Grammatical Inference Using Suffix Trees

Jeroen Geertzen[1] and Menno van Zaanen[2]

[1] ILK, Computational Linguistics
Tilburg University
Tilburg, The Netherlands
J.Geertzen@uvt.nl
[2] ILK, Computational Linguistics
Tilburg University
Tilburg, The Netherlands
M.M.vanZaanen@uvt.nl

Abstract. The goal of the Alignment-Based Learning (ABL) grammatical inference framework is to structure plain (natural language) sentences as if they are parsed according to a context-free grammar. The framework produces good results even when simple techniques are used. However, the techniques used so far have computational drawbacks, resulting in limitations with respect to the amount of language data to be used. In this article, we propose a new alignment method, which can find possible constituents in time linear in the amount of data. This solves the scalability problem and allows ABL to be applied to larger data sets.

1 Introduction

Alignment-Based Learning (ABL) is a symbolic grammar inference framework that structures plain text according to an implicit context-free grammar. The result of the process is a treebank based on the plain input text. The structure should be linguistically relevant, hence the comparison against linguistically motivated data. The framework consists of several phases, which each need to be instantiated with a specific algorithm. Previous work has shown that even ABL systems based on simple phase instantiations yield state-of-the-art results [1].

The main problem with the previous implementation of the ABL systems is that the time complexity of the first phase, that finds possible constituents, is quadratic in the size of the corpus [2]. Even though ABL can learn structure from only a few sentences, scalability is a major problem. We expect that when more data is used, more possible constituents can be found, and hence a higher accuracy can be achieved. Furthermore, when more possible constituents are found, more precise counts are available, which results in more specific statistics in the disambiguation phase.

Here, we present a completely new alignment phase, that is linear in the size of the corpus with respect to time and space complexity. This allows for grammatical induction from much larger corpora. In this article, we first give a brief overview of the ABL grammatical inference framework. Next, we describe

G. Paliouras and Y. Sakakibara (Eds.): ICGI 2004, LNAI 3264, pp. 163–174, 2004.

the new alignment learning instantiations based on suffix trees. We conclude with the results of the suffix trees instantiation applied to the ATIS and WSJ corpora and compare them against the previous implementations.

2 Alignment-Based Learning

The ABL framework consists of three distinct phases. The *alignment learning* phase tries to find possible constituents by searching for regularities within the data. The *clustering* phase groups non-terminals that occur within the same context together. The *selection learning* phase selects the best constituents among all possible constituents generated by the alignment learning phase. All of these phases will be described in some detail below.

2.1 Alignment Learning

The alignment learning phase is based on Harris' [3] linguistic notion of substitutability, which (in terms of syntax) states that if two constituents are of the same type then they can be substituted by each other. The reversed reading of Harris' implication: "if parts of sentences can be substituted by each other then they are constituents of the same type" allows us to find possible constituents, called *hypotheses*, by aligning sentences. If parts of sentences occur in the same context, they can be seen as substitutable and hence are stored as hypotheses.

Previous implementations [4] are based on the well known edit distance algorithm [5]. Each sentence from the plain text corpus is aligned to each other sentence using the edit distance algorithm. This uncovers the substitutable parts of the sentences, which are stored as hypotheses.

2.2 Clustering

The alignment learning phase introduces hypotheses with each of them having a set of non-terminal type labels attached. The clustering phase analyzes the non-terminals and clusters together (merges) non-terminals of hypotheses that occur in the same context. The current implementation of the clustering phase is relatively simple and is fast in practice.

The clustering of non-terminals is not always correct, but so far the accuracy of the non-terminal type assignment has not been researched extensively. In the future, more complex clustering phases may be devised, but the impact of the clustering phase on the rest of the system is minimal; only one selection learning instantiation may make different choices depending on the results of this phase.

2.3 Selection Learning

The selection learning phase selects the best hypotheses of those that were found by the alignment learning phase. The existing alignment learning phases generate overlapping pairs of brackets, which are unwanted if the underlying grammar is considered context-free. The selection learning phase makes sure that overlapping pairs of brackets are removed, resulting in tree structures that are comparable to derivations of a context-free grammar.

2.4 Problems

The main problem of the previous implementations of ABL is scalability [2]. The alignment learning phase compares each sentence in the corpus to all other sentences. Adding more sentences to the corpus greatly increases the execution time. This poses no problem for small corpora (of roughly up to 10,000 sentences), but learning from much larger corpora is currently not feasible.

Previous work [6] showed that ABL outperforms the EMILE grammatical inference system [7] on small corpora. However, EMILE was designed to work on larger corpora (say $> 100,000$ sentences). When we want to compare the two systems on such corpora, the scalability problem of ABL needs to be tackled.

3 Grammatical Inference with Suffix Tree Alignment

To solve the problem of complexity of the previous alignment learning instantiations, we introduce a completely new alignment learning phase, based on aligning sentences using a suffix tree as an alternative to the edit distance algorithm. Where the edit distance alignment uses a two-dimensional matrix as data structure to store information on two sentences, suffix tree alignment uses a suffix tree as an efficient data structure to represent all sentences of a corpus. By using a suffix tree, we avoid pairwise comparison of sentences and, consequently, reduce the complexity in corpus size from quadratic to linear time and space.

3.1 Suffix Trees

A suffix tree is a compact representation of all suffixes of a string in the form of a tree structure. The most apparent use of a suffix tree is in applications of string matching. For example, a substring T of length n can be matched in $O(n)$ time in a string S of length m by constructing a suffix tree for S first. This preprocessing will only take linear $O(m)$ time (and only needs to be done once). Since searching only depends on the length of the search string, suffix trees are a preferred option when S needs to be searched repeatedly. A suffix tree can be defined informally as in Definition 1.

Definition 1. *A suffix tree T for a string S (with $n = |S|$) is a rooted, labeled tree with a leaf for each non-empty suffix of S. Furthermore, a suffix tree satisfies the following conditions:*

- *Each internal node, other than the root, has at least two children;*
- *Each edge leaving a particular node is labeled with a non-empty substring of S for which the first symbol is unique among all first symbols of the edge labels of the edges leaving the node;*
- *For any leaf in the tree, the concatenation of the edge labels on the path from the root to the leaf, exactly spells out a non-empty suffix of S.*

When searching for regularities in a corpus or, more specifically, searching for substitutable parts of sentences, we are particularly interested in a suffix tree

that represents the suffixes of the sentences of a complete corpus. In other words, we use a *generalized suffix tree*, which is a suffix tree representing the suffixes of a set $\{S_1, S_2, \ldots, S_n\}$ of sentences.

An example of a generalized suffix tree for the sentences S_1=[she, was, walking], S_2=[she, is, walking, away], S_3=[she, runs, away] is depicted in Figure 1. Parts of sentences that share the same context can be identified by taking the topology of the suffix tree into account. Using this knowledge, we can introduce a new alignment method based on suffix trees, which consist on two linear steps:

1. Construct the generalized suffix tree for the corpus;
2. Employ an algorithm that finds possible constituents by taking the suffix tree topology into account.

We will describe step 1 briefly and study step 2 in more detail.

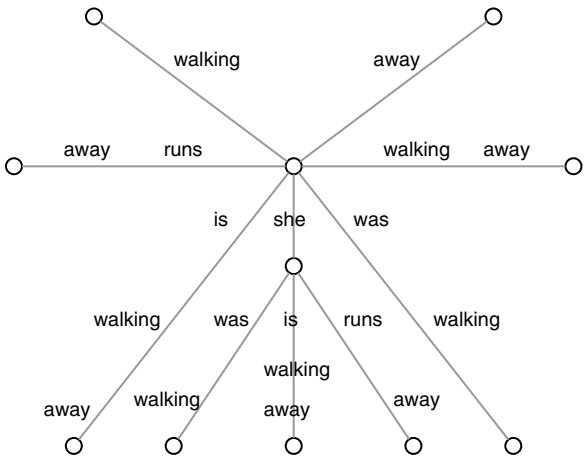

Fig. 1. Generalized suffix tree

3.2 Constructing a Generalized Suffix Tree

Currently, there are three basic algorithms to build a suffix tree: Weiner's [8], McCreight's [9], and Ukkonen's [10]. Weiner presented the first linear time suffix tree construction algorithm. McCreight gave a simpler and less space consuming version, which became the standard in research on suffix trees. In this paper, we consider Ukkonen's algorithm. In contrast to the algorithm of McCreight, it builds the tree incrementally from left to right instead of vice versa.

All m suffixes of string $S = a_1, \ldots, a_m$ with $m = |S|$ need to be present in the suffix tree. To accomplish this, each prefix ending at $S[i]$ of S, starting with $i = 1$ and incrementing i until $i = m$, is processed. For each prefix $S[1..i]$, every suffix $S[j..i]$ where $1 \leq j \leq i$ is considered. This results in algorithm 1 in pseudo-code.

Algorithm 1 Ukkonen simplified

Require:
α : string
p : point
1: **for** i **from** 1 **to** m **do** { prefix phase }
2: **for** j **from** 1 **to** i **do** { suffix phase }
3: $\alpha = S[j..i-j]$
4: $p =$ Find_end_of_path(α)
5: Apply_suffix_extension_rules(p, $S[i]$)
6: **end for**
7: **end for**

The routine *Find_end_of_path()* finds the *point* p in the current tree that indicates the end of the path from the root to the maximum prefix of α that can be represented in the tree.

Once p is determined, the routine *Apply_suffix_extension_rules()* checks whether α can be represented completely. This is the case when point p marks a leaf node and the current tree does not need to be expanded. When p marks either an internal node or a position in an edge, α cannot be represented completely and subsequently the current tree needs to be expanded to represent α as well.

For Algorithm 1 to be linear, both routines should have constant time complexity. One of the essential "tricks" to achieve this is the introduction of *suffix links*. The routine *Find_end_of_path_from_root()* traverses, for each suffix phase, through the suffix tree and considers the edge labels on the way until a path is found to the point where the suffix extension rules can be applied. Using suffix links, we only need to traverse from the root node to the leaf node in the first suffix phase of a particular prefix phase. The remaining suffixes in the prefix phase can be found efficiently by following the suffix links.

3.3 Finding Hypotheses

Now we know how to construct a suffix tree, the question arises how to extract hypotheses from this suffix tree. Due to space restrictions, we will describe the algorithms in an informal way, which will illustrate how the actual algorithms work[1].

Let us consider a mini-corpus consisting of the following sentences:

S_1=[she, is, walking, away],
S_2=[she, was, walking],
S_3=[she, runs, away]

A generalized suffix tree of these sentences is depicted in Figure 1.

[1] [11] gives more formal descriptions of the alignment learning algorithms and discusses them in more detail.

From Definition 1 it follows that words or word groups that are used in more than one sentence *and are followed by disjoint words* have a joint edge. This can also be found when examining the topology of the generalized suffix tree. The location 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 the pair of brackets to close at the end of the sentences for now. This algorithm can be summarized as follows:

1. Use the joint edges in the suffix tree to place all *opening* brackets;
2. Close every opening bracket at the end of the sentence;
3. Assign non-terminal labels to brackets, creating labeled hypotheses.

From now on, we will refer to this algorithm as ST:s (Suffix Tree:suffix). Using ST:s on the generalized suffix tree of S_1, S_2 and S_3 will result in the following bracketing:

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

One of the major limitations of ST:s is that only *suffix hypotheses* are presented. In other words, we only see where joint words are followed by distinct words, but not the other way around. To find distinct words followed with joint words (e.g. "away" in the first and third sentence in the example), we construct a generalized suffix tree for sentences *with reversed word order*. We refer to this kind of suffix tree as a *prefix tree*. A generalized prefix tree for the three example sentences is depicted in Figure 2.

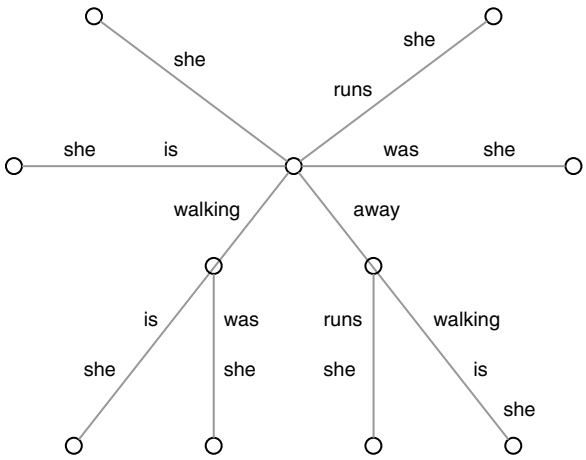

Fig. 2. Generalized prefix tree

By analyzing the prefix tree similarly to the analysis of the suffix tree, we are able to place closing brackets. Since the opening brackets cannot be read

from the prefix tree we assume prefix hypothesis to start at the beginning of the sentence, as we did with the closing brackets in the suffix tree analysis. We can generate both suffix and prefix hypothesis by an algorithm summarized as follows:

1. Use the joint edges in the suffix tree to place all *opening* brackets;
2. Close every opening bracket introduced in the previous step at the end of the sentence;
3. Use the joint edges in the prefix tree to place all *closing* brackets;
4. Open every closing bracket introduced in the previous step at the beginning of the sentence;
5. Label brackets belonging to each other with a non-terminal type label.

We will refer to this algorithm as ST:ps1 (Suffix Tree:prefix suffix version 1). Using ST:ps1 on the generalized suffix tree of S_1, S_2 and S_3 will result in the following bracketing:

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

Although ST:ps1 will capture more hypotheses than ST:s, there is still the limitation that hypotheses *either* start at the beginning of the sentence and stop somewhere in the middle of the sentence *or* start somewhere in the sentence and stop at the end of the sentence. To capture hypotheses that start somewhere in the sentence and stop somewhere later in the sentence as well, we introduce ST:ps2, which uses both prefix and suffix trees but additionally matches opening and closing brackets:

1. Use the joint edges and suffix links in the suffix tree to place all *opening* brackets;
2. Use the joint edges and suffix links in the prefix tree to place all *closing* brackets;
3. Match each opening bracket with each closing bracket appearing later in the sentence to form hypotheses;
4. Label brackets belonging to each other with non-terminals.

An illustration of how ST:ps2 matches opening brackets with closing brackets to induce hypotheses is depicted in Figure 3. The opening and closing brackets are inserted according to information from the trees. The table in the middle of the figure represents the storage that contains the joint identifier for each unique pair of brackets. The sentences in the right part of the figure contain proper hypotheses. The storage is used to replace the identifier of the brackets in one pair with the joint identifier. In this way, each opening bracket (introduced by using the suffix tree) is paired with each closing bracket (introduced by using the prefix tree) to form hypotheses[2].

[2] For the sake of simplicity, we omit placing an initial opening and a closing bracket at the beginning and the end of each sentence.

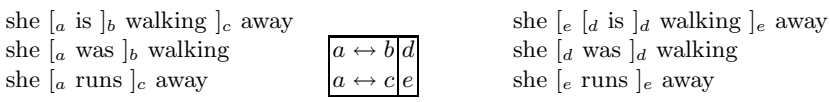

Fig. 3. Matching opening and closing brackets in hypotheses generation

4 Results

4.1 Evaluation Methodology

To evaluate the performance of the learning systems, we extract plain natural language sentences from a treebank, that is considered the *gold standard*. These sentences are used as input of the grammar induction system. We compare the output of the grammar induction system, the learned treebank, against the gold standard. This is done using the unlabeled version of the PARSEVAL metrics: recall (measuring completeness of the structure), precision (correctness), and f-score (geometric mean of recall and precision). Additionally, we examine the number of constituents present in the treebanks.

As we are interested in comparing the performance of alignment learning given different instantiations, we have to keep the performance of the selection learning phase constant. We do this by assuming the *perfect selection learning*. This means that from the hypotheses in its input, the selection learning process selects only those hypotheses that are also present in the original treebank. Since only correct hypotheses are selected from the learned treebank, the precision in these kind of experiments is 100% and the recall indicates the proportion of correct constituents that is introduced by the alignment learning phase. The selection learning can never improve on these values[3].

4.2 Performance on the ATIS Treebank

The English ATIS (Air Traffic Information System) treebank, taken from the Penn treebank [12], contains mostly questions and imperatives on air traffic. It contains 568 sentences with a mean length of 7.5 words per sentence and the lexicon size is 358 words. Examples of sentences from the ATIS are:

How much would the coach fare cost
I need to return on Tuesday
Does this flight serve dinner

The results of applying the different alignment learning phases to the ATIS treebank using ten fold cross-validation and selecting only the hypotheses that are also present in the original treebank can be found in Table 1. The results marked with *rnd* are used as a baseline. They are generated by randomly selecting a right-branching or left-branching tree structure on top of the sentence. *Def*

[3] Considering only perfect selection learning does not show the differences in bias of the different alignment learning algorithms. More research needs to go into the actual effect of this influence.

is the default alignment method based on the edit distance algorithm, with the standard longest common subsequence edit costs [1][4], while the other algorithms are based on suffix trees.

Table 1. Results for alignment learning on the ATIS treebank

		recall	f-score	learned	best	% left
	rnd	28.90 (0.85)	44.83 (0.70)	4,353 (0.0)	1,851 (25.6)	42.5
ED	def	48.08 (0.09)	64.94 (0.08)	12,692 (8.8)	4,457 (4.4)	35.1
ST	s	31.56 (0.00)	47.98 (0.00)	4,194 (0.0)	2,290 (0.0)	54.6
	ps1	34.85 (0.00)	51.68 (0.00)	8,527 (0.0)	2,750 (0.0)	32.3
	ps2	46.85 (0.00)	63.81 (0.00)	25,740 (0.0)	7,277 (0.0)	28.3

The recall and f-score metrics are self explanatory (standard deviation is given between brackets). The *learned*, *best* and *% left* give an overview of the number of hypotheses in the learned treebank at two moments in the learning process. The number of hypotheses present (including overlapping hypotheses) after the alignment learning phase are given in *learned*. The number of remaining hypotheses after the "perfect" selection learning phase are given in *best*. Comparing the two counts indicates how productive the alignment learning is and how many correct hypotheses are left after removing the incorrect ones. *% left* shows the percentage of hypotheses that remain after selection.

To be able to relate the counts of hypotheses, the original ATIS treebank contains 7,017 constituents. Also, note that the performance of the edit distance methods depends on the order of the sentences in the corpus whereas the performance of the suffix tree methods is invariant (hence their zero standard deviation).

When looking at the recall and f-score of the suffix tree methods, the ST:ps2 generates best results, as expected. Comparing them against the baseline, we see that all generate better results. The main reason for ST:s, which is closest to the baseline, to do poorly is that not enough hypotheses are generated (as indicated by the low learned measure). Although ST:ps2 performs reasonably well, the edit distance method performs slightly better. We do not have a clear explanation for this, especially when we consider the fact that ST:ps2 produces substantially more hypotheses, but we expect that adding more data will be in the benefit of ST:ps2.

Taking another look at the number of learned hypotheses, we observe a larger difference between method ST:s and ST:ps1 than difference between ST:ps1 and ST:ps2. This can be explained by considering the right-branching nature of the English language. Since prefix hypotheses implicitly have a left-branching nature, the effect of including prefix hypotheses (ST:ps1) will not result in a high increase in recall. In the same line of thought, we predict that method ST:s performs worse on corpora of left-branching languages such as Japanese.

[4] [1] uses a slightly different ATIS version of 577 sentences. Here we use the original Penn treebank 2 ATIS version of 568 sentences.

4.3 Performance on the WSJ Treebank

The Wall Street Journal (WSJ) treebank consists of newspaper articles. It contains more complex sentences than the questions and imperatives of the ATIS treebank and the lexicon is much larger. The treebank is divided into numbered sections. We shall use section 23, which has informally developed as a test section of the WSJ treebank[5]. It contains 1,904 sentences with an average sentence length of > 20 words. The lexicon of section 23 contains 8,135 distinct words and the section has 65,382 constituents. Some examples of sentences from section 23 of the WSJ treebank are:

> Big investment banks refused to step up to the plate to support the beleaguered floor traders by buying big blocks of stock.
> Once again the specialists were not able to handle the imbalances on the floor of the New York Stock Exchange.
> When the dollar is in a free-fall, even central banks can't stop it.

For the WSJ, we only compare the edit distance alignment method (default) and the best suffix tree alignment method (ST:ps2). The results of applying these algorithms to section 23 of the WSJ are shown in the upper part of Table 2.

Table 2. Results for alignment learning on section 23 and sections 2–21+23 of WSJ

Section			recall	f-score	learned	best	% left
23	ED	def	53.26 (0.07)	69.50 (0.12)	186,593 (94.3)	34,835 (19.7)	18.7
	ST	ps2	67.81 (0.00)	80.82 (0.00)	431,432 (0.0)	58,115 (0.0)	13.5
2–21+23	ST	ps2	65.98 (0.00)	79.47 (0.00)	10,759,549 (0.0)	1,353,022 (0.0)	12.6

It is interesting to see that, even though the corpus is considered more complex, the results seem to be better compared to the ATIS treebank. This is perhaps not entirely true, since many more hypotheses are found and less than 20% of the learned hypotheses remain after perfect selection learning. (This is still over 28% with the ATIS treebank.) However, the upper bound is quite high, and ST:ps2 performs significantly better than the edit distance method.

One of the main reasons to develop the suffix tree alignment method was to reduce the scalability problem. To show that this has indeed been removed, we have applied the system to a large part of the WSJ corpus. For this purpose, we use sections 02–21 (standard training set sections) and 23 together. With a corpus of this size (38,009 long sentences), edit distance alignment learning is computationally not feasible anymore. Therefore, we only present the results for the suffix tree alignment learning (lower part of Table 2). The treebank fragment has 1,355,210 constituents and the lexicon contains 40,982 distinct words.

As can be observed, the recall has dropped slightly. Compared to the performance on the previous treebanks, these results are in the line of expectation.

[5] See e.g. [13, 14, 1].

The proportion of selected hypotheses (*best*) to the learned hypotheses (*learned*) seems to be correlated to the sentence complexity, because this proportion is considerably smaller for WSJ 2–21/23 than it is for ATIS.

4.4 Performance on Execution Time

Apart from the numerical results on actual natural language corpora, we are also interested in the execution times of the systems. Table 3 gives an overview of the time systems takes to learn on several corpora[6]. This clearly shows the benefits of suffix tree alignment over the edit distance variant.

Table 3. Timing results (in seconds) on several corpora

	edit distance	suffix tree
ATIS	4.05 (0.04)	0.37 (0.03)
WSJ23	283.00 (0.73)	6.08 (0.39)
WSJ2–21+23	n.a. n.a.	124.86 (0.11)

On the ATIS corpus, suffix tree alignment is approximately a factor 11 faster than edit distance alignment. On WSJ section 23, this is already a factor 46 and on WSJ section 02–21+23, suffix tree alignment needed about 2 minutes whereas the edit distance alignment learning was aborted after some 14 hours of computation (resulting in a factor >400).

5 Conclusion

In this article, we introduced a new alignment learning instantiation that is to be used in the ABL grammatical inference framework. Instead of using the edit distance algorithm (as all other alignment learning methods), this method is based on suffix trees. When applying the systems to real natural language corpora, we see that the results of the suffix tree alignment methods are comparable or significantly better. However, the main advantage of the new systems is that it solves the scalability problem inherent in the edit distance systems. Where the complexity of the edit distance based alignment learning is squared in the size of the corpus, the complexity of the suffix tree based system is linear. This greatly improves the usability of this approach.

One main aspect of comparing different alignment learning instantiations is the influence of the bias of the algorithms on the selection learning phase. The difficulty is that the specific selection learning instances themselves probably have a preferred bias. Future work should look into the complex interaction between the separate phases.

[6] These results were generated on a AMD Athlon XP 2200, 1800MHz with 1GB internal memory.

References

1. van Zaanen, M.: Bootstrapping Structure into Language: Alignment-Based Learning. PhD thesis, University of Leeds, Leeds, UK (2002)
2. van Zaanen, M.: Theoretical and practical experiences with Alignment-Based Learning. In: Proceedings of the Australasian Language Technology Workshop; Melbourne, Australia. (2003) 25–32
3. Harris, Z.S.: Structural Linguistics. 7th (1966) edn. University of Chicago Press, Chicago:IL, USA and London, UK (1951) Formerly Entitled: Methods in Structural Linguistics.
4. van Zaanen, M.: Implementing Alignment-Based Learning. In Adriaans, P., Fernau, H., van Zaanen, M., eds.: Grammatical Inference: Algorithms and Applications (ICGI); Amsterdam, the Netherlands. Volume 2482 of Lecture Notes in AI., Berlin Heidelberg, Germany, Springer-Verlag (2002) 312–314
5. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. Journal of the Association for Computing Machinery **21** (1974) 168–173
6. van Zaanen, M., Adriaans, P.: Alignment-Based Learning versus EMILE: A comparison. In: Proceedings of the Belgian-Dutch Conference on Artificial Intelligence (BNAIC); Amsterdam, the Netherlands. (2001) 315–322
7. Adriaans, P.: Language Learning from a Categorial Perspective. PhD thesis, University of Amsterdam, Amsterdam, the Netherlands (1992)
8. Weiner, P.: Linear pattern matching algorithms. In: Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory, Los Alamitos, Calif., USA, IEEE Computer Society Press (1973) 1–11
9. McCreight, E.M.: A space-economical suffix tree construction algorithm. Journal of the Association for Computing Machinery **23** (1976) 262–272
10. Ukkonen, E.: On-line construction of suffix trees. Algorithmica **14** (1995) 249–260
11. Geertzen, J.: String alignment in grammatical inference: what suffix trees can do. Technical Report ILK-0311, ILK, Tilburg University, Tilburg, The Netherlands (2003)
12. Marcus, M.P., Santorini, B., Marcinkiewicz, M.A.: Building a large annotated corpus of English: the Penn treebank. Computational Linguistics **19** (1993) 313–330
13. Charniak, E.: Statistical parsing with a context-free grammar and word statistics. In: Proceedings of the Fourteenth National Conference on Artificial Intelligence, American Association for Artificial Intelligence (AAAI) (1997) 598–603
14. Collins, M.: Three generative, lexicalised models for statistical parsing. In: Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL) and the 8th Meeting of the European Chapter of the Association for Computational Linguistics (EACL); Madrid, Spain, Association for Computational Linguistics (ACL) (1997) 16–23