores of association measures.
- **WA training data.** We used 37 word-aligned sentence pairs from trial section plus 100 word-aligned sentence pairs from the testing section for training of feature coefficients.
- **WA testing data.** For final testing and evaluation, 337 sentence pairs from the testing section were used.

2.3 Association measures

As described in [7], (lexical) association measures are mathematical formulas determining the strength of association between two words based on their occurrences and co-occurrences in a (parallel) text corpus. They have a wide spectrum of applications such as automatic collocation extraction, dependency parsing, and in our case, bilingual word alignment.

First, from large sentence-aligned corpus (in our $1,130,104$ sentence pairs of SA training data) we extract these joint and marginal frequencies:

$a = f(e_i, f_j)$	number of bigrams where e_i, f_j co-occurred in aligned sentence pairs
$b = f(e_i, \overline{f_j})$	number of bigrams where e_i occurred and f_j did not
$c = f(\overline{e_i}, f_j)$	number of bigrams where f_j occurred and e_i did not
$d = f(\overline{e_i}, \overline{f_j})$	number of bigrams where neither e_i nor f_j occurred
$f(e_i, *)$	number of bigrams where e_i occurred (marginal frequence)
$f(*, f_j)$	number of bigrams where f_j occurred (marginal frequence)
N	number of all bigrams

Then, for each word pair we compute scores of the following selected association measure presented in [7]:

1. **Conditional probability** ([7].2)
2. **Pointwise mutual information** ([7].4)
3. **Mutual dependency (MD)** ([7].5)
4. **Log frequency biased MD** ([7].6)
5. **Normalized expectation** ([7].7)
6. **Jaccard** ([7].22)
7. **First Kulczynsky** ([7].23)
8. **Second Kulczynsky** ([7].25)
9. **Yulle's** ω ([7].28)
10. **Yulle's Q** ([7].29)
11. **Driver-Kroeber** ([7].30)
12. **Pearson** ([7].32)
13. **Baroni-Urbani** ([7].33)
14. **Braun-Blanquet** ([7].34)
15. **Simpson** ([7].35)
16. **T combined cost** ([7].43)
17. **Confidence** ([7].48)
18. **Certainty factor** ([7].52)
19. **Added value** ([7].53)
20. **Klosgen** ([7].55)

2.4 Linguistic features

Apart from the association measures estimated in the first phase, we also employ some linguistic (orthographic) features for word pair e_i, f_j. These features are motivated by the need to reflect linguistic knowledge and translation experience. Thus we can, for example, try to prevent the alignment between punctuation and non-punctuation token, add preference to alignments of very similar words, etc. These features are added to feature vector as features number 21..34.

21. **Distance** ($\text{abs}(\frac{i}{|e|} - \frac{j}{|f|})$) – This adds preference of alignments which lie near the diagonal.
22. **Relative length** ($\max(\frac{|e_i|}{|f_j|}, \frac{|f_j|}{|e_i|})$) – This is to prevent the alignments of differently sized tokens.
23. **Short and long** – This indicates that the first token in pair is short while the second is long.
24. **Both short** – This indicates that both tokens are short.
25. **Same tokens** – This indicates that the tokens are exactly the same.
26. **Both upper** – This indicates that both words start with upper character.
27. **Number and non-number** – This is to avoid alignment between digit and non-digit tokens.
28. **Punctuation and non-punctuation** – This is to avoid alignment between punctuation and non-punctuation tokens.
29. **Not the same punctuation** – This indicates that both tokens are punctuation, but not the same.
30. **The same punctuation** – This indicates the same punctuation.
31. **Same number** – This indicates the same number.
32. **Longest common substring** – normalized length of longest common substring (continuous)
33. **Edit distance** – Levenshtein distance, normalized
34. **Dictionary** – dictionary of most frequent English-French translations (such as "the" – "le"). The dictionary is very simple and contains only a few translations which we found useful, but it is possible to add more translations to improve the feature.

2.5 Feature coefficients training

To construct a bipartite graph and find maximum-weight edge cover for each sentence pair e and f, we needed to weight each graph edge with the strength of association of possible word connection between the corresponding word pair e_i, f_j.

We weight every word pair e_i, f_j using the linear combination

$$\sum_{k=1..n} a_k \cdot f_k(e_i, f_j)$$

where f_k is feature value and a_k is corresponding linear coefficient. From a relatively small set of word-aligned sentences (WA training data) we extracted 5000 positive bigrams (aligned word pairs) and 5000 negative bigrams (non-aligned word pairs) and represented them as feature vectors $\vec{f} = (f_1, \ldots, f_n)$. The linear coefficients $\vec{a} = (a_1, \ldots, a_n)$ were trained by Support Vector Machine. After trying several kernels, we used radial basis function (RBF) kernel and found the best values of parameters γ and C using grid search (for details see [4]).

2.6 Null alignment

For some words, it is sometimes impossible to find matching word connection in a given sentence pair. This happens, for example, in sentence pair "they know about the overproduction problem ." and "ils connaissent très bien le problème de surproduction .", where the words "très bien" on French side are not translated on English side. Also, this can happen for grammatical reason.

In some alignment techniques, we can ignore such a word so that we do not connect it to any word. However, some methods, including our bipartite matching, require every word to be connected to "something". We can solve this by adding an extra "null word" to both sentences and those words, which do not fit to any target word, can be connected to this null word. In evaluation, the null alignments are ignored.

Since the quality of our alignment depends on finding good graph edge weights, we have to solve the problem of weighting the null edges, that is to find the likelihood of null alignment.

We decided to weight the null edges using function $min + (max - min) \cdot null$, where min is the minimal edge and max is the the maximum weight in graph. Coefficient $null$ sets the weight of null edges in a range $min..max$. It should be small enough to prevent all alignments end up as null alignments, but it also must be large enough to provide good alternative for words which cannot be aligned to anything else. We trained the value of $null$ using grid search.

2.7 Automatic word alignment using maximum-weight edge cover

2.7.1 Definitions

Firstly, let us start with some necessary definitions:
Bipartite graph $G = (V, E)$ where $V = (A \cup B)$, $A \cap B = \emptyset$ is a graph where $\forall e \in E : e = \{a, b\}$, $a \in A$, $b \in B$. We call A, B **partities**. Simply, we can connect only vertices from different partities.
Complete bipartite graph $G = (A \cup B, E)$ is a bipartite graph with all possible edges between A and B ($E = A \times B$).
Example: Complete bipartite graph is shown in Figure 2.2.

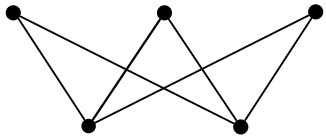


Figure 2.2: Complete bipartite graph

Weighted graph is a graph with a function $w : E \to \mathbb{R}$. (We give a weight to each edge.)
Matching of a graph $G = (V, E)$ is a subset $F \subseteq E$ such that no two of edges in F share a vertex in common.
Perfect matching of a graph $G = (V, E)$ is a matching such that for each vertex there is an adjacent edge in F.
Minimum-weight bipartite matching of a weighted bipartite graph is a perfect matching such that the sum of edge weights is the smallest possible of all perfect matchings.
Edge cover of a graph $G = (V, E)$ is smallest subset $H \subseteq E$ such that for each vertex there is an adjacent edge in H.
Minimum-weight edge cover is an edge cover of a weighted graph, whose sum of edge weights is the smallest of all possible edge covers. (**Maximum-weight edge cover** is an edge cover of a weighted graph, whose sum of edge weights is the largest of all possible edge covers.)

2.7.2 Word-alignment as maximum-weight edge cover

Consider bipartite graph in Figure 2.3.

We used English words "they work with them ." to create vertices of the first partity and French words "elles travaillent avec eux ." to create vertices of the second partity. We now connect all English words with all French words and weight each edge. Weight $w(e)$ indicates the likelihood that pair $e = \{e_i, f_j\}$ will be aligned. (An example of edge weights for this graph is shown in Figure 2.4.) By finding maximum-weight edge cover (Figure 2.4 and 2.5) we find the most valued word-alignment. Firstly, by finding any edge cover we

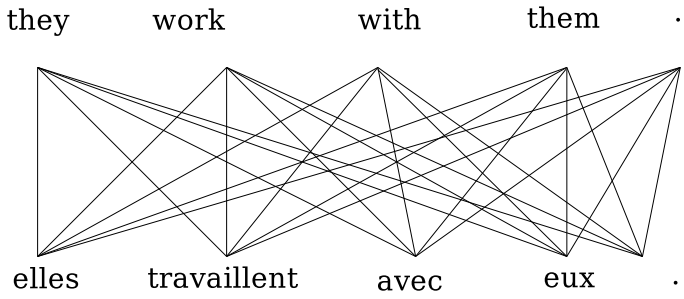


Figure 2.3: Bipartite graph created with words of two bilingual sentences

cover all vertices of graph, therefore we align every word. Secondly, because we have found maximum-weight edge cover, we have chosen the most valued solution of word alignment.

	elles	**travaillent**	**avec**	**eux**	**.**
they	237.5	63.7	38.1	18.6	-344.5
work	80.0	223.5	123.3	41.1	-315.9
with	0.1	21.7	243.5	67.2	-233.1
them	34.0	-16.5	106.8	249.6	-193.5
.	-390.2	-382.4	-244.6	-116.5	332.0

Figure 2.4: Graph weights for English-French sentence pair

2.7.3 Reduction from maximum-weight edge cover to minimum-weight bipartite matching

Because the problem of minimum-weight edge cover is a known graph problem for which polynomial algorithms exist, we will reduce our problem to this one. We choose a large value "bigval" (larger than any value in our graph) and rescale our graph values using this formula:

$$\forall e \in E : w(e) = bigval - w(e)$$

.

We will solve minimum-weight edge cover by reducing it to another problem which can be solved easily. The problem is called minimum-weight bipartite matching. The input of minimum-weight bipartite matching is a weighted bipartite graph, the output is minimum-weight bipartite matching.

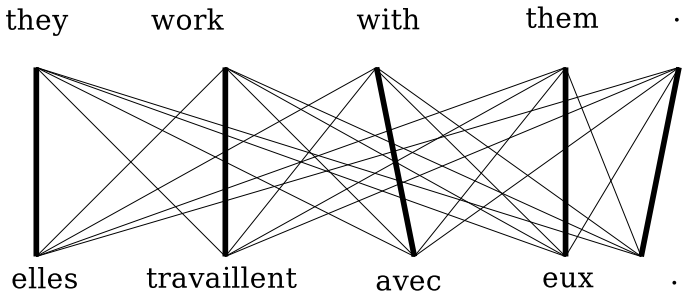


Figure 2.5: Word alignment as maximum-weight edge cover

We have already created a bipartite weighted graph $G_1 = (A_1 \cup B_1, E_1)$, where A_1 is the source partity (source sentence) and B_1 is the target partity (target sentence).

First of all, we make a "copy" of G_1, graph $G_2 = (A_2 \cup B_2, E_2)$, which is exactly the same as G_1, so $E_1 = E_2$, $A_1 = A_2$, $B_1 = B_2$. The weights will be the same, too. Then, we connect G_1 and G_2 to get a new bipartite graph G_{perf} to run minimum-weight bipartite matching on. We connect corresponding vertices of $V_1 = A_1 \cup B_1$ and $V_2 = A_2 \cup B_2$, so we create new edges $\{v_1, v_2\}$, where $v_1 \in V_1$ and $v_2 \in V_2$.

Finally, we have to weight the new edges. For each vertex $a_1 \in A_1$ we look at its neighbours $b_i \in B_1$ and find the smallest weight $w(e)_{\min}$ of all edges $\{a_1, b_i\}$. (The smallest weight of all edges between a_1 and its neighbours.) Once we have found $w(e)_{\min}$, we double it and use it as weight for new edge between a_1 and its corresponding vertex $a_2 \in A_2$. We then repeat the same for each vertex b_1 in B_1.

Example: Figure 2.6 shows how to make new graph G_{perf}.

Note: The new graph G_{perf} is a bipartite graph with partities $A_1 \cup B_2$ and $B_1 \cup A_2$, so $|A_1 \cup B_2| = |B_1 \cup A_2|$, therefore a perfect matching F can be found and $|F| = |A_1 \cup B_2| = |B_1 \cup A_2|$.

Let us now assume we have already solved minimum-weight bipartite matching and have obtained a set $F_{perf} \subseteq E_{perf}$. We now have to turn the results into minimum-weight edge cover and get a set $F_1 \subseteq E_1$ which would be our solution.

- Each edge $e = \{a_1, b_1\}$, where $e \in F_{perf}$, $a_1 \in A_1$ and $b_1 \in B_1$, we add into F_1.
- Each edge $e = \{a_2, b_2\}$, where $e \in F_{perf}$, $a_2 \in A_2$ and $b_2 \in B_2$, we ignore.
- We ignore every edge $e \notin F_{perf}$.
- For each edge $\{a_1, b_2\}$ or $\{b_1, a_2\}$ we find the adjacent edge to a_1, respectively. b_1, with smallest weight and add this edge to F_1.

We claim that this reduction to minimum-weight bipartite matching solves minimum-weight edge cover problem correctly. This reduction along with a proof is described in [8].

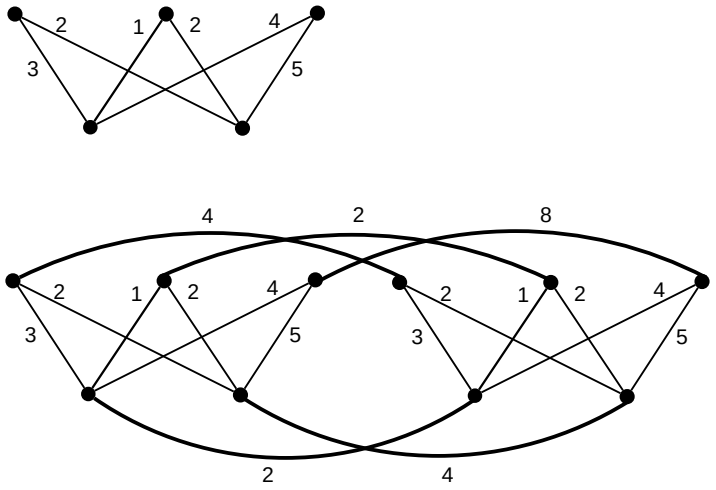


Figure 2.6: Reduction from minimum-weight edge cover to minimum-weight bipartite matching

2.7.4 Solving minimum-weight bipartite matching

Minimum-weight bipartite matching is also a problem which can be solved in polynomial time using a well known Hungarian algorithm. This algorithm is described in [8] or in Czech in [3]. We have implemented this algorithm in polynomial time $\mathcal{O}(n^4)$, where n is number of vertices of graph, that is $|e| + |f|$.

2.8 Evaluation

To evaluate automatic word alignment, the reference manual alignment in WA testing data provided in the Canadian Hansard Corpus is used. Commonly, the aligned pairs are marked as sure or possible. Sure alignments are a subset of possible alignments. However, the maximum-weight edge cover method only produces word alignment pairs without distinction, therefore we consider every word-aligned pair in automatic alignment as sure.

We used the evaluation proposed in [6].

$$recall = \frac{|A \cap S|}{|S|}$$

$$precision = \frac{|A \cap P|}{|A|}$$

$$aer = 1 - \frac{|A \cap S| + |A \cap P|}{|A| + |S|}$$

where A is a set of machine made alignments, S is a set of sure human made alignments and P is a set of possible human made alignments.

Chapter 3

Implementation details

3.1 Platform and programming language

When designing the application, some decisions had to be made to fulfil the needs of the project.

We decided to use the programming language `C++`. The reason for this decision is that `C++`, as an object oriented language, provided us with means to design flexible application class model, which could be easily extended and changed for various experiments. Moreover, thanks to some low-level constructions allowed in this language, We could optimize the speed and memory allocation strategy. To implement most of data structures, we used the `stdlibc++` (STL) library [2]. The program was developed on Linux platform using free `C++` IDE Code::Blocks [1]. For some tasks we used perl scripts and bash scripts.

The application structure description and description of classes can be found in enclosed DVD, as well as the application itself. Here we will describe only the most interesting parts.

3.2 Database

The database of word pair features is implemented in class `CDB` with interface in `cdb.h` and implementation in `cdb.cpp`. Implementing these class methods was the most challenging part of the project from programmer's point of view. Because loading and saving the database and working with it is needed in every run of the word alignment algorithm, we had to efficiently solve the problem of processing large amount of data. It necessarily had to be fast and memory efficient.

The designed application must be able to save, treat and load features $a = f(e_i, f_j), b = (e_i, \overline{f_j}), c = f(\overline{e_i}, f_j), d = (f(\overline{e_i}, \overline{f_j}), f(e_i, *), f(*, f_j), N$. Unigram features $f(e_i, *)$ and $f(*, f_j)$ are easy to maintain, because it requires memory space linear in number of words. These two values are saved in `STL maps`, which are in fact red-black trees, so for a concrete word it takes $\mathcal{O}(\log(n))$ time to find the value $f(e_i, *)$ or $f(*, f_j)$, where n is number of words in the source or target language, respectively.

But to maintain word pair features a, b, c, d, we need memory space quadratic in number

of words. The straightforward implementation, for example saving the data in a two-dimensional array (matrix), would result in large memory consumption and slowdown. Therefore we must find some way to optimize the database.

1. The key observation, which allowed us to reduce the memory requirements, is that only a small fragment of bigrams has $a \neq 0$. In Canadian Hansard Corpus, only about 0.6% bigrams has $a \neq 0$.
2. $b = f(e_i, *) - a$
3. $c = f(*, f_j) - a$
4. $d = N - a - b - c$, where N is number of all bigrams

Therefore, we do not have to save the values b, c, d at all, and we have to save the value a only for those bigrams, for which $a \neq 0$. We will not use a two-dimensional array (matrix), but some sparse structure.

The request for a, b, c, d for a concrete bigram will be served thus: We search requested bigram in our structure. If the requested bigram is found, we use a saved in the structure. If the bigram is not found in the structure, we assume that $a = 0$. The rest, b, c, d is estimated using N, $f(e_i, *)$, $f(*, f_j)$.

This simplification resulted in significant memory saving. But when using the `STL` structures, the performance was still unsatisfactory. We therefore had to create specific structure. For each source word, we save the a values of target words in a separate-chaining dynamic hash table with a simple hash function

```
#define HASH(target_word_index) ((target_word_index) & (capacity-1))
```

where `capacity` is actual hash table capacity (size) and `&` is binary `and`. The `CHashTable` class interface can be found in `chashtable.h` and implementation in `chashtable.cpp`.

We also optimized the hash table memory allocation. Because allocating memory for every single hash table entry would result in slowdown and memory fragmentation, we created a memory allocation manager (class `CHashNodeAllocator` in `chashtable.cpp`), which allocates larger blocks of memory at once.

The database can be loaded and saved either in text mode or in binary mode. When using the text mode, the database is loaded and saved as text file and is human readable. This is comfortable for debugging, feature selection and linguistic research, but it is a slower and memory consumpting way. In binary mode, the database is loaded and saved as binary file. In binary mode, the database for Canadian Hansard Corpus has 110 MB and the loading takes 6.3s, in the text mode the database has 631 MB and the loading takes 61.1s. This was measured on personal computer, AMD Athlon(tm) 64 Processor 3700+, with 1GB RAM.

3.3 Graph

The word alignment, or maximum-weight edge cover, is implemented in class `CGraph` with interface in `cgraph.h` and implementation in `cgraph.cpp`. The complete bipartite graph

is represented by two-dimensional array A, assigned with source language words in the first dimension and target language words in the second dimension. In A_{ij}, there is association measure of pair e_i, f_j.

Chapter 4

Experiments and results

In the testing experiments, we used 337 testing sentences in WA testing data from the Canadian Hansard Corpus with English as the source language and French as the target language. Where there was more than one feature used, we trained the SVM parameters γ and C using 137 word-aligned sentences in WA training data. In all cases, we trained the value of null edges. We carried out these experiments:

1. In the first experiment, we weighted the edges only with the feature normalized expectation ([7].7)

$$\frac{2f(e_i, f_j)}{f(e_i, *) + f(*, f_j)}$$

2. In the second experiment, we added other association measures (1..20).
3. In this experiment, we added the feature of relative distance of words e_i, f_j: $\text{abs}(\frac{i}{|e|} - \frac{j}{|f|})$.
4. In the last experiment, we used all features 1..34.

The results of these experiments are shown in Figure 4.1.

nr.	**experiment**	**null**	γ	C	**recall**	**precision**	**aer**
1.	normalized expectation only	0.853	–	–	55.07	53.36	45.95
2.	AMs	0.8121	8	2	56.52	77.54	33.09
3.	AMs + distance	0.8578	32	0.125	69.76	83.39	22.97
4.	AMs + linguistic features	0.843	8	0.125	80.70	82.23	**18.41**

Figure 4.1: Results

It is very interesting to look at the progress between the experiments. We chose testing sentence pair "this will result in tremendous savings ." and "cela va apporter de les économies énormes ." to show how the graph edge weights changed as we added more features and how the improvement of edge weights estimation influenced the score.

	cela	va	apporter	de	les	économies	énormes	.
this	0.00571	0.00146	0.00028	0.01666	0.01371	0.00006	0.00009	0.01563
will	0.00326	0.00528	0.00031	0.01043	0.01081	0.00013	0.00009	0.00995
result	0.00080	0.00043	0.00016	0.00072	0.00071	0.00028	0.00019	0.00057
in	0.00277	0.00098	0.00025	0.03999	0.03324	0.00010	0.00009	0.02620
tremendous	0.00017	0.00010	0.00010	0.00012	0.00012	0.00022	0.00491	0.00013
savings	0.00015	0.00013	0.00002	0.00015	0.00018	0.01673	0.00026	0.00010
.	0.00394	0.00099	0.00024	0.05444	0.04101	0.00010	0.00010	0.05110

human sure alignment
human possible alignment
automatic alignment

Figure 4.2: Experiment 1 – normalized expectation (null weight 0.04644)

In the first experiment, Figure 4.2, we can already see quite good word alignment covering the weighted bipartite graph ($aer = 45.95$). The lexical pairs *tremendous – énormes* and *savings – économies* were correctly aligned. As we can see, the edge weight between *tremendous – énormes* is far the best among the others so no other alignment could be made, and similarly for the second pair.

But there are many mistakes. The mistake *this – "."* and *"." – de* is quite common in other sentences and is a result of using the feature normalized expectation. This feature depends on value $f(e_i, f_j)$, that is the number of bigrams, where words e_i and f_j co-occurred in aligned sentences. The problem is that any word often co-occurs with *"."*, because it marks the end of sentence.

	cela	va	apporter	de	les	économies	énormes	.
this	-29.3	-43.2	-53.8	-38.7	-42.0	-65.5	-60.0	-37.3
will	-38.5	-16.9	-50.8	-41.8	-39.8	-52.6	-59.1	-40.9
result	-39.7	-45.9	-52.4	-47.5	-50.1	-29.2	-36.4	-54.7
in	-48.3	-53.6	-56.2	-28.9	-31.2	-56.8	-58.6	-35.9
tremendous	-47.7	-54.3	-46.7	-53.8	-56.3	-23.8	12.6	-55.7
savings	-50.2	-49.8	-75.1	-47.5	-47.3	22.9	-20.6	-59.6
.	-41.6	-53.7	-56.8	-28.6	-31.9	-57.5	-57.3	-22.1

Figure 4.3: Experiment 2 – AMs (null weight 4.4)

In the second experiment, Figure 4, thanks to more association measures, which estimate the association differently, the edge weights are more exact and the mistake of aligning words to *"."* has disappeared. Surprisingly, precision improved from 53.36 to 77.54, while recall is still low – only 56.52.

	cela	**va**	**apporter**	**de**	**les**	**économies**	**énormes**	**.**
this	92.8	4.5	-73.9	-55.4	-108.4	-237.3	-260.8	-217.4
will	15.6	124.3	-17.3	-19.5	-55.0	-149.2	-211.4	-182.9
result	-40.1	-20.0	-2.0	1.6	-47.5	-29.0	-93.6	-185.8
in	-109.4	-87.1	-57.0	86.5	70.1	-67.0	-113.9	-69.5
tremendous	-163.8	-144.9	-80.8	-56.8	-22.8	81.0	161.0	-98.1
savings	-219.0	-177.6	-220.3	-85.3	-42.7	224.4	97.5	-63.0
.	-229.8	-229.2	-200.5	-53.9	-25.7	-80.6	-38.7	118.2

Figure 4.4: Experiment 3 – AMs + distance (null weight 155.4)

After adding the feature of distance, Figure 4, the score improved very much, both in recall, which improved by 13.24% to 69.76, and precision, which improved by 5.85% to 83.39. This is not really surprising, because it is well known that English and French have similar word order. In this sentence particularly we cannot see any significant progress, but we could mention the edge *this – cela* which had, thanks to short relative distance (both words are the first in their sentences) received higher weight and therefore it was aligned.

	cela	**va**	**apporter**	**de**	**les**	**économies**	**énormes**	**.**
this	180.2	87.9	5.7	33.0	6.2	-129.4	-146.7	-354.2
will	121.6	198.3	56.1	64.0	41.9	-55.8	-110.5	-324.6
result	63.7	50.2	86.1	74.8	57.6	64.8	14.7	-349.0
in	-13.8	28.1	-76.1	180.9	153.7	-80.5	-32.2	-118.1
tremendous	-69.8	-155.5	14.4	-75.6	49.0	167.8	248.3	-291.2
savings	-114.2	-82.7	-110.3	-14.5	39.6	298.0	188.4	-247.6
.	-365.1	-259.1	-372.5	-104.6	-96.0	-272.1	-225.9	326.9

Figure 4.5: Experiment 4 – AMs + linguistic features (null weight 217.1)

In the last experiment, Figure 4, we can see how the other linguistic features helped to resolve problems with punctuation and numbers. Finally, we receive the alignment *". – ."* and there are also almost no punctuation mistakes in other sentences.

In every new experiment, we managed to improve aer by adding more association measures. In the second experiment, aer improved from 45.95 by 12.96%, in the third experiment, by adding distance feature, aer improved by 10.12% and in the last experiment, by adding more linguistic features, we reach $aer = 18.41$, which is improvement by 4.56%.

It is also interesting to look at the linear coefficients $\vec{a} = (a_1, \ldots, a_{34})$, which were trained by SVM ([4]). All feature values had been scaled to interval $-1..1$ before training. The scaling was linear, from original interval, to $-1..1$. The Figure 4 shows the SVM trained coefficients for experiment 4.

All association measures (1..20) and features Distance (21), Relative length (22), Longest common substring (32) and Edit distance (33) are features of quality, which means they are real numbers from interval $-1..1$ (after scaling). The rest of features are boolean features – either true (value 1) or false (value -1 after scaling).

Most features add positive value to the edge weight. Some features carry negative information. For example, for Edit distance (33): The largest the edit distance of words, the more penalized edge weight. Also boolean information Number and non-number (27) is negative: Once mixed bigram with number and non-number is found, 39.8382 is discounted.

It is remarkable how important the features Distance (21) and Punctuation and non-punctuation (28) are. The punctuation and non-punctuation word pair does not increase its weight by 76.7004, the weight is even diminished by 76.7004, so relatively its weight is decreased by $2 \cdot 76.7004 = 153.4008$. It is clear that after such a restriction of the weight, the bigram can hardly be aligned.

1.	Conditional probability	4.64272	18.	Certainty factor	7.02473
2.	Pointwise mutual inform.	31.1436	19.	Added value	7.04163
3.	Mutual dependency (MD)	26.5865	20.	Klosgen	11.1818
4.	Log frequency biased MD	18.0913	21.	Distance	−139.326
5.	Normalized expectation	5.86988	22.	Relative length	−19.4866
6.	Jaccard	4.66795	23.	Short and long	−40.1249
7.	First Kulczynsky	3.41223	24.	Both short	2.8189
8.	Second Kulczynsky	4.68653	25.	Same tokens	20.4684
9.	Yulle's ω	44.6386	26.	Both upper	10.4421
10.	Yulle's Q	55.5044	27.	Number and non-number	−39.8382
11.	Driver-Kroeber	6.10513	28.	Punct. and non-punct.	−76.7004
12.	Pearson	6.02892	29.	Not the same punct.	−4.37681
13.	Baroni-Urbani	69.3136	30.	The same punctuation	6.96838
14.	Braun-Blanquet	5.43717	31.	Same number	4.25
15.	Simpson	4.02532	32.	Longest common substr.	19.5312
16.	T combined cost	14.1037	33.	Edit distance	−35.5207
17.	Confidence	4.02532	34.	Dictionary	20.4525

Figure 4.6: SVM trained linear coefficients for features in experiment 4

Chapter 5

Conclusion

We have implemented a word alignment method based on discriminative approach. The main advantages of this method are:

- effective algorithm – Once we have trained the coefficients of features, the repeated run of the algorithm is polynomial in number of words ($\mathcal{O}(n^4)$) which makes the algorithm comfortable to use.
- easy addition of new features – To add new features, one has to train new coefficients using SVM (such as [4]).

Flexible object oriented implementation allows a user to easily adapt the application behaviour, for example changing input/output formats, comfortably add other features, etc. Moreover, the careful effective implementation makes the application suitable for extensive research using an ordinary personal computer.

We have carried out some basic experiments and have shown how to improve the results by adding more features, such as preventing the punctuation mistakes. The best result we reached was $aer = 18.41$. However, by careful consideration and adding suitable features, this result could be improved.

Bibliography

[1] Code::blocks (http://www.codeblocks.org).

[2] libstdc++ (http://gcc.gnu.org/onlinedocs/libstdc++/latest-doxygen/index.html).

[3] Jiří Demel. *Grafy a jejich aplikace*. Academia, Praha, 2002.

[4] Thorsten Joachims. Making large-scale support vector machine learning practical. In A. Smola B. Schölkopf, C. Burges, editor, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1998.

[5] Robert C. Moore. Association-based bilingual word alignment. In *Proceedings of the ACL Workshop on Building and Using Parallel Texts*, Ann Arbor, 2005.

[6] Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–21, 2003.

[7] Pavel Pecina and Pavel Schlesinger. Combining association measures for collocation extraction. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, Sydney, 2006.

[8] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer-Verlag, 2003.

[9] Ben Taskar, Simon Lacoste-Julien, and Dan Klein. A discriminative matching approach to word alignment. In *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 73–80, Vancouver, British Columbia, Canada, 2005.

Chapter 6

Appendixes

The enclosed DVD contains the application, the complete source codes, perl and bash scripts, programmers documentation and this thesis in PDF format. The software uses the MIT License.