Error Mining for Wide-Coverage Grammar Engineering

Gertjan van Noord
Alfa-informatica University of Groningen
POBox 716
9700 AS Groningen
The Netherlands
vannoord@let.rug.nl

Abstract

Parsing systems which rely on hand-coded linguistic descriptions can only perform adequately in as far as these descriptions are correct and complete.

The paper describes an *error mining* technique to discover problems in hand-coded linguistic descriptions for parsing such as grammars and lexicons. By analysing parse results for very large unannotated corpora, the technique discovers missing, incorrect or incomplete linguistic descriptions.

The technique uses the frequency of n-grams of words for arbitrary values of n. It is shown how a new combination of suffix arrays and perfect hash finite automata allows an efficient implementation.

1 Introduction

As we all know, hand-crafted linguistic descriptions such as wide-coverage grammars and large scale dictionaries contain mistakes, and are incomplete. In the context of parsing, people often construct sets of example sentences that the system should be able to parse correctly. If a sentence cannot be parsed, it is a clear sign that something is wrong. This technique only works in as far as the problems that might occur have been anticipated. More recently, tree-banks have become available, and we can apply the parser to the sentences of the tree-bank and compare the resulting parse trees with the gold standard. Such techniques are limited, however, because tree-banks are relatively small. This is a serious problem, because the distribution of words is Zipfian (there are very many words that occur very infrequently), and the same appears to hold for syntactic constructions.

In this paper, an *error mining* technique is described which is very effective at automatically discovering systematic mistakes in a parser by using very large (but unannotated) corpora. The idea is very simple. We run the parser on a large set of sentences, and then analyze those sentences the parser cannot parse successfully. Depending on the nature of the parser, we define the notion 'successful parse' in different ways. In the experiments described here, we use the Alpino wide-coverage parser for Dutch (Bouma et al., 2001; van der Beek et al., 2002b). This parser is based on a large constructionalist HPSG for Dutch as well as a very large electronic dictionary (partly derived from CELEX, Parole, and CGN). The parser is robust in the sense that it essentially always produces a parse. If a full parse is not possible for a given sentence, then the parser returns a (minimal) number of parsed non-overlapping sentence parts. In the context of the present paper, a parse is called successful only if the parser finds an analysis spanning the full sentence.

The basic idea is to compare the frequency of words and word sequences in sentences that cannot be parsed successfully with the frequency of the same words and word sequences in unproblematic sentences. As we illustrate in section 3, this technique obtains very good results if it is applied to large sets of sentences.

To compute the frequency of word sequences of arbitrary length for very large corpora, we use a new combination of suffix arrays and perfect hash finite automata. This implementation is described in section 4.

The error mining technique is able to discover systematic problems which lead to parsing failure. This includes missing, incomplete and incorrect lexical entries and grammar rules. Problems which cause the parser to assign complete but incorrect parses cannot be discovered. Therefore, tree-banks and hand-crafted sets of example sentences remain important to discover problems of the latter type.

2 A parsability metric for word sequences

The *error mining* technique assumes we have available a large corpus of sentences. Each sentence is a sequence of words (of course, words might include tokens such as punctuation marks, etc.). We run the parser on all sentences, and we note for which sentences the parser is successful. We define the parsability of a word $R(w)$ as the ratio of the number of times the word occurs in a sentence with a

successful parse ($C(w|\text{OK})$) and the total number of sentences that this word occurs in ($C(w)$):

$$R(w) = \frac{C(w|\text{OK})}{C(w)}$$

Thus, if a word only occurs in sentences that cannot be parsed successfully, the parsability of that word is 0. On the other hand, if a word only occurs in sentences with a successful parse, its parsability is 1. If we have no reason to believe that a word is particularly easy or difficult, then we expect its parsability to be equal to the coverage of the parser (the proportion of sentences with a successful parse). If its parsability is (much) lower, then this indicates that something is wrong. For the experiments described below, the coverage of the parser lies between 91% and 95%. Yet, for *many* words we found parsability values that were much lower than that, including quite a number of words with parsability 0. Below we show some typical examples, and discuss the types of problem that are discovered in this way.

If a word has a parsability of 0, but its frequency is very low (say 1 or 2) then this might easily be due to chance. We therefore use a frequency cut-off (e.g. 5), and we ignore words which occur less often in sentences without a successful parse.

In many cases, the parsability of a word depends on its context. For instance, the Dutch word *via* is a preposition. Its parsability in a certain experiment was more than 90%. Yet, the parser was unable to parse sentences with the phrase *via via* which is an adverbial expression which means *via some complicated route*. For this reason, we generalize the parsability of a word to word sequences in a straightforward way. We write $C(w_i \dots w_j)$ for the number of sentences in which the sequence $w_i \dots w_j$ occurs. Furthermore, $C(w_i \dots w_j|\text{OK})$, is the number of sentences with a successful parse which contain the sequence $w_i \dots w_j$. The parsability of a sequence is defined as:

$$R(w_i \dots w_j) = \frac{C(w_i \dots w_j|\text{OK})}{C(w_i \dots w_j)}$$

If a word sequence $w_i \dots w_j$ has a low parsability, then this might be because it is part of a difficult phrase. It might also be that part of the sequence is the culprit. In order that we focus on the relevant sequence, we consider a longer sequence $w_h \dots w_i \dots w_j \dots w_k$ only if its parsability is lower than the parsability of each of its substrings:

$$R(w_h \dots w_i \dots w_j \dots w_k) < R(w_i \dots w_j)$$

This is computed efficiently by considering the parsability of sequences in order of length (shorter sequences before longer ones).

We construct a parsability table, which is a list of n-grams sorted with respect to parsability. An n-gram is included in the parsability table, provided:

- its frequency in problematic parses is larger than the frequency cut-off
- its parsability is lower than the parsability of all of its sub-strings

The claim in this paper is that a parsability table provides a wealth of information about systematic problems in the grammar and lexicon, which is otherwise hard to obtain.

3 Experiments and results

3.1 First experiment

Data. For our experiments, we used the Twente Nieuws Corpus, version pre-release 0.1.[1] This corpus contains among others a large collection of news articles from various Dutch newspapers in the period 1994-2001. In addition, we used all news articles from the Volkskrant 1997 (available on CD-ROM). In order that this material can be parsed relatively quickly, we discarded all sentences of more than 20 words. Furthermore, a time-out per sentence of twenty CPU-seconds was enforced. The Alpino parser normally exploits a part-of-speech tag filter for efficient parsing (Prins and van Noord, 2003) which was switched off, to ensure that the results were not influenced by mistakes due to this filter. In table 1 we list some basic quantitative facts about this material.

We exploited a cluster of Linux PCs for parsing. If only a single PC had been available, it would have taken in the order of 100 CPU days, to construct the material described in table 1.

These experiments were performed in the autumn of 2002, with the Alpino parser available then. Below, we report on more recent experiments with the latest version of the Alpino parser, which has been improved quite a lot on the basis of the results of the experiments described here.

Results. For the data described above, we computed the parsability table, using a frequency cut-off of 5. In figure 1 the frequencies of parsability scores in the parsability table are presented. From the figure, it is immediately clear that the relatively high number of word sequences with a parsability of (almost) zero cannot be due to chance. Indeed, the

[1] `http://wwwhome.cs.utwente.nl/~druid/TwNC/TwNC-main.html`

newspaper	sents	coverage %
NRC 1994	582K	91.2
NRC 1995	588K	91.5
Volkskrant 1997	596K	91.6
AD 2000	631K	91.5
PAROOL 2001	529K	91.3
total	2,927K	91.4

Table 1: Overview of corpus material; first experiment (Autumn 2002).

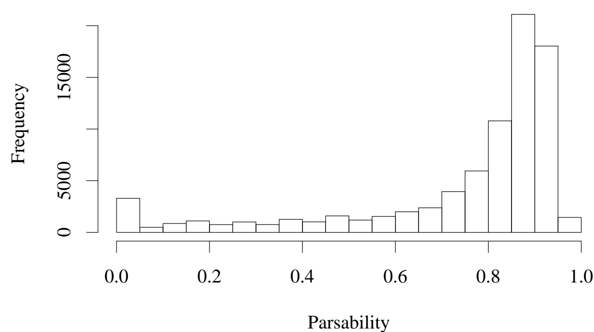

Figure 1: Histogram of the frequencies of parsability scores occurring in parsability table. Frequency cut-off=5; first experiment (Autumn 2002).

parsability table starts with word sequences which constitute systematic problems for the parser. In quite a lot of cases, these word sequences originate from particular types of newspaper text with idiosyncratic syntax, such as announcements of new books, movies, events, television programs etc.; as well as checkers, bridge and chess diagrams. Another category consists of (parts of) English, French and German phrases.

We also find frequent spelling mistakes such as *de de* where only a single *de* (the definite article) is expected, and *heben* for *hebben* (to have), *indentiek* for *identiek* (identical), *koninging* for *koningin* (queen), etc. Other examples include *wordt ik (becomes I), vindt ik (finds I), vind hij (find he)* etc.

We now describe a number of categories of examples which have been used to improve the parser.

Tokenization. A number of n-grams with low parsability scores point towards systematic mistakes during tokenization. Here are a number of examples:[2]

R	C	n-gram	
0.00	1884	@ .	@ .
0.00	385	@ !	*@ !*
0.00	22	's advocaat	*'s lawyer*
0.11	8	H. 's	*H. 's*
0.00	98	@ , roept	*@ , yells*
0.00	20	@ , schreeuwt	*@ , screams*
0.00	469	@ , vraagt	*@ , asks*

The first and second n-gram indicate sentences which start with a full stop or an exclamation mark, due to a mistake in the tokenizer. The third and fourth n-grams indicate a problem the tokenizer had with a sequence of a single capital letter with a dot, followed by the genitive marker. The grammar assumes that the genitive marking is attached to the proper name. Such phrases occur frequently in reports on criminals, which are indicated in news paper only with their initials. Another systematic mistake is reflected by the last n-grams. In reported speech such as

(1) Je bent gek!, roept Franca.
You are crazy!, yells Franca.
Franca yells: You are crazy!

the tokenizer mistakenly introduced a sentence boundary between the exclamation mark and the comma. On the basis of examples such as these, the tokenizer has been improved.

Mistakes in the lexicon. Another reason an n-gram receives a low parsability score is a mistake in the lexicon. The following table lists two typical examples:

R	C	n-gram	
0.27	18	de kaft	*the cover*
0.30	7	heeft opgetreden	*has performed*

In Dutch, there is a distinction between neuter and non-neuter common nouns. The definite article *de* combines with non-neuter nouns, whereas neuter nouns select *het*. The common noun *kaft*, for example, combines with the definite article *de*. However, according to the dictionary, it is a neuter common noun (and thus would be expected to combine only with the definite article *het*). Many similar errors were discovered.

Another syntactic distinction that is listed in the dictionary is the distinction between verbs which take the auxiliary *hebben (to have)* to construct a perfect tense clause vs. those that take the auxiliary *zijn (to be)*. Some verbs allow both possibilities. The last example illustrates an error in the dictionary with respect to this syntactic feature.

[2]The @ symbol indicates a sentence boundary.

Incomplete lexical descriptions. The majority of problems that the parsability scores indicate reflect incomplete lexical entries. A number of examples is provided in the following table:

R	C	n-gram	
0.00	11	begunstigden	*favoured (N/V)*
0.23	10	zich eraan dat	*self there-on that*
0.08	12	aan te klikken	*on to click*
0.08	12	doodzonde dat	*mortal sin that*
0.15	11	zwarts	*black's*
0.00	16	dupe van	*victim of*
0.00	13	het Turks .	*the Turkish*

The word *begunstigden* is ambiguous between on the one hand the past tense of the verb *begunstigen (to favour)* and on the other hand the plural nominalization *begunstigden (beneficiaries)*. The dictionary contained only the first reading.

The sequence *zich eraan dat* illustrates a missing valency frame for verbs such as *ergeren* (to irritate). In Dutch, verbs which take a prepositional complement sometimes also allow the object of the prepositional complement to be realized by a subordinate (finite or infinite) clause. In that case, the prepositional complement is R-pronominalized. Examples:

(2) a. Hij ergert zich aan zijn aanwezigheid
He is-irritated self on his presence
He is irritated by his presence
b. Hij ergert zich er niet aan dat ...
He is-irritated self there not on that ...
He is not irritated by the fact that ...

The sequence *aan te klikken* is an example of a verb-particle combination which is not licensed in the dictionary. This is a relatively new verb which is used for *click* in the context of buttons and hyperlinks.

The sequence *doodzonde dat* illustrates a syntactic construction where a copula combines with a predicative complement and a sentential subject, if that predicative complement is of the appropriate type. This type is specified in the dictionary, but was missing in the case of *doodzonde*. Example:

(3) Het is doodzonde dat hij slaapt
It is mortal-sin that he sleeps
That he is sleeping is a pity

The word *zwarts* should have been analyzed as a genitive noun, as in (typically sentences about chess or checkers):

(4) Hij keek naar zwarts toren
He looked at black's rook

whereas the dictionary only assigned the inflected adjectival reading.

The sequence *dupe van* illustrates an example of an R-pronominalization of a PP modifier. This is generally not possible, except for (quite a large) number of contexts which are determined by the verb and the object:

(5) a. Hij is de dupe van jouw vergissing
He is the victim of your mistake
He has to suffer for your mistake
b. Hij is daar nu de dupe van
He is there now the victim of
He has to suffer for it

The word *Turks* can be both an adjective (*Turkish*) or a noun *the Turkish language*. The dictionary contained only the first reading.

Very many other examples of incomplete lexical entries were found.

Frozen expressions with idiosyncratic syntax. Dutch has many frozen expressions and idioms with archaic inflection and/or word order which breaks the parser. Examples include:

R	C	n-gram	
0.00	13	dan schaadt het	*then harms it*
0.00	13	@ God zij	*@ God be[I]*
0.22	25	God zij	*God be[I]*
0.00	19	Het zij zo	*It be[I] so*
0.45	12	goeden huize	*good house[I]*
0.09	11	berge	*mountain[I]*
0.00	10	hele gedwaald	*whole[I] dwelled*
0.00	14	te weeg	

The sequence *dan schaadt het* is part of the idiom *Baat het niet, dan schaadt het niet* (meaning: it might be unsure whether something is helpful, but in any case it won't do any harm). The sequence *God zij* is part of a number of archaic formulas such as *God zij dank* (Thank God). In such examples, the form *zij* is the (archaic) subjunctive form of the Dutch verb *zijn* (to be). The sequence *Het zij zo* is another fixed formula (English: *So be it*), containing the same subjunctive. The phrase *van goeden huize* (*of good family*) is a frozen expression with archaic inflection. The word *berge* exhibits archaic inflection on the word *berg* (*mountain*), which only occurs in the idiomatic expression *de haren rijzen mij te berge* (*my hair rises to the mountain*) which expresses a great deal of surprise. The n-gram *hele gedwaald* only occurs in the idiom *Beter ten halve gekeerd dan ten hele gedwaald*: it is better to turn halfway, then to go all the way in the wrong direc-

tion. Many other (parts of) idiomatic expressions were found in the parsability table.

The sequence *te weeg* only occurs as part of the phrasal verb *te weeg brengen* (to cause).

Incomplete grammatical descriptions. Although the technique strictly operates at the level of words and word sequences, it is capable of indicating grammatical constructions that are not treated, or not properly treated, in the grammar.

R	C	n-gram	
0.06	34	Wij Nederlanders	We Dutch
0.08	23	Geeft niet	Matters not
0.00	15	de alles	the everything
0.10	17	Het laten	The letting
0.00	10	tenzij .	unless .

The sequence *Wij Nederlanders* constitutes an example of a pronoun modified by means of an apposition (not allowed in the grammar) as in

(6) Wij Nederlanders eten vaak aardappels
We Dutch eat often potatoes
We, the Dutch, often eat potatoes

The sequence *Geeft niet* illustrates the syntactic phenomenon of topic-drop (not treated in the grammar): verb initial sentences in which the topic (typically the subject) is not spelled out. The sequence *de alles* occurs with present participles (used as prenominal modifiers) such as *overheersende* as in *de alles overheersende paniek* (literally: the all dominating panic, i.e., the panic that dominated everything). The grammar did not allow prenominal modifiers to select an NP complement. The sequence *Het laten* often occurs in nominalizations with multiple verbs. These were not treated in the grammar. Example:

(7) Het laten zien van problemen
The letting see of problems
Showing problems

The word sequence *tenzij .* is due to sentences in which a subordinate coordinator occurs without a complement clause:

(8) Gij zult niet doden, tenzij.
Thou shallt not kill, unless.

A large number of n-grams also indicate elliptical structures, not treated in that version of the grammar. Another fairly large source of errors are irregular named entities (*Gil y Gil, Osama bin Laden* . . .).

newspaper	# sentences	coverage %
NRC 1994	552,833	95.0
Volkskrant 1997	569,314	95,2
AD 2000	662,380	95,7
Trouw 1999	406,339	95,5
Volkskrant 2001	782,645	95,1

Table 2: Overview of corpus material used for the experiments; second experiment (January 2004).

3.2 Later experiment

Many of the errors and omissions that were found on the basis of the parsability table have been corrected. As can be seen in table 2, the coverage obtained by the improved parser increased substantially. In this experiment, we also measured the coverage on additional sets of sentences (all sentences from the Trouw 1999 and Volkskrant 2001 newspaper, available in the TwNC corpus). The results show that coverage is similar on these unseen testsets.

Obviously, coverage only indicates how often the parser found a full parse, but it does not indicate whether that parse actually was the correct parse. For this reason, we also closely monitored the performance of the parser on the Alpino tree-bank[3] (van der Beek et al., 2002a), both in terms of parsing accuracy and in terms of average number of parses per sentence. The average number of parses increased, which is to be expected if the grammar and lexicon are extended. Accuracy has been steadily increasing on the Alpino tree-bank. Accuracy is defined as the proportion of correct named dependency relations of the first parse returned by Alpino.

Alpino employs a maximum entropy disambiguation component; the first parse is the most promising parse according to this statistical model. The maximum entropy disambiguation component of Alpino assigns a score $S(x)$ to each parse x:

$$S(x) = \sum_i \theta_i f_i(x) \quad (1)$$

where $f_i(x)$ is the frequency of a particular feature i in parse x and θ_i is the corresponding weight of that feature. The probability of a parse x for sentence w is then defined as follows, where $Y(w)$ are all the parses of w:

$$p(x|w) = \frac{\exp(S(x))}{\sum_{y \in Y(w)} \exp(S(y))} \quad (2)$$

The disambiguation component is described in detail in Malouf and van Noord (2004).

[3] http://www.let.rug.nl/~vannoord/trees/

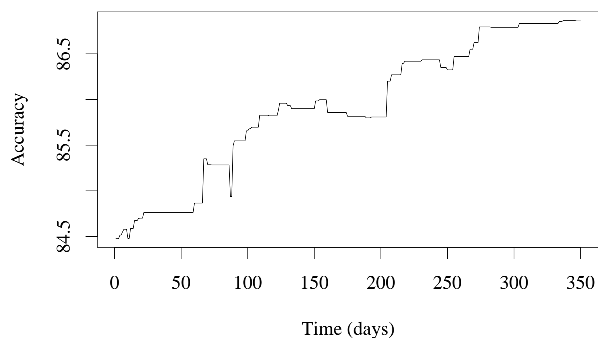

Figure 2: Development of Accuracy of the Alpino parser on the Alpino Tree-bank

Figure 2 displays the accuracy from May 2003-May 2004. During this period many of the problems described earlier were solved, but other parts of the system were improved too (in particular, the disambiguation component was improved considerably). The point of the graph is that apparently the increase in coverage has not been obtained at the cost of decreasing accuracy.

4 A note on the implementation

The most demanding part of the implementation consists of the computation of the frequency of n-grams. If the corpus is large, or n increases, simple techniques break down. For example, an approach in which a hash data-structure is used to maintain the counts of each n-gram, and which increments the counts of each n-gram that is encountered, requires excessive amounts of memory for large n and/or for large corpora. On the other hand, if a more compact data-structure is used, speed becomes an issue. Church (1995) shows that *suffix arrays* can be used for efficiently computing the frequency of n-grams, in particular for larger n. If the corpus size increases, the memory required for the suffix array may become problematic. We propose a new combination of suffix arrays with perfect hash finite automata, which reduces typical memory requirements by a factor of five, in combination with a modest increase in processing efficiency.

4.1 Suffix arrays

Suffix arrays (Manber and Myers, 1990; Yamamoto and Church, 2001) are a simple, but useful data-structure for various text-processing tasks. A corpus is a sequence of characters. A suffix array s is an array consisting of all suffixes of the corpus, sorted alphabetically. For example, if the corpus is the string `abba`, the suffix array is $\langle$`a`,`abba`,`ba`,`bba`$\rangle$. Rather than writing out each suffix, we use integers i to refer to the suffix starting at position i in the corpus. Thus, in this case the suffix array consists of the integers $\langle 3, 0, 2, 1 \rangle$.

It is straightforward to compute the suffix array. For a corpus of $k + 1$ characters, we initialize the suffix array by the integers $0 \ldots k$. The suffix array is sorted, using a specialized comparison routine which takes integers i and j, and alphabetically compares the strings starting at i and j in the corpus.[4]

Once we have the suffix array, it is simple to compute the frequency of n-grams. Suppose we are interested in the frequency of all n-grams for $n = 10$. We simply iterate over the elements of the suffix array: for each element, we print the first ten words of the corresponding suffix. This gives us all occurrences of all 10-grams in the corpus, sorted alphabetically. We now count each 10-gram, e.g. by piping the result to the Unix `uniq -c` command.

4.2 Perfect hash finite automata

Suffix arrays can be used more efficiently to compute frequencies of n-grams for larger n, with the help of an additional data-structure, known as the *perfect hash* finite automaton (Lucchiesi and Kowaltowski, 1993; Roche, 1995; Revuz, 1991). The perfect hash automaton for an alphabetically sorted finite set of words $w_0 \ldots w_n$ is a weighted minimal deterministic finite automaton which maps $w_i \rightarrow i$ for each $w_{0 \leq i \leq n}$. We call i the *word code* of w_i. An example is given in figure 3.

Note that perfect hash automata implement an order preserving, minimal perfect hash function. The function is minimal, in the sense that n keys are mapped into the range $0 \ldots n - 1$, and the function is order preserving, in the sense that the alphabetic order of words is reflected in the numeric order of word codes.

4.3 Suffix arrays with words

In the approach of Church (1995), the corpus is a sequence of characters (represented by integers reflecting the alphabetic order). A more space-efficient approach takes the corpus as a sequence of words, represented by word codes reflecting the alphabetic order.

To compute frequencies of n-grams for larger n, we first compute the perfect hash finite automaton for all words which occur in the corpus,[5] and map

[4] The suffix sort algorithm of Peter M. McIlroy and M. Douglas McIlroy is used, available as `http://www.cs.dartmouth.edu/~doug/ssort.c`; This algorithm is robust against long repeated substrings in the corpus.

[5] We use an implementation by Jan Daciuk freely available from `http://www.eti.pg.gda.pl/~jandac/fsa.html`.

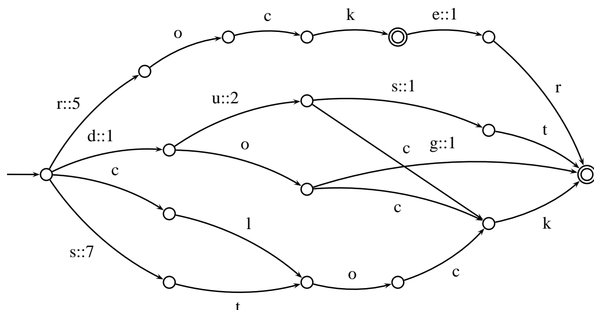

Figure 3: Example of a perfect hash finite automaton for the words *clock, dock, dog, duck, dust, rock, rocker, stock*. Summing the weights along an accepting path in the automaton yields the rank of the word in alphabetic ordering.

the corpus to a sequence of integers, by mapping each word to its word code. Suffix array construction then proceeds on the basis of word codes, rather than character codes.

This approach has several advantages. The representation of both the corpus and the suffix array is more compact. If the average word length is k, then the corresponding arrays are k times smaller (but we need some additional space for the perfect hash automaton). In Dutch, the average word length k is about 5, and we obtained space savings in that order.

If the suffix array is shorter, sorting should be faster too (but we need some additional time to compute the perfect hash automaton). In our experience, sorting is about twice as fast for word codes.

4.4 Computing parsability table

To compute parsability scores, we assume there are two corpora c_m and c_a, where the first is a subcorpus of the second. c_m contains all sentences for which parsing was not successful. c_a contains all sentences overall. For both corpora, we compute the frequency of all n-grams for all n; n-grams with a frequency below a specified frequency cut-off are ignored. Note that we need not impose an a priori maximum value for n; since there is a frequency cut-off, for some n there simply aren't any sequences which occur more frequently than this cut-off. The two n-gram frequency files are organized in such a way that shorter n-grams precede longer n-grams.

The two frequency files are then combined as follows. Since the frequency file corresponding to c_m is (much) smaller than the file corresponding to c_a, we read the first file into memory (into a hash data structure). We then iteratively read an n-gram frequency from the second file, and compute the parsability of that n-gram. In doing so, we keep track of the parsability scores assigned to previous (hence shorter) n-grams, in order to ensure that larger n-grams are only reported in case the parsability scores decrease. The final step consists in sorting all remaining n-grams with respect to their parsability.

To give an idea of the practicality of the approach, consider the following data for one of the experiments described above. For a corpus of 2,927,016 sentences (38,846,604 words, 209Mb), it takes about 150 seconds to construct the perfect hash automaton (mostly sorting). The automaton is about 5Mb in size, to represent 677,488 distinct words. To compute the suffix array and frequencies of all n-grams (cut-off=5), about 15 minutes of CPU-time are required. Maximum runtime memory requirements are about 400Mb. The result contains frequencies for 1,641,608 distinct n-grams. Constructing the parsability scores on the basis of the n-gram files only takes 10 seconds CPU-time, resulting in parsability scores for 64,998 n-grams (since there are much fewer n-grams which actually occur in problematic sentences). The experiment was performed on a Intel Pentium III, 1266MHz machine running Linux. The software is freely available from `http://www.let.rug.nl/~vannoord/software.html`.

5 Discussion

An error mining technique has been presented which is very helpful in identifying problems in hand-coded grammars and lexicons for parsing. An important ingredient of the technique consists of the computation of the frequency of n-grams of words for arbitrary values of n. It was shown how a new combination of suffix arrays and perfect hash finite automata allows an efficient implementation. A number of potential improvements can be envisioned.

In the definition of $R(w)$, the absolute frequency of w is ignored. Yet, if w is very frequent, $R(w)$ is more reliable than if w is not frequent. Therefore, as an alternative, we also experimented with a set-up in which an exact binomial test is applied to compute a confidence interval for $R(w)$. Results can then be ordered with respect to the maximum of these confidence intervals. This procedure seemed to improve results somewhat, but is computationally much more expensive. For the first experiment described above, this alternative set-up results in a parsability table of 42K word tuples, whereas the original method produces a table of 65K word tuples.

R	C	n-gram
0.00	8	Beter ten
0.20	12	ten halve
0.15	11	halve gekeerd
0.00	8	gekeerd dan
0.09	10	dan ten hele
0.69	15	dan ten
0.17	10	ten hele
0.00	10	hele gedwaald
0.00	8	gedwaald .
0.20	10	gedwaald

Table 3: Multiple n-grams indicating same error

The parsability table only contains longer n-grams if these have a lower parsability than the corresponding shorter n-grams. Although this heuristic appears to be useful, it is still possible that a single problem is reflected multiple times in the parsability table. For longer problematic sequences, the parsability table typically contains partially overlapping parts of that sequence. This phenomenon is illustrated in table 3 for the idiom *Beter ten halve gekeerd dan ten hele gedwaald* discussed earlier. This suggests that it would be useful to consider other heuristics to eliminate such redundancy, perhaps by considering statistical feature selection methods.

The definition used in this paper to identify a successful parse is a rather crude one. Given that grammars of the type assumed here typically assign very many analyses to a given sentence, it is often the case that a specific problem in the grammar or lexicon rules out the intended parse for a given sentence, but alternative (wrong) parses are still possible. What appears to be required is a (statistical) model which is capable of judging the plausibility of a parse. We investigated whether the maximum entropy score $S(x)$ (equation 1) can be used to indicate parse plausibility. In this set-up, we considered a parse successful only if $S(x)$ of the best parse is above a certain threshold. However, the resulting parsability table did not appear to indicate problematic word sequences, but rather word sequences typically found in elliptical sentences were returned. Apparently, the grammatical rules used for ellipsis are heavily punished by the maximum entropy model in order that these rules are used only if other rules are not applicable.

Acknowledgments

This research was supported by the PIONIER project *Algorithms for Linguistic Processing* funded by NWO.

References

Gosse Bouma, Gertjan van Noord, and Robert Malouf. 2001. Wide coverage computational analysis of Dutch. In W. Daelemans, K. Sima'an, J. Veenstra, and J. Zavrel, editors, *Computational Linguistics in the Netherlands 2000*.

Kenneth Ward Church. 1995. Ngrams. ACL 1995, MIT Cambridge MA, June 16. ACL Tutorial.

Claudio Lucchiesi and Tomasz Kowaltowski. 1993. Applications of finite automata representing large vocabularies. *Software Practice and Experience*, 23(1):15–30, Jan.

Robert Malouf and Gertjan van Noord. 2004. Wide coverage parsing with stochastic attribute value grammars. In *Beyond shallow analyses. Formalisms and statistical modeling for deep analysis*, Sanya City, Hainan, China. IJCNLP-04 Workshop.

Udi Manber and Gene Myers. 1990. Suffix arrays: A new method for on-line string searching. In *Proceedings of the First Annual AC-SIAM Symposium on Discrete Algorithms*, pages 319–327. `http://manber.com/publications.html`.

Robbert Prins and Gertjan van Noord. 2003. Reinforcing parser preferences through tagging. *Traitement Automatique des Langues*, 44(3):121–139. in press.

Dominique Revuz. 1991. *Dictionnaires et lexiques: méthodes et algorithmes*. Ph.D. thesis, Institut Blaise Pascal, Paris, France. LITP 91.44.

Emmanuel Roche. 1995. Finite-state tools for language processing. ACL 1995, MIT Cambridge MA, June 16. ACL Tutorial.

Leonoor van der Beek, Gosse Bouma, Robert Malouf, and Gertjan van Noord. 2002a. The Alpino dependency treebank. In Mariët Theune, Anton Nijholt, and Hendri Hondorp, editors, *Computational Linguistics in the Netherlands 2001. Selected Papers from the Twelfth CLIN Meeting*, pages 8–22. Rodopi.

Leonoor van der Beek, Gosse Bouma, and Gertjan van Noord. 2002b. Een brede computationele grammatica voor het Nederlands. *Nederlandse Taalkunde*, 7(4):353–374. in Dutch.

Mikio Yamamoto and Kenneth W. Church. 2001. Using suffix arrays to compute term frequency and document frequency for all substrings in a corpus. *Computational Linguistics*, 27(1):1–30.