Theoretical and Practical Experiences with Alignment-Based Learning

Menno van Zaanen
ILK, Tilburg University
P.O. Box 90153, 5000 LE Tilburg, The Netherlands
mvzaanen@uvt.nl
Tel: +31 13 466 8260, Fax: +31 13 466 3110

Abstract

This article discusses empirical and theoretical properties of the Alignment-Based Learning (ABL) grammar induction framework (van Zaanen, 2002a). ABL is an unsupervised grammar induction system that generates structure, applying Harris's (1951) idea of substitutability to a set of plain sentences. Sentences are aligned against each other in pairs, resulting in a partition of equal and unequal parts of both sentences. Since substituting unequal parts of sentences results in another valid sentence, the unequal parts of sentences are considered possible constituents, called hypotheses. Next, the possibly conflicting hypotheses are disambiguated using statistics.

This article concentrates on the current implementation of ABL (van Zaanen, 2002b). Following a detailed description of the distinct phases, theoretical time bounds are computed. The actual time used when applying the system is much lower than the theoretical upper bounds, alleviating the claim that ABL is not practically usable.

1 Introduction

Recently, there has been a growing interest in the (unsupervised) learning of context-free grammars. This interest has come not only from the field of formal learnability, as can be seen, for example, in the recent proceedings of the ICGI (Adriaans et al., 2002), but also from a linguistic, or practical point of view (Clark, 2001; Klein and Manning, 2001). Additionally, these two fields seem to merge (de la Higuera et al., 2003).

Alignment-Based Learning (ABL) is a state-of-the-art unsupervised grammar induction system, that aims to be useful to both formal (Starkie, 2003) and practical fields (van Zaanen, 2002a). Using ideas from linguistics and statistics, it takes unstructured sentences as input and generates (context-free) parse trees as output.

In the next section, all programs contained in the ABL software package will be discussed in detail. This includes the actual learning programs, as well as a selection of support programs that allow for a complete project (conversion, learning, and evaluation) to be set up and executed. In previous work, the ABL system was divided into two phases. Here, three distinct phases (and hence programs) are recognized, effectively dividing the first phase into two: *alignment learning*, *clustering*, and *selection learning*.

Next, theoretical limits of the system will be discussed and typical execution times will be given. In the section on future work, alternative phases will be discussed briefly, together with their possible improvements in the theoretical limits of the system.

2 Alignment-Based Learning package

The programs in the ABL software package can be divided into two groups, the *main programs* that perform the actual learning and the *support*

programs that facilitate the handling of the data flow, the conversion and the evaluation of the system.

The main programs of the current ABL implementation, consisting of the three phases (discussed in detail below), are written in C++. This language was chosen, because it is fast, flexible, allowing easy incorporation of different instantiations to be selected (even at run-time), has several useful complex data types built in (with the Standard Template Library), and is quite portable.

The support programs can be divided into two groups: *Conversion tools* allow easy handling of data in several corpus and treebank formats. Also, it is possible to remove structure (useful for testing) and set certain restrictions on the data. Most of these programs are written in Perl, because of the easy data handling. *Meta programs* help manage the collection of programs. `abl_convert` manages all the conversion programs, giving the user one simple interface for all conversions. Similarly, `abl_start` keeps track of all settings and phases that typically occur in a project. The user uses one program to set

- the input file that should be used (and in which format it is),
- the ABL phases that should be performed and whether intermediate output should be saved,
- for each phase, several settings, such as different instances of the phases or random seeds,
- the output directory where all data is written to.

Once the settings are chosen, the program can generate a script that contains all necessary program calls to compute all selected output or it can start the scripts directly. Of course, these programs can also be called by the user manually; the generated script merely makes it easier to handle the different options and data files.

2.1 Learning

Learning in ABL consists of three phases. First, plain sentences are used in the *alignment learning* phase (`abl_align`) to generate possible structure. This is then forwarded to the *clustering* phase (`abl_cluster`), where type labels are grouped. The final phase, *selection learning* (`abl_select`) selects the best structure. This can then be handed to the evaluation phase.

`abl_build` is a meta program that can handle the different programs that perform the actual learning. It can be told to do one, two or all three phases of ABL and also whether the output of a phase should be stored for further evaluation or whether it can be discarded immediately afterward.

Furthermore, one or more different instantiations of each phase can be selected as well as a number of random seeds to be used. This allows the generation of several output files (with automatically composed file names) in only one program call.

2.1.1 Alignment learning phase

The alignment learning phase finds possible constituents, called hypotheses, by aligning pairs of sentences to each other. Following an adjusted and reversed version of Harris's (1951) implication:

> If parts of sentences can be substituted by each other then they are constituents of the same type.

Groups of words that are *unequal* in a pair of sentences are considered to be yields of hypotheses.

The sentences in figure 1 give two examples of alignments. In the first case, *sees* is aligned within the two sentences (the alignment is indicated by underlining). No other words can be aligned at the same time. Based on this alignment, ABL concludes that *The man* and *John* are interchangeable and thus are considered hypotheses, as are *Mary* and *a woman*. This is indicated by the pairs of brackets. Furthermore, these two sentences only give evidence that *The man* is interchangeable with *John* and not with for example *Mary*, even though in practice they

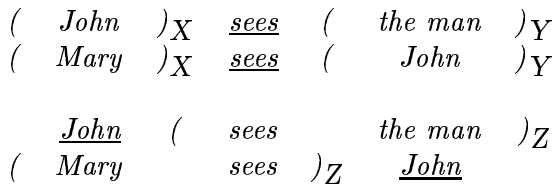

Figure 1: Different possible alignments

are. To keep track of this information, the non-terminal type labels X and Y are added to the hypotheses.[1]

Of course, learning only from two sentences does not give much information (even though more than one pair of hypotheses can be introduced namely hypotheses with type X and Y, as in the alignment of the first pair of sentences in the example). A proper induction system should be able to handle more sentences. This is done by aligning all sentences in a corpus with all other sentences, pair by pair.

In the current implementation, three alignment learning methods are implemented:

default This method analyzes the sentences and finds an alignment that has the maximum number of aligned words.

biased Sometimes, the **default** method aligns words that are very far apart. Figure 1 shows that certain pairs of sentences have multiple possible alignments. Since both alignments are valid, the **default** method selects one at random. However, it is clear that the first alignment is preferred, since it matches our ideas about natural language syntax more closely. The second one aligns words that are relatively far apart in the sentences (at the beginning and the end). The **biased** method has a strong preference for alignments that are roughly in the same part of the sentence, disallowing the second alignment.

all Further research should point out if the **biased** method really solves the problem of wrong alignments. The **all** method takes another approach. Instead of trying to select the "correct" alignment, it simply uses all possible alignments. This keeps the alignment learning algorithm simple (no decisions have to be made regarding possible alignments), but this increases the burden on the selection learning phase.

Figure 2 illustrates another potential problem of the approach at hand. The first sentence is aligned to the second, finding the unequal parts *Book Delta 128* and *Give me all flights*. These are considered hypotheses (which is indicated by the pair of brackets).

When the second sentence is aligned to the third, it receives another hypothesis, which *overlaps* with the older hypothesis. However, overlapping hypotheses cannot exist, if the structure described by the hypotheses is seen as a parse of a context-free grammar. The problem of overlapping hypotheses is solved by the selection learning phase (as described in section 2.1.3).

2.1.2 Clustering phase

The goal of the clustering phase is to group similar non-terminal types of hypotheses. This effectively merges non-terminal types that are considered to be equal.

The alignment learning phase generates a new non-terminal type (in the example, non-terminals X and Y are introduced) whenever a new pair of hypotheses is found. This results in an explosion of non-terminal types, which is of course unwanted.

Based on Harris's notion of substitutability, a clustering algorithm is implemented that groups two non-terminals if they occur in the same context. In practice, if different non-terminals are assigned to one hypothesis by the alignment learning phase, they are clustered together and

[1] When aligning with multiple sentences, care has to be taken when new hypotheses should be introduced in the same location as already existing hypotheses. See (van Zaanen, 2002a) for more information.

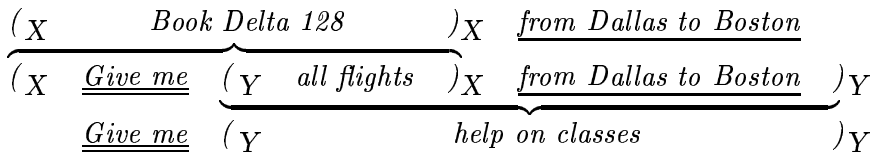

Figure 2: Overlapping hypotheses

all the occurrences of the non-terminals are replaced throughout the generated structure.

Even though merging non-terminals this way is in line with Harris's idea, it sometimes introduces incorrect type labels. Consider for example the sentences in figure 3. In this case, the hypotheses containing *meat* and *quickly* receive the same non-terminal type, which means that they are always interchangeable, which is of course not the case.

A more complex clustering method should take into account that hypotheses are interchanged in certain contexts, but not all. In the case of *meat* and *quickly*, *meat* can occur in NP contexts, such as a subject position, but *quickly* cannot.

Currently, the correctness of the non-terminal labels is not evaluated. However, the results of some selection learning methods are influenced by the assignment and clustering of the non-terminal labels.

2.1.3 Selection learning phase

The alignment learning phase generates many hypotheses by aligning sentences against each other. Some of these hypotheses overlap (as shown in figure 2). If we assume that the underlying grammar (i.e. the grammar to be learned) is context-free and the pairs of brackets in the sentence describe a derivation of the sentence based on the underlying grammar, no overlapping hypotheses should be present.

The task of the selection learning phase is to select the best hypotheses (or remove the worst ones), so a proper tree structure in the form of pairs of brackets remains.

Similarly to the alignment learning phase, the selection learning phase can have different instantiations. The current implementation contains the following three methods:

incr The easiest way to make sure that no overlapping hypotheses occur is to never even allow them. The **incr** method works chronologically. If at some point a hypothesis is introduced that overlaps with an already existing hypothesis, the new hypothesis is discarded. This way, no overlapping hypotheses are introduced. However, once an incorrect hypothesis is learned, it will never be corrected.

leaf The **leaf** method is a probabilistic method that tries to solve the problem of correcting wrong hypotheses by preferring more probable hypotheses. The probability of a hypothesis h from the set of hypotheses U, containing all hypotheses in the corpus, is computed as follows:

$$P^U_{\mathbf{leaf}}(h) = \frac{|h' \in U : Y(h') = Y(h)|}{|U|}$$

where Y denotes the yield of a hypothesis, which is the list of words that are marked by the brackets. The probability of hypothesis h is the proportion of hypotheses that have the same yield as h.

branch The **branch** method is similar to the **leaf** method, the only difference being that the computation of the probability of a hypothesis is relative to the non-terminal (returned by R). The probability of a hypothesis h is the proportion of hypotheses that have the same yield *and* non-terminal label as h:

$$P^U_{\mathbf{branch}}(h) = \frac{|h' \in U : Y(h') = Y(h) \wedge R(h') = R(h)|}{|h'' \in U : R(h'') = R(h)|}$$

The probabilistic methods compute the probability of each hypothesis. The most probable

John eats (*meat*)$_X$
John eats (*quickly*)$_X$

Figure 3: Assignment of incorrect non-terminal types

combination of hypotheses is then selected by computing the combined probability (using the geometric mean) of each possible combination of non-overlapping hypotheses (see (van Zaanen, 2002a) for a more elaborate discussion).

2.2 Evaluation

When all learning phases are done, the user might want to evaluate the experiment. For this, the package contains `abl_eval`. This program receives files to evaluate (i.e. the generated data and the originally structured version of this data) and the evaluation metrics that need to be computed.

Currently, the evaluation program calls the EVALB program (Collins, 1997) to compute metrics (such as precision and recall), but additional metrics such as the number of learned hypotheses and the number of unique non-terminal types can also be calculated.

Sometimes, it may be useful to know what the results of the alignment learning phase (or the alignment learning and the clustering phase together) by itself are. The problem is that these phases generate ambiguous structures. To solve this, the `abl_max_score` program is implemented, which extracts the best structure still possible from the generated files. It does this by selecting only those pairs of brackets that occur in both the generated data and the original treebank, simulating a perfect selection learning phase. The results computed in this way indicate the upper bound that is reached if the selection learning method is perfect. It gives an indication of how well the alignment-learning and clustering methods work. Furthermore, once the upper bound is known, the effectiveness of the selection learning methods can be evaluated.

3 Results

Different combinations of instantiations of the phases of ABL have been tested on different corpora and compared against other systems, e.g. (van Zaanen and Adriaans, 2001; Clark, 2001). Here, the theoretical limits of different instantiations of the phases will be investigated first, followed by empirical results of the system.

3.1 Theoretical limits

3.1.1 Alignment learning limits

The current implementations of the alignment learning instantiations (except **all**) depend on the well-known edit-distance algorithm (Wagner and Fischer, 1974)[2], which computes the Levenshtein distance between two strings (Levenshtein, 1965). From this information it is possible to find which words are equal in both sentences, and hence find all possible alignments.

The edit-distance algorithm builds an n by m matrix with n and m the lengths of the sentences. Using a dynamic programming technique, this matrix is filled in $O(nm)$.[3] We then need to do a trace-back to find the actual alignment. This can be done in $O(n+m)$. If there are multiple alignments, say p, these can found in $O(p(n+m))$.

The **default** and **biased** methods read in the corpus and simply use the edit distance algorithm to find an alignment. Based on this alignment, hypotheses are added to the sentences. The time complexity of alignment learning two sentences is $O(s^2+s+h)$ with s the length of the longest sentence and h the number of introduced hypotheses[4] (s^2 for the filling of the matrix, s for the traceback and h for the insertion of the hypotheses). This needs to be done for each pair of sentences in the corpus ($c(c+1)/2$ times), so

[2]The article also describes the time complexity of the algorithm.

[3]Note that there exists another algorithm, often named *Four Russians* that finds the edit distance in $O(n^2/\log n)$ for two strings of size n, assuming a fixed alphabet. This algorithm is fast when the strings are very long (Gusfield, 1997).

[4]h depends on the length of the sentences and on the size of the lexicon.

the alignment phase for an entire corpus takes $O(c + c^2(s^2 + s + h))$, c for reading in the corpus and c^2 times alignment of pairs of sentences. Simplified: $O(c^2(s^2 + h))$.

The **all** method needs to handle more alignments. Building the table still takes $O(s^2)$, but retrieving p alignments[5] takes $O(ps)$. The rest of the algorithm is the same as above, resulting in $O(c^2(s^2 + ps + h))$. However, in the current implementation another algorithm is used (van Zaanen, 2002a, p. 47). This algorithm finds all sub-sentences that occur in both sentences (taking $O(s^2)$). Alignments are found when sub-sentences equal in both sentences are matched. By considering all possible combinations of the sub-sentences that have non-crossing links, all possible alignments are found. This is done in $O(c^2(s^2 + w^2 + h))$, where w is the number of sub-sentences that are found in both sentences.

The main problem with the approach seems to be that the time complexity depends on the size of the corpus squared. Since we would like to be able to handle very large corpora, even with the increasing computational power of computers, this might pose a problem.

3.1.2 Clustering limits

The clustering phase reads in a corpus, which takes $O(c)$. It then scans the corpus for hypotheses that have more than one non-terminal type assigned to it by the alignment learning phase ($O(ch)$). These types are clustered, which again takes $O(ch)$, and the resulting corpus is written as output ($O(c)$). In total, this simple phase can be done in $O(ch)$.

3.1.3 Selection learning limits

In the selection learning phase, the corpus is read ($O(c)$) and then analyzed for overlapping hypotheses. To find whether a hypothesis overlaps with another, it is compared to the other hypotheses in the same sentence taking $O(h^2)$, so $O(ch^2)$ for the entire corpus.

The **incr** selection learning method simply rejects hypotheses that overlap with already existing hypotheses, so assuming the hypotheses are stored in chronological order, hypotheses that overlap can be automatically removed. Once the entire corpus is analyzed, the phase is done and the corpus can be written as output. Since no additional computation needs to take place, this method can be done in $O(ch^2)$.

Both **leaf** and **branch** methods need more analysis before hypotheses can be removed. In addition to reading the corpus and finding the overlapping hypotheses, the probabilities of all possible combinations of the hypotheses that do not overlap need to be computed. A naive approach that really analyzes all combinations in the corpus would take $O(c2^h)$. However, using a Viterbi algorithm optimization (Viterbi, 1967), we can reduce this to $O(ch^2)$. The time complexity of the probabilistic instances is then also $O(ch^2)$.

3.2 Experimental results

The different instantiations of the phases are tested on two treebanks. The ATIS treebank, taken from the Penn Treebank 2 (Marcus et al., 1993), is an English treebank containing 568 sentences with a mean 7.5 words per sentence (maximum length is 34 words). The OVIS (Openbaar Vervoer Informatie Systeem) treebank (Bonnema et al., 1997) is a Dutch treebank, containing 6797 sentences of roughly 3.5 words (maximum length is 23 words).

It is important to know how the corpus size influences the execution times, since we would like to apply the system on much larger corpora. The sentence length is not very interesting in the context of language, since the sentence length only changes marginally. It is extremely difficult, if not impossible, to change the number of hypotheses, but it is expected (considering the theoretical limits) that this will not be the most important term with respect to timing. Similarly, the number of unique words influences the system, but the relation to the timing results is unclear and changing it is difficult. Therefore, the focus is on the number of sentences only.

From each treebank, 8 subsets of increasing size are extracted (the largest being the entire treebank). We have taken into account that the number of sentences *and* the number of words

[5] p also depends on the size of the lexicon and on the length of the sentences.

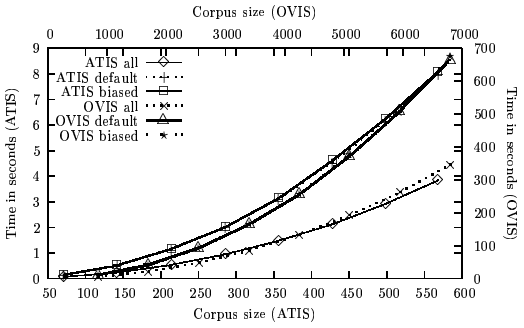

Figure 4: Alignment learning

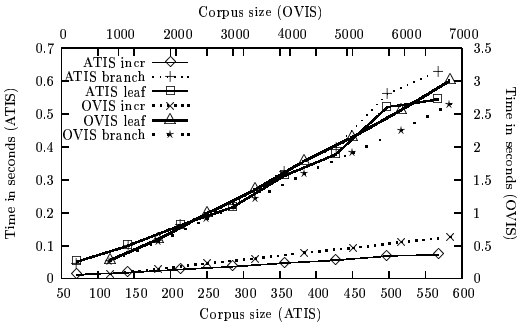

Figure 5: Selection learning

is increasing linearly with larger subsets. Each smaller subset is contained in the larger subset. The number of unique words is slowly increasing, which is unfortunate, but this could be expected since the texts become larger. The corpora are not large enough to diminish this effect.

All instances of all phases are run ten times on the subsets and timing information is collected. Figures 4 and 5 show the increase in time when the corpus size increases. The alignment learning phase takes longest of all phases. The increase is slightly worse than linear, but fortunately not as bad as one might have expected. In all cases, the **all** method is fastest, which may seem surprising. This is due to the fact that there are not many words that occur in both sentences when aligning a pair of sentences, so less work has to be done compared to the edit distance algorithm.

The clustering phase takes at most 4 seconds on the largest OVIS set, and grows slowly and nearly linearly in the corpus size. With respect to the time of the other phases, this phase hardly needs to be taken into account.

The selection learning phase grows nearly linearly as well, with the **incr** method clearly being fastest. This is because no expensive computation and search has to be performed on the hypotheses.

4 Future work

In the near future, new instantiations of all three phases will be implemented. The main focus will be on the computationally expensive alignment learning phase. One idea is to use a suffixtree (Ukkonen, 1995) to store the corpus, which can be built in $O(cs)$. Then, the compact representation of the corpus can be analyzed. Using suffixtrees will be a lot faster, but not all alignments will be found, which again may not be a problem since with this approach larger corpora can be used.

Alternative clustering methods need to be investigated, since the current approach sometimes assigns incorrect non-terminal types to hypotheses, as depicted in figure 3.

New selection learning methods will implement alternative ways of computing the probabilities of hypotheses, which will increase the final results in the sense of precision and recall. Furthermore, more research needs to go into the interaction between the clustering phase and the selection learning phase.

5 Conclusion

The current implementation of the ABL grammar induction system, which will become publicly available soon, has three main programs with an easy interface which makes it very user-friendly. Because it is programmed in C++, it is efficient and easily extensible.

Apart from a description of the implementation of ABL, this article also investigated the theoretical and empirical computational complexity. It was found that the alignment learning phase requires most computational power with $O(c^2(s^2 + h))$, c corpus size, s sentence length and h number of hypotheses per sentence or $O(c^2(s^2 + w^2 + h))$ with w the number of subsentences that can be found in both sentences for the **all** method. The cluster phase only requires $O(ch)$, while the selection learning phase

takes $O(ch^2)$.

The empirical results showed that the alignment learning phase takes most time and increases fastest, making the application of ABL on very large corpora currently impractical, however, the system was not as slow as one would expect from the theoretical limits. With this in mind, developing new instances should be directed to the "slow" alignment learning phase.

Acknowledgments

I would like to thank Tanja Gaustad and two anonymous reviewers for their useful comments.

References

Association for Computational Linguistics (ACL). 1997. *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*, July.

Pieter Adriaans, Henning Fernau, and Menno van Zaanen, editors. 2002. *Grammatical Inference: Algorithms and Applications (ICGI); Amsterdam, the Netherlands*, volume 2482 of *Lecture Notes in AI*, Berlin Heidelberg, Germany, September 23–25. Springer-Verlag.

R. Bonnema, R. Bod, and R. Scha. 1997. A DOP model for semantic interpretation. In (ACL, 1997), pages 159–167.

Alexander Clark. 2001. Unsupervised induction of stochastic context-free grammars using distributional clustering. In (CoNLL, 2001), pages 105–112.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In (ACL, 1997), pages 16–23.

Computational Language Learning (CoNLL). 2001. *Proceedings of the Workshop on Computational Natural Language Learning held at the 39th Annual Meeting of the Association for Computational Linguistics (ACL) and the 10th Meeting of the European Chapter of the Association for Computational Linguistics (EACL); Toulouse, France*, July.

Colin de la Higuera, Pieter Adriaans, Menno van Zaanen, and Jose Oncina, editors. 2003. *Proceedings of the Workshop and Tutorial on Learning Context-Free Grammars held at the 14th European Conference on Machine Learning (ECML) and the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD); Dubrovnik, Croatia.*

Dan Gusfield. 1997. *Algorithms on strings, trees, and sequences: computer science and computational biology.* Cambridge University Press, Cambridge, UK. Reprinted 1999 (with corrections).

Zellig S. Harris. 1951. *Structural Linguistics.* University of Chicago Press, Chicago:IL, USA and London, UK, 7th (1966) edition. Formerly Entitled: Methods in Structural Linguistics.

Dan Klein and Christopher D. Manning. 2001. Distributional phrase structure induction. In (CoNLL, 2001), pages 113–120.

V. I. Levenshtein. 1965. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSR*, 163(4):845–848. Original in Russian.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn treebank. *Computational Linguistics*, 19(2):313–330.

Bradford Starkie. 2003. Left aligned grammars—identifying a class of context-free grammar in the limit from positive data. In (de la Higuera et al., 2003), pages 89–101.

E. Ukkonen. 1995. On-line construction of suffix trees. *Algorithmica*, 14:249–260.

Menno van Zaanen and Pieter Adriaans. 2001. Alignment-Based Learning versus EMILE: A comparison. In *Proceedings of the Belgian-Dutch Conference on Artificial Intelligence (BNAIC); Amsterdam, the Netherlands*, pages 315–322, October.

Menno van Zaanen. 2002a. *Bootstrapping Structure into Language: Alignment-Based Learning.* Ph.D. thesis, University of Leeds, Leeds, UK, January.

Menno van Zaanen. 2002b. Implementing Alignment-Based Learning. In (Adriaans et al., 2002), pages 312–314.

A. Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, 13:260–269.

Robert A. Wagner and Michael J. Fischer. 1974. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21(1):168–173.