
Technical Report

Alignment-Based Learner

version 1.0 alpha

Reference Guide¹

Jeroen Geertzen
j.geertzen [at] uvt.nl

Language and Information Science
Tilburg University, P.O. Box 90153
5000 LE Tilburg, The Netherlands

Menno M. van Zaanen
menno [at] ics.mq.edu.au

Division of Information and Communication Sciences
Department of Computing
Macquarie University
2109 Sydney, NSW, Australia

July 2006

¹This document is available from:
<http://www.ics.mq.edu.au/~menno/research/software/abl/>

Contents

1	License terms	5
2	Installation	7
3	Software usage	8
3.1	Using ABL	8
3.1.1	Alignment learning	8
3.1.2	Clustering	9
3.1.3	Selection learning	10
3.2	Practical issues	11
3.2.1	Using pipes	11
3.2.2	Supported character sets	11
3.2.3	Empty hypotheses	11
4	Alignment learning methods	13
4.1	Edit distance based methods	13
4.1.1	default	13
4.1.2	biased	14
4.1.3	all	14
4.2	Suffix tree based methods	14
4.2.1	st1	14
4.2.2	st2	15
4.2.3	st3	15
4.2.4	st4	15
5	Selection learning methods	16
5.1	first	16
5.2	leaf	16
5.3	branch	16
6	Data formats and command-line options	17
6.1	Data formats	17
6.2	Command-line options	18
6.2.1	abl_align	18
6.2.2	abl_cluster	19
6.2.3	abl_select	19

7	Optimizing ABL usage	20
7.1	Edit-distance based: exhaustive or smart?	20
7.2	Generating all hypotheses or projecting on existing hypotheses . .	22
8	Source code organization	23

Preface

Alignment-Based Learning (ABL) is a symbolic grammar inference framework that has successfully been applied for several unsupervised machine learning tasks in Natural Language Processing (NLP). Given sequences of symbols only, a system that implements ABL induces structure by aligning and comparing the input sequences. As a result, the input sequences are augmented with the induced structure.

The first phase in ABL, *alignment learning*, builds a search space of possible structures, called *hypotheses*, by comparing the sequences and clustering subsequences that share similar context. In the second phase, *selection learning*, the most probable combination of hypothesized structures are selected in an Expectation-Maximalization search.

We believe the implementation of ABL presented here will be beneficial for further explorations in applying alignment-based grammar inference and will turn out to be useful for unsupervised machine learning research in NLP and other areas.

This document is structured as follows. In the first chapter, the terms of the license are given to which you are allowed to use this ABL implementation. In the subsequent chapters, instructions are given how to install the package on your computer (Chapter 2) and how to run the software and run experiments (Chapter 3). The instructions are accompanied with examples for inducing natural language syntax. This document ends with a global description of how the C++ source code is organized to support extensions and modifications.

We would appreciate it if you would send bug reports, ideas for extensions, extensions that could be included in future releases, as well as comments and feedback on this manual, to menno@ics.mq.edu.au.

Chapter 1

License terms

Downloading and using the ABL software implies that you accept the following terms:

Macquarie University (henceforth "Licensers") grants you, the registered user (henceforth "User") the non-exclusive license to download a single copy of the ABL implementation software code and related documentation (henceforth jointly referred to as "Software") and to use the copy of the code and documentation solely in accordance with the following terms and conditions:

1. The license is only valid when you register as a user. If you have obtained a copy without registration, you must immediately register by sending an e-mail to menno@ics.mq.edu.au.
2. User may only use the Software for educational or non-commercial research purposes.
3. Users may make and use copies of the Software internally for their own use.
4. Without executing an applicable commercial license with Licensers, no part of the code may be sold, offered for sale, or made accessible on a computer network external to your own or your organization's in any format; nor may commercial services utilizing the code be sold or offered for sale. No other licenses are granted or implied.
5. Licensers have no obligation to support the Software it is providing under this license. To the extent permitted under the applicable law, Licensers are licensing the Software "AS IS", with no express or implied warranties of any kind, including, but not limited to, any implied warranties of merchantability or fitness for any particular purpose or warranties against infringement of any proprietary rights of a third party and will not be liable to you for any consequential, incidental, or special damages or for any claim by any third party.
6. Under this license, the copyright for the Software remains the property of the Centre for Language Technology at Macquarie University. Except as specifically authorized by the above licensing agreement, User may not use, copy or transfer this code, in any form, in whole or in part.
7. Licensers may at any time assign or transfer all or part of their interests in any rights to the Software, and to this license, to an affiliated or unaffiliated company or person.
8. Licensers shall have the right to terminate this license at any time by written notice. User shall be liable for any infringement or damages resulting from User's failure to abide by the terms of this License.

9. In publication of research that makes use of the Software, a citation should be given of: *Menno van Zaanen and Jeroen Geertzen (2006). ABL: Alignment-based Learner, version 1.0, Reference Guide, Available from <http://www.ics.mq.edu.au/~menno/research/software/abl/>*
10. For information about commercial licenses for the Software, contact `menno@ics.mq.edu.au`, or send your request in writing to:

Menno van Zaanen
Division of Information and Communication Sciences (ICS)
Department of Computing
Macquarie University
2109 North Ryde, (Sydney), NSW
Australia

Chapter 2

Installation

The ABL software package can be found at:

```
http://www.ics.mq.edu.au/~menno/research/software/abl/
```

Before downloading the file `abl-1.0.tar.gz`, fill in the registration information and accept the license agreement. The file is a gzipped tar archive that contains the ABL software package, the license, and documentation.

To install the package on your UNIX based system, issue the following command from the command line prompt (`>`) :

```
> tar xvzf abl-1.0.tar.gz
```

As a result, a directory `abl-1.0` will be created. Enter this directory and compile the software, assuming that you have `autoconf` and `automake` on your system:

```
> cd abl-1.0/  
> ./configure  
> make  
> make install
```

If the process was successful, three new commands are available: `abl_align`, `abl_cluster`, and `abl_select`. Additionally, within the directory `abl-1.0`, a `doc/` directory has been created with a postscript version of this document.

The software is now ready to be used. For each program, help can be printed on the command line by using it with the switch `-h`. For any questions on compiling, installing, or using the software, please send a message to `menno@ics.mq.edu.au`.

Chapter 3

Software usage

This chapter shows how to use the Alignment-Based Learning (ABL) software package by discussing the steps in the grammar induction process, the commands to be issued, how to select specific algorithms and metrics by setting the right parameters, and other practical issues. The instructions are exemplified by simple natural language examples.

3.1 Using ABL

Alignment-based learning [van Zaanen, 2002] is a two phase algorithm with different possible instantiations. In the first phase, called *alignment learning*, sequences of symbols (in our examples sentences of words) are compared to find possible constituents (hypotheses). Directly after alignment learning, it is required to cluster hypothesized constituents with similar context together. We will refer to this step as *clustering*. In the second phase, called *selection learning*, the most probable combination of hypotheses are selected to be the induced constituents.

3.1.1 Alignment learning

Let us consider the following example-corpus of plain text sentences with one sentence per line in a file called `ex01.txt` :

```
she is smart
she is tired
```

To do alignment learning on this file, we can give the following command on the command line prompt:

```
> abl_align -a wm -p u -i ex01.txt -o ex01.txt.aligned
```

The `-i` parameter and `-o` parameter specify input and output files respectively. With the `-a` parameter, the alignment algorithm or method is selected¹. In this case, by providing `wm` we use unbiased string edit distance [Wagner and Fischer, 1974]. With the `-p` parameter we select which part of the

¹See Section 4 for an overview and explanation of all alignment methods.

sentence to consider as possible constituents, which is usually the unequal parts (-p u). The resulting output², for each sentence will be the sentence itself followed by one or more hypotheses. The sentence and hypotheses are delimited by the string '@@@':

```
she is smart @@@ (0,3,[0])(2,3,[1])
she is tired @@@ (2,3,[1])(0,3,[0])
```

As can be observed in the output, an hypothesis or constituent is formatted in the following way:

```
(A,B,[C])
```

where non-terminal C spans from position A in the sentence to position B in the sentence. As such, the output in file `ex01.txt.aligned` can also be visualized as follows:

```
(she is (smart)1)0
(she is (tired)1)0
```

As can be observed, those parts that are unequal are grouped into the same equivalence class. In the remainder of this document we will use the parenthesized representation instead of the ABL representation for the sake of easy visualization.

When using a large corpus as input in combination with a string edit distance based algorithm on machines that are low on internal memory, the switch `-m` can be used to preserve memory use at the cost of execution speed.

3.1.2 Clustering

The string edit distance based algorithms in the alignment learning phase work by pairwise comparison of sentences in the corpus. As a result of these comparisons, hypotheses can have multiple non-terminals that all share the same context. What is needed is a *clustering* step that clusters these same-context-non-terminals to a single non-terminal. E.g. when we apply alignment learning on the following input of file `ex02.txt`:

```
she works well
she sings well
she listens well
```

we will obtain the following structure:

```
(she (works)1,2 well)0
(she (sings)1,3 well)0
(she (listens)2,3 well)0
```

In this output hypotheses 1,2 and 1,3 and 2,3 share the same context. We can cluster all non-terminals that share the same context with the following command:

²In ABL format

```
> abl_cluster -i ex01.txt.aligned -o ex01.txt.clustered
```

in which the `-i` and `-o` parameter specify input and output files respectively. After clustering, the resulting file, `ex01.txt.clustered`, will have the following content:

```
(she (works)1 well)0
(she (sings)1 well)0
(she (listens)1 well)0
```

3.1.3 Selection learning

The set of hypotheses that are generated during alignment learning contain hypotheses that are unlikely to be correct constituents. We also assume the underlying grammar that is induced to be context-free, which implies that overlapping constituents are unacceptable. In the selection learning phase, the most probable combination of hypotheses from all hypotheses that are generated by the alignment learning phase is selected according to a specific metric.

Let us consider the following sentences in `ex03.txt`:

```
she runs from CBD to Bondi
he drives from CBD to Bondi
he walks terribly slow
```

the resulting output after alignment learning and clustering (in `ex03.txt.clustered`) is as follows:

```
((she runs)1 from CBD to Bondi)0
((1 he (2 drives)1 from CBD to Bondi)2)0
(he (walks terribly slow)2)0
```

The underlined words indicate those parts that are shared context. As can be observed, the hypotheses one and two are overlapping in the second sentence. We can apply selection learning on file `ex03.txt.clustered` by issuing the following command:

```
> abl_select -s b -i ex03.txt.clustered -o ex03.txt.selected
```

in which the `-i` and `-o` parameter specify input and output files respectively. With the `-s` parameter, we can specify the selection method³. When we have applied selection learning, the resulting output in `ex03.txt.selected` contains for each sentence constituents that are not overlapping:

```
((she runs)1 from CBD to Bondi)0
((1 he drives)1 from CBD to Bondi)0
(he (walks terribly slow)2)0
```

As can be observed in sentence two, hypothesis two is discarded as a constituent. When using a large corpus as input on machines that are low on internal memory, the switch `-m` can be used to preserve memory use at the cost of execution speed.

³See Section 5 for an overview and explanation of all alignment methods.

3.2 Practical issues

3.2.1 Using pipes

All three programs, `abl_align`, `abl_cluster`, and `abl_select` are able to both read/write from files and read/write to standard input and standard output. The latter possibility allows these programs nicely to be put in sequence by using pipes. For example, the following command:

```
> abl_align -a wm -p u | abl_cluster | abl_select -s b
```

allows to type on standard input (close with CTR-d) after which the induced structure is printed on the standard output. Likewise, the following commands:

```
> cat in.txt | abl_align -a wm -p u | abl_cluster \  
> | abl_select -s b > out.txt
```

applies ABL on file `in.txt` and saves the induced structure in file `out.txt`. To get to know exactly what is generated in each step, we can use the `-v` or `--verbose` switch on each program. Apart from the data to the file `out.txt`, status information is written to the terminal. E.g.:

```
./abl_align      : # sentences loaded           : 61  
./abl_align      : # hypotheses generated      : 520  
./abl_align      : # seconds execution time    : 0.02  
./abl_cluster    : # sentences loaded           : 61  
./abl_cluster    : # hypotheses loaded         : 520  
./abl_cluster    : # unique non-terminals input : 1058  
./abl_cluster    : # unique non-terminals output : 145  
./abl_cluster    : # hypotheses clustered       : 913 (0.86)  
./abl_cluster    : # seconds execution time    : 0.03  
./abl_select     : # sentences loaded           : 61  
./abl_select     : # hypotheses loaded         : 520  
./abl_select     : # hypotheses selected        : 361  
./abl_select     : # seconds execution time    : 0.02
```

3.2.2 Supported character sets

The ABL software supports the use of 7-bit (ASCII) and 8-bit (extended ASCII, ISO-8859) ASCII characters. There is no *special* support for handling unicode characters. Words in sentences can contain any character from these encodings, except for spaces (word delimiters), newlines (sentence delimiters), or the string '@@@' (sentence-hypotheses delimiter).

3.2.3 Empty hypotheses

By default, the alignment-learning phase generates so called *empty hypotheses*: hypotheses that do not span any word. Consider for example:

```
(she is (very)1 ill)0  
(she is ()1 ill)0
```

in which hypothesis 1 is empty in the second sentence. Depending on the task ABL is used in, it might not be desired to have empty spans being generated. In these cases, use the `-e` switch on `abl_align`. Taking the sentences from the previous example, the output of `abl_align -a wm -p o -e` would be:

```
(she is (very)1 ill)0  
(she is ill)0
```

Chapter 4

Alignment learning methods

This version of the ABL implementation supports seven alignment learning methods: three edit distance based methods and four suffix tree based methods. For a corpus of n sentences, the algorithmic complexity of the edit distance based methods generally is $O(n^2)$, whereas the algorithmic complexity of the suffix tree based methods generally is $O(n)$. However, the alignments obtained by edit distance based methods seem to be better biased towards natural language syntax (see [Geertzen and van Zaanen, 2004]).

Each method available for alignment learning will be explained briefly in the following sections. For more details about the edit distance based methods and the suffix tree based methods, see [van Zaanen, 2002] and [Geertzen, 2003] respectively.

4.1 Edit distance based methods

Edit distance based methods that can be specified when using the command `abl_align` are : *default* (-a wm), *biased* (-a wb), and *all* (-a a). These alignment methods are all based on the edit distance algorithm [Wagner and Fischer, 1974]. This algorithm finds the minimum edit cost to transform one sentence into the other based on a predefined cost function. This function assigns a cost to each of the edit operations: insertion, deletion and substitution.

4.1.1 default

The first method, which is called *default*, uses a cost function that makes the algorithm find the longest common subsequences in two sentences. As the groups of words that are not in these longest common subsequences are the words that are unequal in both sentences, they should be stored as hypotheses.

4.1.2 biased

Using the longest common subsequence to find the unequal parts of the sentences does not always result in the preferred hypotheses. Consider the following sentences:

```
... from Central Business District to Bondi
... from Bondi to Central Business District
```

When aligning these sentences using *default* alignment gives us the following output:

```
... from ()1 Central Business District ( to Bondi )2
... from (Bondi to)1 Central Business District ()2
```

which is syntactically undesirable. The preferred syntactical structure emerges when, instead of getting the longest common subsequences, we prefer to link the word "to", which would result in the following alignment:

```
... from (Central Business District)1 to (Bondi)2
... from (Bondi)1 to (Central Business District)2
```

To let the system have a preference to link "to" instead of "Central Business District" can be accomplished by biasing the cost function towards linking words that have similar relative offsets in the sentence. This alignment learning method is called *biased*.

4.1.3 all

It may be the case that the biased does not find the correct alignment. A third method tries to solve this problem by introducing all possible alignments to find hypotheses, making the assumption that during selection learning, the correct hypotheses will emerge as constituents. This alignment learning method is called *all*. Note that this method introduces considerably more hypotheses (including many overlapping ones) and puts a higher load on the selection learning phases.

4.2 Suffix tree based methods

Suffix tree based methods that can be specified when using the command `abl_align` are: *suffix tree 1* (`-a st1`), *suffix tree 2* (`-a st2`), *suffix tree 3* (`-a st3`), and *suffix tree 4* (`-a st4a`). These alignment methods are all based on building a generalized suffix tree (GST) of the corpus and combining structural information of common suffixes and prefixes.

4.2.1 st1

The location in the GST where one edge branches into multiple edges marks the position in the sentences involved where hypotheses can start. Since it is less obvious where the hypotheses should end, we assume for `st1` the pair of brackets to close at the end of the sentences. The hypotheses generated by this algorithm for the next three sentences are indicated with brackets:

she [_a is walking away]_a
 she [_a was walking]_a
 she [_a runs away]_a

4.2.2 st2

To find distinct words followed with joint words (e.g. “away” in the first and third sentence in previous section), we can construct a generalized prefix tree (GPT). By analyzing the GPT similarly to the GST in **st1**, we are able to place closing brackets. Since the opening brackets cannot be read from the GPT we assume prefix hypotheses to start at the beginning of the sentence.

4.2.3 st3

The method **st3** combines the alignments of **st1** and **st2**, such that the hypotheses generated by this algorithm for the three sentences look as follows:

[_b [_c she [_a is]_c walking]_b away]_a
 [_c she [_a was]_c walking]_a
 [_b she [_a runs]_b away]_a

4.2.4 st4

To capture hypotheses that start somewhere in the sentence and stop somewhere later in the sentence as well, we introduce **st4**, which uses both GPT and GST but additionally matches opening and closing brackets, such that the hypotheses generated by this algorithm for the three sentences look as follows:

she [_a is]_b walking]_c away
 she [_a was]_b walking
 she [_a runs]_c away

Chapter 5

Selection learning methods

This version of the ABL implementation supports three basic selection learning methods that can be specified when using the command `abl_select : first (-s f)`, `leaf (-s l)`, and `branch (-s b)`.

When using one of the probabilistic selection learning methods, it may be possible that several combinations of hypotheses have the same probability (after computing the combined probability). The system then selects one of these combinations at random.

5.1 first

The first method, *first*, is a non-probabilistic method that builds upon the assumption that a hypothesis that is learned earlier is always correct. This means that newly learned hypotheses that overlap with older ones are considered to be incorrect, and thus should be removed.

5.2 leaf

The method *leaf* computes the probability of a hypothesis by counting the number of times the particular words of the hypothesis have occurred in the learned text as a hypothesis, divided by the total number of hypotheses.

5.3 branch

In addition to only consider the words of the sentence delimited by the hypothesis as in the method *leaf*, this model computes the probability based the words of the hypothesis and its type label.

Chapter 6

Data formats and command-line options

6.1 Data formats

All ABL programs assume a single sentence (possibly augmented with hypotheses and constituents) to be on single line. As a result, input for alignment learning could look as follows:

```
she is smart
she is tired
```

The output of `abl_align` and the input and output of `abl_cluster` and `abl_select` are expected to be in *ABL format*. For each line is specified:

- *the plain sentence*, consisting of 1 or more words;
- followed by a *separator string* '@@@';
- followed by one of more *constituents*. A constituent is formatted in the following way:

```
(A,B,[C])
```

where non-terminal *C* spans from position *A* in the sentence to position *B* in the sentence.

For instance:

```
she is smart @@@ (0,3,[0])(2,3,[1])
she is tired @@@ (2,3,[1])(0,3,[0])
```

The two sentences in ABL format can also be visualized as follows:

```
(she is (smart)1)0
(she is (tired)1)0
```

6.2 Command-line options

Parameters to structure learning with ABL can be specified on the command-line. Typing for any of the ABL programs the argument '-h' will provide you with an overview of possible parameters to specify. An elaborated overview of all possible parameters is presented in this section.

Parameters that each program in the package have in common are the following:

- i --input=FILE Name of the input file. When this parameter is omitted, input is expected on STDIN.
- o --output=FILE Name of the output file. When this parameter is omitted, output will be given on STDOUT.
- d --debug Prints information about internal processes useful for debugging purposes.
- v --verbose Show details about the processing that might be interesting to the user, such as execution time and statistics.
- V --version Shows version of the program and subsequently exits.

6.2.1 abl_align

The program for the alignment learning phase, `abl_align`, has the following *specific* parameters:

- a --align The method to use for alignment learning. The available methods are:
 - wagner_min, wm
wagner_fisher edit distance with default gamma (see 4.1.1)
 - wagner_biased, wb
wagner_fisher edit distance with biased gamma (see 4.1.2)
 - all, aa, a
all possible alignments (see 4.1.3)
 - left, l
only produce left branching trees.
 - right, r
only produce right branching trees.
 - both, b
produce left and right branching trees.
 - suffix_tree_1, st1
suffix tree based alignment, method 1 (see 4.2.1)
 - suffix_tree_2, st2
suffix tree based alignment, method 2 (see 4.2.2)
 - suffix_tree_3, st3
suffix tree based alignment, method 3 (see 4.2.3)
 - suffix_tree_4, st4
suffix tree based alignment, method 4 (see 4.2.4)

-p --part Part of the sentences that should be used as hypotheses.
 (defaults to unequal)
 - equal, e
 equal parts
 - unequal, u
 unequal parts
 - both, b
 equal and unequal parts
-s --seed Seed (for the both alignment type)
-t --time Number of seconds between each checkpoint
-c --check prefix of checkpoint files (default is abl_align)
-e --excl_empty Do not generate hypotheses that span 0 words (see 3.2.3)
-n --nomerge Do not try to merge hypotheses (see 7.2)
-x --exhaustive Compare exhaustively each possible sentence pair (see 7.1)

6.2.2 abl_cluster

The program for clustering non-terminals, **abl_cluster**, has no *specific* parameters.

6.2.3 abl_select

The program for the selection learning phase, **abl_select**, has the following *specific* parameters:

-s --select The selection method:
 - first, f
 earlier learned constituents are correct (see 5.1)
 - leaf, l
 terms selection method (see 5.2)
 - branch, b
 const selection method (see 5.3)

Chapter 7

Optimizing ABL usage

Depending on the characteristics of the training data, several options in alignment learning and selection learning make it possible to reduce the execution time of the algorithm. In the following two sections, the effects of several options are discussed.

7.1 Edit-distance based: exhaustive or smart?

When using *edit distance* based alignment learning, each sentence in the training data can be compared in pairwise fashion with each other sentence in the training data:

Algorithm 1 pairwise comparison of all sentences in training data.

Require: S : ordered set of training sentences

Require: s_i : i -th sentence

```
for all sentence  $s_i \in S$  do
  for all sentence  $s_j \in S$  where  $j > i$  do
    compare( $s_i, s_j$ )
  end for
end for
```

Execution of Algorithm 1 results in $N(N - 1)/2$ pairwise comparisons. Because comparing sentences that have no words in common will never result in hypothesis generation, the number of pairwise comparisons can be reduced:

When comparing Algorithm 1 with Algorithm 2 in terms of runtime complexity, we can see that in Algorithm 2 reduction of the number of pairwise comparisons is achieved at the cost of indexing and listing sentences to compare with, two preprocessing steps which are both linear in corpus size. The program `abl_align` uses Algorithm 2 by default. However, there are two reasons why Algorithm 1 might be better used instead:

1. *small training set*. If the training set is relatively small, or the sentences in the training set have few overlap, the cost of considering twice all sentences in the set does not outweigh the advantage of fewer comparison steps.

Algorithm 2 smart comparison of all sentences in training data.

Require: S : set of training sentences**Require:** s_i : i -th sentence

```
for all sentence  $s_i \in S$  do
  index_words( $s_i$ )
end for
for all sentence  $s_i \in S$  do
  make_compare_list( $s_i$ )
end for
for all sentence  $s_i \in$  compare_list( $s_i$ ) do
  for all sentence  $s_j \in S$  where  $j > i$  do
    compare( $s_i, s_j$ )
  end for
end for
```

2. *conservative memory usage* When conservative computer memory usage is important, not creating the compare lists for the sentences reduces memory usage.

The second algorithm can be selected by specifying the switch `-x` or `--exhaustive` for `abl_align`. To give an indication of the difference in execution time between both algorithms and hence which method applies best for which case, let us consider the results of alignment learning on 6,797 sentences of the OVIS corpus [Strik et al., 1997] and 44 424 sentences of the WSJ corpus [Marcus et al., 1993]:

Table 7.1: Number of compares on 2000 sentences OVIS

	OVIS	WSJ
# sentences	6 797	44 323
# comparisons Algorithm 1	23 096 206	982 242 003
# comparisons Algorithm 2	6 240 829	749 019 471
Algorithm 2 / Algorithm 1	0.11	0.76

For the OVIS sentences, almost 90 % of pairwise comparisons turned out to be unnecessary. For the WSJ, despite having longer and more complicated sentences, there is 24 % that could be avoided. To give insight in *when* comparing all sentence pairs can be useful, let us look to the execution speed as function of the time for both algorithms (Figure 7.1, left).

For the OVIS sentences Algorithm 2 turns out to be the most efficient already right from the beginning. This has also been confirmed for larger natural language corpora such as the WSJ, we can conclude the same. This makes switch `-x` only favorable when we need to be conservative in computer memory usage.

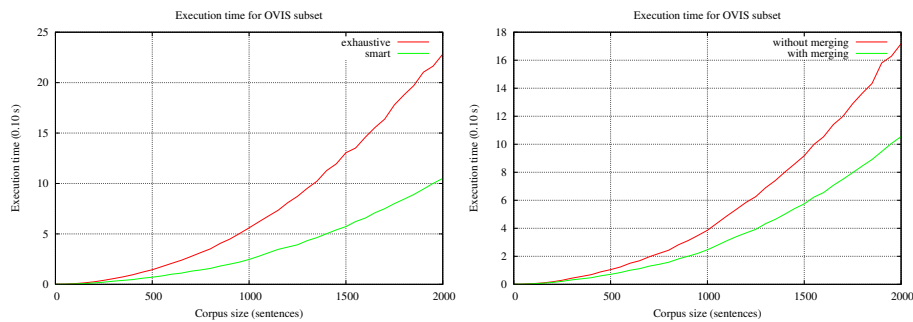


Figure 7.1: Execution time for 2000 sentences OVIS.

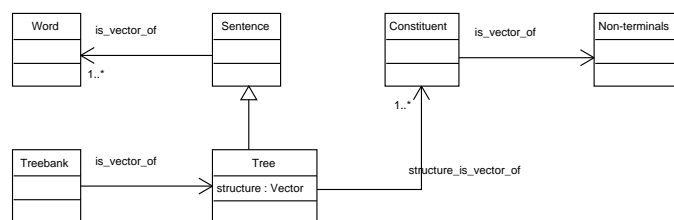
7.2 Generating all hypotheses or projecting on existing hypotheses

By default, hypotheses that are about to be introduced for a sentence pair are compared with already found hypotheses for this sentence pair and merged when they span similar words. This requires more steps in computation, but gets compensated by quite smaller data structures to store the generated hypothesis in. This optimization at the cost of memory can be switched off with the `-n / --nomerge` switch. The difference in execution time between merging and not merging is expressed in the right graph in Figure 7.1. From the graph we can observe that for the OVIS corpus execution time decreases considerably when attempting to merge hypotheses.

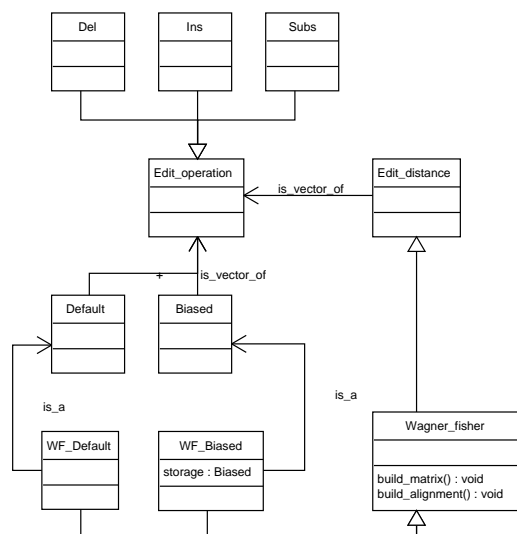
Chapter 8

Source code organization

The main classes in the source code are organized as follows. Classes that support reading/writing and operating on treebanks:



Classes that support edit distance based alignment methods:



Bibliography

- [Geertzen, 2003] Geertzen, J. (2003). String alignment in grammatical inference: what suffix trees can do. Technical Report ILK-0311, ILK, Tilburg University, Tilburg, The Netherlands.
- [Geertzen and van Zaanen, 2004] Geertzen, J. and van Zaanen, M. M. (2004). Grammatical inference using suffix trees. In *Proceedings of the 7th International Colloquium on Grammatical Inference (ICGI)*, pages 163–174, Athens, Greece.
- [Marcus et al., 1993] Marcus, M. P., Santorini, B., and Marcinkiewicz, M. (1993). Building a large annotated corpus of English: the Penn Treebank. *Computational linguistics*, 19:313–330. Reprinted in Susan Armstrong, ed. 1994, *Using large corpora*, Cambridge, MA: MIT Press, 273–290.
- [Strik et al., 1997] Strik, H., Russel, A., van den Heuvel, H., Cucchiarini, C., and Boves, L. (1997). A spoken dialog system for the dutch public transport information service. *International Journal of Speech Technology*, 2(2):119–129.
- [van Zaanen, 2002] van Zaanen, M. M. (2002). *Bootstrapping Structure into Language: Alignment-Based Learning*. PhD thesis, University of Leeds, Leeds, UK.
- [Wagner and Fischer, 1974] Wagner, R. A. and Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21(1):168–173.