Fast Statistical Parsing with Parallel Multiple Context-Free Grammars

Krasimir Angelov and **Peter Ljunglöf**
University of Gothenburg and Chalmers University of Technology
Göteborg, Sweden
krasimir@chalmers.se
peter.ljunglof@cse.gu.se

Abstract

We present an algorithm for incremental statistical parsing with Parallel Multiple Context-Free Grammars (PMCFG). This is an extension of the algorithm by Angelov (2009) to which we added statistical ranking. We show that the new algorithm is several times faster than other statistical PMCFG parsing algorithms on real-sized grammars. At the same time the algorithm is more general since it supports non-binarized and non-linear grammars.

We also show that if we make the search heuristics non-admissible, the parsing speed improves even further, at the risk of returning sub-optimal solutions.

1 Introduction

In this paper we present an algorithm for incremental parsing using Parallel Multiple Context-Free Grammars (PMCFG) (Seki et al., 1991). This is a non context-free formalism allowing discontinuity and crossing dependencies, while remaining with polynomial parsing complexity.

The algorithm is an extension of the algorithm by Angelov (2009; 2011) which adds statistical ranking. This is a top-down algorithm, shown by Ljunglöf (2012) to be similar to other top-down algorithms (Burden and Ljunglöf, 2005; Kanazawa, 2008; Kallmeyer and Maier, 2009). None of the other top-down algorithms are statistical.

The only statistical PMCFG parsing algorithms (Kato et al., 2006; Kallmeyer and Maier, 2013; Maier et al., 2012) all use bottom-up parsing strategies. Furthermore, they require the grammar to be binarized and linear, which means that they only support linear context-free rewriting systems (LCFRS). In contrast, our algorithm naturally supports the full power of PMCFG. By lifting these restrictions, we make it possible to experiment with novel grammar induction methods (Maier, 2013) and to use statistical disambiguation for hand-crafted grammars (Angelov, 2011).

By extending the algorithm with a statistical model, we allow the parser to explore only parts of the search space, when only the most probable parse tree is needed. Our cost estimation is similar to the estimation for the Viterbi probability as in Stolcke (1995), except that we have to take into account that our grammar is not context-free. The estimation is both admissible and monotonic (Klein and Manning, 2003) which guarantees that we always find a tree whose probability is the global maximum.

We also describe a variant with a non-admissible estimation, which further improves the efficiency of the parser at the risk of returning a suboptimal parse tree.

We start with a formal definition of a weighted PMCFG in Section 2, and we continue with a presentation of our algorithm by means of a weighted deduction system in Section 3. In Section 4, we prove that our estimations are admissible and monotonic. In Section 5 we calculate an estimate for the minimal inside probability for every category, and in Section 6 we discuss the non-admissible heuristics. Sections 7 and 8 describe the implementation and our evaluation, and the final Section 9 concludes the paper.

2 PMCFG definition

Our definition of weighted PMCFG (Definition 1) is the same as the one used by Angelov (2009; 2011), except that we extend it with weights for the productions. This definition is also similar to Kato et al (2006), with the small difference that we allow non-linear functions.

As an illustration for PMCFG parsing, we use a simple grammar (Figure 1) which can generate phrases like *"both black and white"* and *"either red or white"* but rejects the incorrect combina-

Definition 1
A parallel multiple context-free grammar is a tuple $G = (N, T, F, P, S, d, d_i, r, a)$ *where:*

- N *is a finite set of categories and a positive integer* $d(A)$ *called dimension is given for each* $A \in N$.
- T *is a finite set of terminal symbols which is disjoint with* N.
- F *is a finite set of functions where the arity* $a(f)$ *and the dimensions* $r(f)$ *and* $d_i(f)$ $(1 \leq i \leq a(f))$ *are given for every* $f \in F$. *For every positive integer* d, $(T^*)^d$ *denote the set of all* d*-tuples of strings over* T. *Each function* $f \in F$ *is a total mapping from* $(T^*)^{d_1(f)} \times (T^*)^{d_2(f)} \times \cdots \times (T^*)^{d_{a(f)}(f)}$ *to* $(T^*)^{r(f)}$, *defined as:*

$$f := (\alpha_1, \alpha_2, \ldots, \alpha_{r(f)})$$

Here α_i *is a sequence of terminals and* $\langle k; l\rangle$ *pairs, where* $1 \leq k \leq a(f)$ *is called argument index and* $1 \leq l \leq d_k(f)$ *is called constituent index.*

- P *is a finite set of productions of the form:*

$$A \xrightarrow{w} f[A_1, A_2, \ldots, A_{a(f)}]$$

where $A \in N$ *is called result category,* $A_1, A_2, \ldots, A_{a(f)} \in N$ *are called argument categories,* $f \in F$ *is the function symbol and* $w > 0$ *is a weight. For the production to be well formed the conditions* $d_i(f) = d(A_i)$ $(1 \leq i \leq a(f))$ *and* $r(f) = d(A)$ *must hold.*

- S *is the start category and* $d(S) = 1$.

tions *both-or* and *either-and*. We avoid these combinations by coupling the right pairs of words in a single function, i.e. we have the abstract conjunctions *both_and* and *either_or* which are linearized as discontinuous phrases. The phrase insertion itself is done in the definition of *conjA*. It takes the conjunction as its first argument, and it uses $\langle 1; 1\rangle$ and $\langle 1; 2\rangle$ to insert the first and the second constituent of the argument at the right places in the complete phrase.

A tree of function applications that yelds a complete phrase is the parse tree for the phrase. For instance, the phrase *"both red and either black or white"* is represented by the tree:

(*conjA both_and red*
(*conjA either_or black white*))

$$\begin{aligned}
A &\xrightarrow{w_1} \mathit{conjA}[\mathit{Conj}, A, A] \\
A &\xrightarrow{w_2} \mathit{black}[] \\
A &\xrightarrow{w_3} \mathit{white}[] \\
A &\xrightarrow{w_4} \mathit{red}[] \\
\mathit{Conj} &\xrightarrow{w_5} \mathit{both_and}[] \\
\mathit{Conj} &\xrightarrow{w_6} \mathit{either_or}[]
\end{aligned}$$

$$\begin{aligned}
\mathit{conjA} &:= (\langle 1;1\rangle\langle 2;1\rangle\langle 1;2\rangle\langle 3;1\rangle) \\
\mathit{black} &:= (\text{"black"}) \\
\mathit{white} &:= (\text{"white"}) \\
\mathit{red} &:= (\text{"red"}) \\
\mathit{both_and} &:= (\text{"both"}, \text{"and"}) \\
\mathit{either_or} &:= (\text{"either"}, \text{"or"})
\end{aligned}$$

Figure 1: Example Grammar

The weight of a tree is the sum of the weights for all functions that are used in it. In this case the weight for the example is $w_1+w_5+w_4+w_1+w_6+w_2+w_3$. If there are ambiguities in the sentence, the algorithm described in Section 3 always finds a tree which minimizes the weight.

Usually the weights for the productions are logarithmic probabilities, i.e. the weight of the production $A \to f[\vec{B}]$ is:

$$w = -\log P(A \to f[\vec{B}] \mid A)$$

where $P(A \to f[\vec{B}] \mid A)$ is the probability to choose this production when the result category is fixed. In this case the probabilities for all productions with the same result category sum to one:

$$\sum_{A \xrightarrow{w} f[\vec{B}] \in P} e^{-w} = 1$$

However, the parsing algorithm does not depend on the probabilistic interpretation of the weights, so the same algorithm can be used with any other kind of weights.

3 Deduction System

We define the algorithm as weighted deduction system (Nederhof, 2003) which generalizes Angelov's system.

A key feature in his algorithm is that the expressive PMCFG is reduced to a simple context-free grammar which is extended dynamically at parsing time in order to account for context dependent features in the original grammar. This

can be exemplified with the grammar in Figure 1, where there are two productions for category *Conj*. Given the phrase *"both black and white"*, after accepting the token *both*, only the production $\mathit{Conj} \xrightarrow{w_5} \mathit{both\_and}[]$ can be applied for parsing the second part of the conjunction. This is achieved by generating a new category Conj_2 which has just a single production:

$$\mathit{Conj}_2 \xrightarrow{w_5} \mathit{both\_and}[] \tag{1}$$

The parsing algorithm is basically an extension of Earley's (1970) algorithm, except that the parse items in the chart also keep track of the categories for the arguments. In the particular case, the corresponding chart item will be updated to point to Conj_2 instead of *Conj*. This guarantees that only *and* will be accepted as a second constituent after seeing that the first constituent is *both*.

Now since the set of productions is dynamic, the parser must keep three kinds of items in the chart, instead of two as in the Earley algorithm:

Productions The parser maintains a dynamic set with all productions that are derived during the parsing. The initial state is populated with the productions from the set P in the grammar.

Active Items The active items play the same role as the active items in the Earley algorithm. They have the form:

$$[{}_j^k A \xrightarrow{w} f[\vec{B}]; l : \alpha \bullet \beta; w_i; w_o]$$

and represent the fact that a constituent l of a category A has been partially recognized from position j to k in the sentence. Here $A \xrightarrow{w} f[\vec{B}]$ is the production and the concatenation $\alpha\beta$ is the sequence of terminals and $\langle k; r\rangle$ pairs which defines the l-th constituent of function f. The dot $\bullet$ between α and β separates the part of the constituent that is already recognized from the part which is still pending. Finally w_i and w_o are the inside and outside weights for the item.

Passive Items The passive items are of the form:

$$[{}_j^k A; l; \hat{A}]$$

and state that a constituent with index l from category A was recognized from position j to position k in the sentence. As a consequence the parser has created a new category $\hat{A}$. The set of productions derived for $\hat{A}$ compactly records all possible ways to parse the $j - k$ fragment.

3.1 Inside and outside weights

The inside weight w_i and the outside weight w_o in the active items deserve more attention since this is the only difference compared to Angelov (2009; 2011). When the item is complete, it will yield the forest of all trees that derive the sub-string covered by the item. For example, when the first constituent for category *Conj* is completely parsed, the forest will contain the single production in (1). The inside weight for the active item is the currently best known estimation for the lowest weight of a tree in the forest. The trees yielded by the item do not cover the whole sentence however. Instead, they will become part of larger trees that cover the whole sentence. The outside weight is the estimation for the lowest weight for an extension of a tree to a full tree. The sum $w_i + w_o$ estimates the weight of the full tree.

Before turning to the deduction rules we also need a notation for the lowest possible weight for a tree of a given category. If $A \in N$ is a category then w_A will denote the lowest weight that a tree of category A can have. For convenience, we also use $w_{\vec{B}}$ as a notation for the sum $\sum_i w_{B_i}$ of the weight of all categories in the vector $\vec{B}$. If the category A is defined in the grammar then we assume that the weight is precomputed as described in Section 5. When the parser creates the category, it will compute the weight dynamically.

3.2 Deduction rules

The deduction rules are shown in Figure 2. Here the assumption is that the active items are processed in the order of increasing $w_i + w_o$ weight. In the actual implementation we put all active items in a priority queue and we always take first the item with the lowest weight. We never throw away items but the processing of items with very high weight might be delayed indefinitely or they may never be processed if the best tree is found before that. Furthermore, we think of the deduction system as a way do derive a set of items, but in our case we ignore the weights when we consider whether two active items are the same. In this way, every item is derived only once and the weights for the active items are computed from the weights of the first antecedents that led to its derivation.

Finally, we use two more notations in the rules: $\mathrm{rhs}(g, r)$ denotes constituent with index r in function g; and ω_k denotes the k-th token in the sentence.

INITIAL PREDICT

$$\frac{S \xrightarrow{w} f[\vec{B}]}{[^0_0 S \xrightarrow{w} f[\vec{B}]; 1 : \bullet\, \gamma; w + w_{\vec{B}}; 0]} \quad S = \text{start category},\ \gamma = \text{rhs}(f, 1)$$

PREDICT

$$\frac{B_d \xrightarrow{w_1} g[\vec{C}] \quad [^k_j A \xrightarrow{w_2} f[\vec{B}]; l : \alpha \bullet \langle d; r\rangle\, \beta; w_i; w_o]}{[^k_k B_d \xrightarrow{w_1} g[\vec{C}]; r : \bullet\, \gamma; w_1 + w_{\vec{C}}; w_i - w_{B_d} + w_o]} \quad \gamma = \text{rhs}(g, r)$$

SCAN

$$\frac{[^k_j A \xrightarrow{w} f[\vec{B}]; l : \alpha \bullet s\, \beta; w_i; w_o]}{[^{k+1}_j A \xrightarrow{w} f[\vec{B}]; l : \alpha\, s \bullet \beta; w_i; w_o]} \quad s = \omega_{k+1}$$

COMPLETE

$$\frac{[^k_j A \xrightarrow{w} f[\vec{B}]; l : \alpha \bullet; w_i; w_o]}{\hat{A} \xrightarrow{w} f[\vec{B}] \qquad [^k_j A; l; \hat{A}]} \quad \hat{A} = (A, l, j, k),\ w_{\hat{A}} = w_i$$

COMBINE

$$\frac{[^u_j A \xrightarrow{w} f[\vec{B}]; l : \alpha \bullet \langle d; r\rangle\, \beta; w_i; w_o] \quad [^k_u B_d; r; \hat{B}_d]}{[^k_j A \xrightarrow{w} f[\vec{B}\{d := \hat{B}_d\}]; l : \alpha\, \langle d; r\rangle \bullet \beta; w_i + w_{\hat{B}_d} - w_{B_d}; w_o]}$$

Figure 2: Deduction Rules

The first rule on Figure 2 is INITIAL PREDICT and here we predict the initial active items from the productions for the start category S. Since this is the start category, we set the outside weight to zero. The inside weight is equal to the sum of the weight w for the production and the lowest possible weight $w_{\vec{B}}$ for the vector of arguments $\vec{B}$. The reason is that despite that we do not know the weight for the final tree yet, it cannot be lower than $w + w_{\vec{B}}$ since $w_{\vec{B}}$ is the lowest possible weight for the arguments of function f.

The interaction between inside and outside weights is more interesting in the PREDICT rule. Here we have an item where the dot is before $\langle d; r\rangle$ and from this we must predict one item for each production $B_d \xrightarrow{w_1} g[\vec{C}]$ of category B_d. The inside weight for the new item is $w_1 + w_{\vec{C}}$ for the same reasons as for the INITIAL PREDICT rule. The outside weight however is not zero because the new item is predicted from another item. The inside weight for the active item in the antecedents is now part of the outside weight of the new item. We just have to subtract w_{B_d} from w_i because the new item is going to produce a new tree which will replace the d-th argument of f. For this reason the estimation for the outside weight is $w_i - w_{B_d} + w_o$, where we also added the outside weight for the antecedent item.

In the SCAN rule, we just move the dot past a token, if it matches the current token ω_{k+1}. Both the inside and the outside weights are passed untouched from the antecedent to the consequent.

In the COMPLETE rule, we have an item where the dot has reached the end of the constituent. Here we generate a new category $\hat{A}$ which is unique for the combination (A, l, j, k), and we derive the production $\hat{A} \xrightarrow{w} f[\vec{B}]$ for it. We set the weight $w_{\hat{A}}$ for $\hat{A}$ to be equal to w_i and in Section 4, we will prove that this is indeed the lowest weight for a tree of category $\hat{A}$.

In the last rule COMBINE, we combine an active item with a passive item. The outside weight w_o for the new active item remains the same. However, we must update the inside weight since we have replaced the d-th argument in $\vec{B}$ with the newly generated category $\hat{B}_d$. The new weight is $w_i + w_{\hat{B}_d} - w_{B_d}$, i.e. we add the weight for the new category and we subtract the weight for the previous category B_d.

Now for the correctness of the weights we must prove that the estimations are both admissible and monotonic.

4 Admissibility and Monotonicity

We will first prove that the weights grow monotonically, i.e. if we derive one active item from another then the sum $w_i + w_o$ for the new item is always greater or equal to the sum for the previous

item. PREDICT and COMBINE are the only two rules with an active item both in the antecedents and in the consequents.

Note that in PREDICT we choose one particular production for category B_d. We know that the lowest possible weight of a tree of this category is w_{B_d}. If we restrict the set of trees to those that not only have the same category B_d but also use the same production $B_d \xrightarrow{w_1} g[\vec{C}]$ on the top level, then the best weight for such a tree will be $w_1 + w_{\vec{C}}$. According to the definition of w_{B_d}, it must follow that:

$$w_1 + w_{\vec{C}} \geq w_{B_d}$$

From this we can trivially derive that:

$$(w_1 + w_{\vec{C}}) + (w_i - w_{B_d} + w_o) \geq w_i + w_o$$

which is the monotonicity condition for rule PREDICT. Similarly in rule COMBINE, the condition:

$$w_{\hat{B}_d} \geq w_{B_d}$$

must hold because the forest of trees for $\hat{B}_d$ is included in the forest for B_d. From this we conclude the monotonicity condition:

$$(w_i + w_{\hat{B}_d} - w_{B_d}) + w_o \geq w_i + w_o$$

The last two inequalities are valid only if we can correctly compute $w_{\hat{B}_d}$ for a dynamically generated category $\hat{B}_d$. This happens in rule COMPLETE, where we have a complete active item with a correctly computed inside weight w_i. Since we process the active items in the order of increasing $w_i + w_o$ weight and since we create $\hat{A}$ when we find the first complete item for category A, it is guaranteed that at this point we have an item with minimal $w_i + w_o$ value. Furthermore, all items with the same result category A and the same start position j must have the same outside weight. It follows that when we create $\hat{A}$ we actually do it from an active item with minimal inside weight w_i. This means that it is safe to assign that $w_{\hat{A}} = w_i$.

It is also easy to see that the estimation is admissible. The only places where we use estimations for the unseen parts of the sentence is in the rules INITIAL PREDICT and PREDICT where we use the weights $w_{\vec{B}}$ and $w_{\vec{C}}$ which may include components corresponding to function argument that are not seen yet. However by definition it is not possible to build a tree with weight lower than the weight for the category. This means that the estimation is always admissible.

5 Initial Estimation

The minimal weight for a dynamically created category is computed by the parser, but we must initialize the weights for the categories that are defined in the grammar. The easiest way is to just set all weights to zero, and this is safe since the weights for the predefined categories are used only as estimations for the yet unseen parts of the sentence. Essentially this gives us a statistical parser which performs Dijkstra search in the space of all parse trees. Any other reasonable weight assignment will give us an A^* algorithm (Hart et al., 1968).

In general it is possible to devise different heuristics which will give us different improvements in the parsing time. In our current implementation of the parser we use a weight assignment which considers only the already known probabilities for the productions in the grammar.

The weight for a category A is computed as:

$$w_A = \min_{A \xrightarrow{w} f[\vec{B}] \in P} (w + w_{\vec{B}})$$

Here the sum $w + w_{\vec{B}}$ is the minimal weight for a tree constructed with the production $A \xrightarrow{w} f[\vec{B}]$ at the root. By taking the minimum over all productions for A, we get the corresponding weight w_A. This is a recursive equation since its right-hand side contains the value $w_{\vec{B}}$ which depends on the weights for the categories in $\vec{B}$. It might happen that there are mutually dependent categories which will lead to a recursion in the equation.

The solution is found with iterative assignments until a fixed point is reached. In the beginning we assign $w_A = 0$ for all categories. After that we recompute the new weights with the equation above until we reach a fixed point.

6 Non-admissible heuristics

The set of active items is kept in a priority queue and at each step we process the item with the lowest weight. However, when we experimented with the algorithm we noticed that most of the time the item that is selected would eventually contribute with an alternative reading of the sentence but not to the best parse. What happens is that despite that there are already items ending at position k in the sentence, the current best item might have a span $i - j$ where $j < k$. The parser then picks the best item only to discover later that the item became much heavier until it reached the span $i - k$.

This suggests that when we compare the weights of items with different end positions, then we must take into account the weight that will be accumulated by the item that ends earlier until the two items align at the same end position.

We use the following heuristic to estimate the difference. The first time when we extend an item from position i to position $i+1$, we record the weight increment $w_\Delta(i+1)$ for that position. The increment w_Δ is the difference between the weights for the best active item reaching position $i+1$ and the best active item reaching position i. From now on when we compare the weights for two items x_j and x_k, with end positions j and k respectively ($j < k$), then we always add to the score w_{x_j} of the first item a fraction of the sum of the increments for the positions between j and k. In other words, instead of using w_{x_j} when comparing with w_{x_k}, we use

$$w_{x_j} + h \cdot \sum_{j<i\leq k} w_\Delta(i)$$

We call the constant $h \in [0,1]$ the "heuristics factor". If $h = 0$, we obtain the basic algorithm that we described earlier which is admissible and always returns the best parse. However, the evaluation in Section 8.3 shows that a significant speedup can be obtained by using larger values of h. Unfortunately, if $h > 0$, we loose some accuracy and cannot guarantee that the best parse is always returned first.

Note that the heuristics does not change the completeness of the algorithm – it will succeed for all grammatical sentences and fail for all non-grammatical. But it does not guarantee that the first parse tree will be the optimal.

7 Implementation

The parser is implemented in C and is distributed as a part of the runtime system for the open-source Grammatical Framework (GF) programming language (Ranta, 2011).[1] Although the primary application of the runtime system is to run GF applications, it is not specific to one formalism, and it can serve as an execution platform for other frameworks where natural language parsing and generation is needed.

The GF system is distributed with a library of manually authored resource grammars (Ranta, 2009) for over 25 languages, which are used as a resource for deriving domain specific grammars. Adding a big lexicon to the resource grammar results in a highly ambiguous grammar, which can give rise to millions of trees even for moderately complex sentences. Previously, the GF system has not been able to parse with such ambiguous grammars, but with our statistical algorithm it is now feasible.

8 Evaluation

We did an initial evaluation on the GF English resource grammar augmented with a large-coverage lexicon of 40 000 lemmas taken from the Oxford Advanced Learner's Dictionary (Mitton, 1986). In total the grammar has 44 000 productions. The rule weights were trained from a version of the Penn Treebank (Marcus et al., 1993) which was converted to trees compatible with the grammar.

The trained grammar was tested on Penn Treebank sentences of length up to 35 tokens, and the parsing times were at most 7 seconds per sentence. This initial test was run on a computer with a 2.4 GHz Intel Core i5 processor with 8 GB RAM. This result was very encouraging, given the complexity of the grammar, so we decided to do a larger test and compare with an existing state-of-the-art statistical PMCFG parser.

Rparse (Kallmeyer and Maier, 2013) is a another state-of-the-art training and parsing system for PMCFG.[2] It is written in Java and developed at the Universities of Tübingen and Düsseldorf, Germany. Rparse can be used for training probabilistic PMCFGs from discontinuous treebanks. It can also be used for parsing new sentences with the trained grammars.

In our evaluation we used Rparse to extract PMCFG grammars from the discontinuous German Tiger Treebank (Brants et al., 2002). The reason for using this treebank is that the extracted grammars are non-context-free, and our parser is specifically made for such grammars.

8.1 Evaluation data

In our evaluations we got the same general results regardless of the size of the grammar, so we only report the results from one of these runs.

In this particular example, we trained the grammar on 40 000 sentences from the Tiger Treebank with lengths up to 160 tokens. We evaluated on

[1] http://www.grammaticalframework.org/

[2] https://github.com/wmaier/rparse

	Count
Training sentences	40 000
Test sentences	4 607
Non-binarized grammar rules	30 863
Binarized grammar rules	26 111

Table 1: Training and testing data.

4 600 Tiger sentences, with a length of 5–60 tokens. The exact numbers are shown in Table 1. All tests were run on a computer with a 2.3 GHz Intel Core i7 processor with 16GB RAM.

As a comparison, Maier et al (2012) train on approximately 15 000 sentences from the Negra Treebank, and only evaluate on sentences of at most 40 tokens.

8.2 Comparison with Rparse

We evaluated our parser by comparing it with Rparse's built-in parser. Note that we are only interested in the efficiency of our implementation, not the coverage and accuracy of the trained grammar. In the comparison we used only the admissible heuristics, and we did confirm that the parsers produce optimal trees with exactly the same weight for the same input.

Rparse extracts grammars in two steps. First it converts the treebank into a PMCFG, and then it binarizes that grammar. The binarization process uses markovization to improve the precision and recall of the final grammar (Kallmeyer and Maier, 2013). We tested both Rparse's standard (Kallmeyer and Maier, 2013) and its new improved parsing alogorithm (Maier et al., 2012). The new algorithm unfortunately works only with LCFRS grammars with a fan-out ≤ 2 (Maier et al., 2012).

In this test we used the optimal binarization method described in Kallmeyer (2010, chapter 7.2). This was the only binarization algorithm in Rparse that produced a grammar with fan-out ≤ 2.

As can be seen in Figure 3, our parser outperforms Rparse for all sentence lengths. For sentences longer than 15 tokens, the standard Rparse parser needs on average 100 times longer time than our parser. This difference increases with sentence length, suggesting that our algorithm has a better parsing complexity than Rparse.

The PGF parser also outperforms the improved Rparse parser, but the relative difference seems to stabilize on a speedup of 10–15 times.

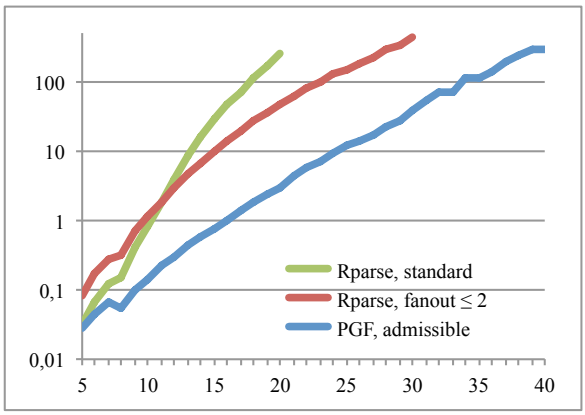

Figure 3: Parsing time (seconds) compared with Rparse.

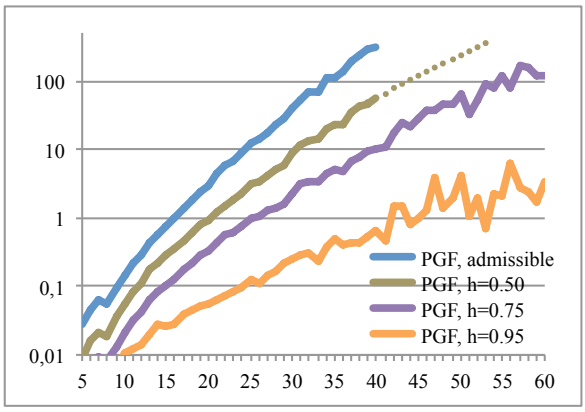

Figure 4: Parsing time (seconds) with different heuristics factors.

8.3 Comparing different heuristics

In another test we compared the effect of the heuristic factor h described in Section 6. We used the same training and testing data as before, and we tried four different heuristic factors: $h = 0$, 0.50, 0.75 and 0.95. As mentioned in Section 6, a factor of 0 gives an admissible heuristics, which means that the parser is guaranteed to return the tree with the best weight.

The parsing times are shown in Figure 4. As can be seen, a higher heuristics factor h gives a considerable speed-up. For 40 token sentences, $h = 0.50$ gives an average speedup of 5 times, while $h = 0.75$ is 30 times faster, and $h = 0.95$ is almost 500 times faster than using the admissible heuristics $h = 0$. This is more clearly seen in Figure 5, where the parsing times are shown relative to the admissible heuristics.

Note that all charts have a logarithmic y-axis, which means that a straight line is equivalent to exponential growth. If we examine the graph lines

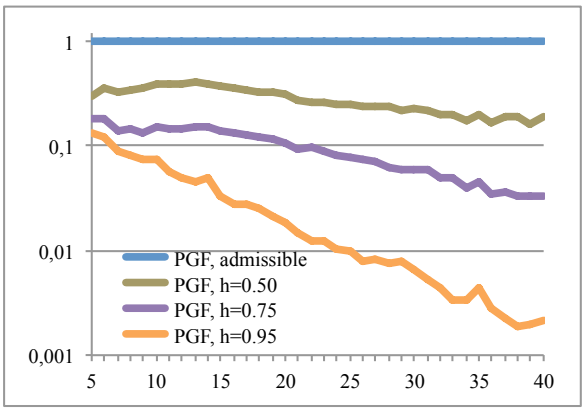

Figure 5: Relative parsing time for different values of h, compared to admissible heuristic.

more closely, we can see that they are not straight. The closest curves are in fact polynomial, with a degree of 4–6 depending on the parser and the value of h.[3]

8.4 Non-admissibility and parsing quality

What about the loss of parsing quality when we use a non-admissible heuristics? Firstly, as mentioned in Section 6, the parser still recognizes exactly the same language as defined by the grammar. The difference is that it is not guaranteed to return the tree with the best weight.

In our evaluation we saw that for a factor $h = 0.50$, 80% of the trees are optimal, and only 3% of the trees have a weight more than 5% from the optimal weight. The performance gradually gets worse for higher h, and with $h = 0.95$ almost 10% of the trees have a weight more than 20% from the optimum.

These numbers only show how the parsing quality degrades relative to the grammar. But since the grammar is trained from a treebank it is more interesting to evaluate how the parsing quality on the treebank sentences is affected when we use a non-admissible heuristics. Table 2 shows how the labelled precision and recall are changed with different values for h. The evaluation was done using the EVALB measure which is implemented in Rparse (Maier, 2010). As can be seen, a factor of $h = 0.50$ only results in a f-score loss of 3 points, which is arguably not very much. On the other extreme, for $h = 0.95$ the f-score drops 14 points.

[3]The exception is the standard Rparse parser, which has a polynomial degree of 8.

	Precision	Recall	F-score
admissible	71.1	67.7	69.3
$h = 0.50$	68.0	64.9	66.4
$h = 0.75$	63.0	60.8	61.9
$h = 0.95$	55.1	55.6	55.3

Table 2: Parsing quality for different values of h.

9 Discussion

The presented algorithm is an important generalization of the classical algorithms of Earley (1970) and Stolcke (1995) for parsing with probabilistic context-free grammars to the more general formalism of parallel multiple context-free grammars. The algorithm has been implemented as part of the runtime for the Grammatical Framework (Ranta, 2011), but it is not limited to GF alone.

9.1 Performance

To show the universality of the algorithm, we evaluated it on large LCFRS grammars trained from the Tiger Treebank.

Our parser is around 10–15 times faster than the latest, optimized version of the Rparse state-of-the-art parser. This improvement seems to be constant, which means that it can be a consequence of low-level optimizations. More important is that our algorithm does not impose any restrictions at all on the underlying PMCFG grammar. Rparse on the other hand requires that the grammar is both binarized and has a fan-out of at most 2.

By using a non-admissible heuristics, the speed improves by orders of magnitude, at the expense of parsing quality. This makes it possible to parse long sentences (more than 50 tokens) in just around a second on a standard desktop computer.

9.2 Future work

We would like to extend the algorithm to be able to use lexicalized statistical models (Collins, 2003). Furthermore, it would be interesting to develop better heuristics for A^* search, and to investigate how to incorporate beam search pruning into the algorithm.

References

Krasimir Angelov. 2009. Incremental parsing with parallel multiple context-free grammars. In *Proceedings of EACL 2009, the 12th Conference of the European Chapter of the Association for Computational Linguistics*, Athens, Greece.

Krasimir Angelov. 2011. *The Mechanics of the Grammatical Framework*. Ph.D. thesis, Chalmers University of Technology, Gothenburg, Sweden.

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. The TIGER treebank. In *Proceedings of TLT 2002, the 1st Workshop on Treebanks and Linguistic Theories*, Sozopol, Bulgaria.

Håkan Burden and Peter Ljunglöf. 2005. Parsing linear context-free rewriting systems. In *Proceedings of IWPT 2005, the 9th International Workshop on Parsing Technologies*, Vancouver, Canada.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4):589–637.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

Peter Hart, Nils Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of Systems Science and Cybernetics*, 4(2):100–107.

Laura Kallmeyer and Wolfgang Maier. 2009. An incremental Earley parser for simple range concatenation grammar. In *Proceedings of IWPT 2009, the 11th International Conference on Parsing Technologies*, Paris, France.

Laura Kallmeyer and Wolfgang Maier. 2013. Data-driven parsing using probabilistic linear context-free rewriting systems. *Computational Linguistics*, 39(1):87–119.

Laura Kallmeyer. 2010. *Parsing Beyond Context-Free Grammars*. Springer.

Makoto Kanazawa. 2008. A prefix-correct Earley recognizer for multiple context-free grammars. In *Proceedings of TAG+9, the 9th International Workshop on Tree Adjoining Grammar and Related Formalisms*, Tübingen, Germany.

Yuki Kato, Hiroyuki Seki, and Tadao Kasami. 2006. Stochastic multiple context-free grammar for RNA pseudoknot modeling. In *Proceedings of TAGRF 2006, the 8th International Workshop on Tree Adjoining Grammar and Related Formalisms*, Sydney, Australia.

Dan Klein and Christopher D. Manning. 2003. A^* parsing: fast exact Viterbi parse selection. In *Proceedings of HLT-NAACL 2003, the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, Edmonton, Canada.

Peter Ljunglöf. 2012. Practical parsing of parallel multiple context-free grammars. In *Proceedings of TAG+11, the 11th International Workshop on Tree Adjoining Grammar and Related Formalisms*, Paris, France.

Wolfgang Maier, Miriam Kaeshammer, and Laura Kallmeyer. 2012. PLCFRS parsing revisited: Restricting the fan-out to two. In *Proceedings of TAG+11, the 11th International Workshop on Tree Adjoining Grammar and Related Formalisms*, Paris, France.

Wolfgang Maier. 2010. Direct parsing of discontinuous constituents in German. In *Proceedings of SPRML 2010, the 1st Workshop on Statistical Parsing of Morphologically-Rich Languages*, Los Angeles, California.

Wolfgang Maier. 2013. LCFRS binarization and debinarization for directional parsing. In *Proceedings of IWPT 2013, the 13th International Conference on Parsing Technologies*, Nara, Japan.

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

Roger Mitton. 1986. A partial dictionary of English in computer-usable form. *Literary & Linguistic Computing*, 1(4):214–215.

Mark-Jan Nederhof. 2003. Weighted deductive parsing and Knuth's algorithm. *Computational Linguistics*, 29(1):135–143.

Aarne Ranta. 2009. The GF resource grammar library. *Linguistic Issues in Language Technology*, 2(2).

Aarne Ranta. 2011. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford.

Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229.

Andreas Stolcke. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201.