Probabilistic Robust Parsing with Parallel Multiple Context-Free Grammars

ABSTRACT
We present an algorithm for incremental statistical parsing with Parallel Multiple Context-Free Grammars (PMCFG). This is an extension of the algorithm in Angelov (2009) to which we added statistical ranking and robustness. The new algorithm preserves the empirically linear complexity of the parser, and it continues to support erasing and reduplication in the grammar. The extension is important since it make it possible to use the algorithm for parsing with large and ambiguous grammars.

KEYWORDS: Parallel Multiple Context-Free Grammars, Statistical Parsing, Robustness.

1 Introduction

There is always an interest for developing efficient algorithms for expressive grammatical formalisms. The expressivity helps to better formalize a linguistic theory, but, even the most sophisticated theory at the end needs to backup with some statistical model. The model is used for disambiguation on one hand, and on the other hand for guiding the parser in the most prominent direction. In this paper we present an extension of the incremental parsing algorithm for Parallel Multiple Context-Free Grammars (PMCFG) (Seki et al., 1991) which was first introduced in Angelov (2009) and later developed in Angelov (2011). The algorithm naturally supports erasing and reduplication and thus it supports the full power of PMCFG which goes beyond a linear context-free rewriting system. We extend the basic algorithm with a statistical model which lets the parser to explore only parts of the search space, when only the most probable parse tree is needed. In addition, we make the parser robust by introducing a new kind of chart items.

Our work is related to Kato et al. (2006) and Kallmeyer and Maier (2010), but since we start from an algorithm for parsing with PMCFG, we are not limited to linear context-free grammars. We do not even require that the grammar is binarized as they do. Furthermore, our cost estimation works differently since we use top-down instead of bottom-up algorithm. The estimation is actually much closer 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 that we use is both admissible and monotonic (Klein and Manning, 2003) which guarantees that we always find the tree whose probability is the global maximum.

We start with a formal definition of a weighted PMCFG in Section 2, and we continue with a weighted deduction system in Section 3. In Section 4, we prove that our estimations are admissible and monotonic. The algorithm requires an estimate for the minimal inside probability for every category, and we show the computation in Section 5. Finally in Section 6, we demonstrate how to make the algorithm robust in the presence of unknown words and unknown syntactic constructions.

2 PMCFG definition

We use the same definition of PMCFG as is used in Angelov (2009) except that we extend it with probabilities for the productions:

Definition 1 *A parallel multiple context-free grammar is an 8-tuple* $G = (N, T, F, P, S, d, r, a)$ *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 and $f \in F$ is the function symbol. 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. The weight of the production is $w > 0$.
- *S is the start category and $d(S) = 1$.*

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

$$w = -\log P(A \to f[\vec{B}] \mid A) \tag{1}$$

where $P(A \to f[\vec{B}] \mid A)$ is the probability to choose this production when the result category is fixed. In general the sum of the probabilities for all productions with the same result category is less than or equal to one, i.e. for category A:

$$\sum_{A \xrightarrow{w} f[\vec{B}] \in P} e^{-w} \leq 1 \tag{2}$$

We will use the inequality in Section 6 to make the parser robust.

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 combinations of prepositions *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 parse tree in PMCFG is a tree of function applications. For instance, the phrase *"both red and either black or white"* is represented by the tree:

$$(\textit{ConjA}\ \textit{both\_and}\ \textit{red}\ (\textit{ConjA}\ \textit{either\_or}\ \textit{black}\ \textit{white}))$$

The weight of a tree is the sum of the weights for all functions used in the tree. 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 parser always finds the tree which minimizes the weight.

3 Deduction System

A key feature in Angelov's algorithm is that the powerful PMCFG formalism is reduced to a simple context-free grammar which is extended dynamically at parsing time in order to account for non context-free features in the original grammar. This can be exemplified with the grammar on Figure 1. There are two productions for category *Conj*, but in the phrase *"both black and white"*, after accepting the token *both*, only the production $\textit{Conj} \xrightarrow{w_5} \textit{both\_and}[]$ can be applied for parsing the second part of the conjunction. This is achieved by generating a new category $\textit{Conj}_2$ which has just a single production:

$$\textit{Conj}_2 \xrightarrow{w_5} \textit{both\_and}[] \tag{3}$$

$$
\begin{aligned}
A &\xrightarrow{w_1} \mathit{ConjA}[\mathit{Conj}, A, A] \\
A &\xrightarrow{w_2} \mathit{black}[] \\
A &\xrightarrow{w_3} \mathit{white}[] \\
A &\xrightarrow{w_4} \mathit{red}[] \\
\mathit{Conj} &\xrightarrow{w_5} \mathit{both\_and}[] \\
\mathit{Conj} &\xrightarrow{w_6} \mathit{either\_or}[]
\end{aligned}
$$

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

Figure 1: Example Grammar

The parsing algorithm is basically an extension of the Earley (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 $\langle k; l\rangle$ 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:

$$[{}_j^k A; l; N]$$

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 N. The set of productions derived for N compactly records all possible ways to parse the $j - k$ fragment.

INITIAL PREDICT

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

PREDICT

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

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

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

COMBINE

$$\frac{[{}^{u}_{j}A \xrightarrow{w} f[\vec{B}]; l : \alpha \bullet \langle d; r\rangle\ \beta; w_i; w_o] \quad [{}^{k}_{u}B_d; r; N]}{[{}^{k}_{j}A \xrightarrow{w} f[\vec{B}\{d := N\}]; l : \alpha\ \langle d; r\rangle \bullet \beta; w_i + w_N - w_{B_d}; w_o]}$$

Figure 2: Deduction Rules

The inside w_i and the outside w_o weights in the active items deserve more attention since this is the only difference compared to Angelov (2009). When the item is complete, it will yield the forest of all trees that derive the substring covered by the item. For the example when the first constituent for category *Conj* from the example in the beginning of the section is completely parsed, the forest will contain the single production in (3). 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 does 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 notion 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 (Section 5). When the category is created by the parser then it will also compute the weight.

The deduction rules are shown on Figure 2. This is a weighted deduction system and it works with the assumption 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: $\text{rhs}(g, r)$ denotes constituent with index r in function g; and ω_k denotes the k-th token in the sentence.

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 is moved to the end of the constituent. Here we generate a new category N which is unique for the combination (A, l, j, k), and we derive the production $N \xrightarrow{w} f[\vec{B}]$ for it. We set the weight w_N for N 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 N.

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 N. The new weight is $w_i + w_N - w_{B_d}$, 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 the same function g on the top level, then the best weight for such 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:

$$w_N \geq w_{B_d}$$

must hold because the forest of trees for N is included in the forest for B_d. From this we conclude the monotonicity condition:

$$(w_i + w_N - w_{B_d}) + w_o \geq w_i + w_o$$

The last two inequalities are valid only if we can correctly compute w_N for a dynamically generated category N. 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 N 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 N we actually do it from an active item with minimal inside weight w_i. This means that it is safe to assign that $w_N = w_i$.

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 which 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 statistical parser which performs Dijkstra search in the space of all parse trees. Any other reasonable weight assignment will give us an A^* algorithm.

In general it is possible to devise different heuristics which will give us different improvements in the parsing complexity. Right now we use a very simple 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 just 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 Robustness

There is always a tension between using more expressive formalisms that can capture richer linguistic structures and simpler formalisms which are not that powerful but are generally more robust. In order to compensate for this tension we built in robustness in our model. The approach is similar to the introduction of wildcard states as it is described in Stolcke (1995), but we made it a bit more general by allowing the wildcards to appear in any position and not only on the top of the tree as in Stolcke. In addition, we had to adapt the technique that was originally for context-free grammar to the richer PMCFG formalism.

The idea is that we work under the assumption that the grammar is incomplete in general. For every new sentence we might expect to find syntactic constructions that are not described by the grammar. The parsing rules on Figure 2, however, expect that at any point there is some production and a function definition that the parser must follow in order to move forward. We must relax these restrictions in order to make the parser robust.

The first important question is how we want to represent incomplete trees. In PMCFG the outcome from the parser is a tree composed by function applications. When we have a sentence that is not in the scope of the grammar, then we can interpret it as a sentence that is produced by applying some unknown and undefined function. Our expectation is that this will happen only locally while as a whole the grammar will have good coverage. This is a well known situation in type theory and functional programming where it is often necessary to represent partial programs. In such cases the program is allowed to contain metavariables which serve as placeholders for the parts that are not completed yet. For instance, if we want to represent the phrase *"neigther black nor white"* with the grammar on Figure 1, then we can choose one of the following two representations:

$$(\mathit{ConjA}\ ?\ \mathit{black}\ \mathit{white}) \tag{4}$$

or

$$(?\ \mathit{black}\ \mathit{white}) \tag{5}$$

Here in both cases $?$ is the symbol for a metavariable. The only difference is that while the first case represents the assumption that the pair *neither-nor* is some kind of conjunction (which is correct in this case), the second case is more general and just glues together the parts in the sentence that are known to the grammar. Which of the two representations will be chosen depends on the statistical model for the grammar.

In Section 2, we said that the sum of the probabilities for all productions of a given category is less or equal to one. This directly reflects the fact that we consider the grammar incomplete. There is always some probability that a new unknown production must be used for a given sentence. For that purpose for every category we introduce the weight u_A which is the cost of introducing a metavariable. With the new weight, we can turn the inequality (2) into strict equality:

$$\Big(\sum_{A \xrightarrow{w} f[\vec{B}] \in P} e^{-w}\Big) + e^{-u_A} = 1 \tag{6}$$

Now we can extend the deduction rules on Figure 2 with new rules which will predict and manipulate meta variables. Basically each time when we do prediction for some category, in addition to the known productions for the category, we must also predict a metavariable since it is always possible to encounter an unknown construction. The corner case is when $u_A = \infty$. This means that there is a zero probability of encountering unknown constructions. We use this as a filter which makes some categories closed, i.e. only strict matching is allowed, and leaves other categories open-ended.

The metavariables can be applied to any set of arguments and an active item associated with a metavariable can scan over any token. This guarantees that the parser can skip over any ungrammatical input while still extracting those pieces that are parseable. The down side of this flexibility is that the meta rules will introduce far too many ambiguities. In order to control the over-generation, we must introduce few more weights which will guide the parser to select the best possible tree.

The first observation is that we do not want the the syntactic categories to be nested in an arbitrary way. For example if some phrase is already recognized as a verb, it would be very unusual to predict that there is a verb phrase nested inside the verb. On the contrary, the opposite is very likely. The probability for a particular kind of nesting is captured with the matrix of weights $c_{A,B}$ which gives the cost of nesting B inside A.

Similarly, we use the weight t_A to distinguish between categories that are mostly lexical and categories that are mostly syntactic. Although the PMCFG formalism does not have built in distinction between the two kinds of categories, the linguistic tradition is to assign some category (part of speech tag) to each word in the sentence, and after that to glue those lexical categories into phrases labelled with syntactic categories. The weight t_A tells us how likely it is to assign the category A to some word. For instance we can assign $t_{Conj} = 0$ (which corresponds to probability one) and $t_A > 0$ for the example on Figure 1. This will guide the parser to prefer parses for *"neither black nor white"* where the unknown words are assigned to the category *Conj*. In other words it will prefer the tree (4) instead of tree (5).

The rules for prediction and manipulation of metavariables are given on Figure 3 and are very similar to the one for normal productions. The main difference is that now we use the pseudo production $A \to ?[\vec{B}]$ which plays the role of the wildcard state in Stolcke's algorithm. In addition we use the weights u_A, $c_{A,B}$ and t_A for guiding the parser.

The two prediction rules INITIAL META PREDICT and META PREDICT produce an active item with a metavariable for the start category or for a category from another active item. Note that the antecedent in the META PREDICT rule is still an active item for a known function and not for a metavariable. If we did allow metavariables in the antecedent this will let the parser generate trees where one metavariable is an argument of another. This is unnecessary since such trees can be flattened without losing useful information. At the same time, the restriction helps by reducing the search space for the parser. Otherwise, the main difference between the meta rules and the normal prediction rules is that now the weights u_S and u_A are used as the inside weight for the new items. This means that in order to guarantee that the new rules are monotonic, for every category A, the weight u_A must be bound by the condition $u_A \geq w_A$. In fact we expect that $u_A \gg w_A$ which simply states that the coverage of the grammar must be close to complete. If this is not satisfied, it might happen that the parser will choose to use a metavariable in place where a normal production can be applied.

The META SCAN rule is the same as the corresponding SCAN rule, except than now we increase the inside weight with t_A for each token. In this way the unbound extension of a metavariable is penalized and in general the parser will prefer shorter spans. Without this penalty, it might happen

INITIAL META PREDICT

$$\frac{}{[{}^{0}_{0}S \to ?[]; 1 : \bullet; u_S; 0]} \quad S \text{ - start category}$$

META PREDICT

$$\frac{[{}^{k}_{j}A \xrightarrow{w_2} f[\vec{B}]; l : \alpha \bullet \langle d; r \rangle\ \beta; w_i; w_o]}{[{}^{k}_{k}B_d \to ?[]; r : \bullet; u_{B_d}; w_i - w_{B_d} + w_o]}$$

META SCAN

$$\frac{[{}^{k}_{j}A \to ?[\vec{B}]; l : \alpha \bullet; w_i; w_o]}{[{}^{k+1}_{j}A \to ?[\vec{B}]; l : \alpha\ s \bullet; w_i + t_A; w_o]} \quad s = \omega_{k+1}$$

META COMPLETE

$$\frac{[{}^{k}_{j}A \to ?[\vec{B}]; l : \alpha \bullet; w_i; w_o]}{N \xrightarrow{w_i - w_{\vec{B}}} g[\vec{B}] \qquad [{}^{k}_{j}A; l; N]} \quad N = (A, l, j, k)$$

META COMBINE

$$\frac{[{}^{u}_{j}A \to ?[\vec{B}]; l : \alpha \bullet; w_i; w_o] \quad [{}^{k}_{u}X; r; N]}{[{}^{k}_{j}A \to ?[\vec{B}, N]; l : \alpha\ \langle d; r \rangle \bullet; w_i + w_N + c_{A,X}; w_o]}$$

Figure 3: Meta Rules

that the parser will choose to replace a large complete subtree with a metavariable, when the weight of the subtree has grown higher than the weight u_A. The weight t_A compensates by increasing the weight for the metavariable as well.

The rules META COMBINE and COMBINE are related in a similar fashion. This time we add another argument (i.e. a subtree) to a metavariable, but again we penalize the unbound expansion with the weight $c_{A,X}$.

Note that both in META SCAN and in META COMBINE we add the new elements before instead of after the dot in the item. In this way it is possible at any time to complete the item with the META COMPLETE rule.

7 Training

The training is fairly trivial in the absence of metavariables. We need a treebank with PMCFG trees from which we can estimate the probabilities $P(A \to f[\vec{B}] \mid A)$ by using simple counting and smoothing. The weights for the productions are then assigned with Equation (1).

The training with metavariables is just slightly different. If we have a grammar with partial coverage, then some of the nodes in the treebank will have to be marked with metavariables. In this case for each category, we will have both the probability of having a metavariable and also one probability for each production. The probability for a metavariable is of course the source of the weight u_A.

The weight $c_{A,B}$ is computed by counting how often in the treebank a tree of category B appears as an argument of a function or a metavariable whose result category is A. The negative logarithm of the probability is the weight $c_{A,B}$.

Similarly t_A is computed by counting how often a token is yielded directly from the category A. Intuitively this gives a low weight for categories like noun and verb and a much higher weight for the syntactic categories.

8 Evaluation

The parser has been implemented and tested as part of the runtime system for the Grammatical Framework (GF) programming language (Ranta, 2004). Although the primary application of the runtime system is currently limited to running GF applications, in principle it is not language specific, 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 large coverage grammars (Ranta, 2009) for over 20 languages, which are used as a resource for deriving small domain specific grammars. The first application for our probabilistic PMCFG parser was to apply these resource grammars directly for parsing. This is an unusual application since the grammars are highly ambiguous and they lack the domain knowledge that the domain specific grammars can use for disambiguation. We compensated the lack of explicit knowledge with a statistical model. Although this is not a new approach in the data-driven language processing, it is a turning point in the otherwise knowledge driven development in GF.

We used the resource grammar for English in combination with a large lexicon of 40000 lemmas which is an adaptation to GF of the Oxford Advanced Learner's Dictionary (Mitton, 1986). In total the grammar has 44000 productions where we count both the syntactic and the lexical productions. For training we used the Penn Treebank (Marcus et al., 1993) which we converted to trees compatible with the resource grammar. When the grammar does not have a complete coverage for some sentence, then we inserted some metavariables. The training procedure itself was the same as the one in Section 7.

We are interested in the evaluation of two main aspects. First of all the output of the parser must be reasonable, i.e. the statistical model should give higher weights to trees that are closer to the gold standard. Second the statistical ranking should help in improving the algorithmic complexity by guiding the parser in the most prominent search direction.

The evaluation of the parsing precision shows results similar to earlier studies on probabilistic context-free grammars. In general the parser makes good predictions for the part of speech tags[1] and for the local dependencies between the words. The main source of errors are sentences with an ambiguity in a propositional phrase attachment. This suggests that further research should be invested into incorporating lexicalized statistical models (Collins, 2003) in PMCFG. Another source of errors is when the sentence is not fully parseable with the grammar. In this case the parser can use the meta rules from Section 6 but the tree with the best score is not always the right one. The reason is that the parser tends to use existing grammar productions as much as possible which can result in a combination of a metavariable and some low probability productions. Although the probability for the productions is low, the overall probability of the tree happens to be higher than the probability of using only a metavariable. This choice makes sense in some cases but is completely wrong in other cases. Again, it will help if the parser can use a statistical model conditionalized on the lexical heads.

The parsing complexity was our main concern since parsing with the resource grammars is quite expensive. Samples from the evaluation are shown on Figures 4 and 5. The dots on Figure 4 show the relation between the sentence length and the parsing time in seconds. Each dot is a single sample from the test corpus. It can be seen that the relation is close to linear although there might be some small factors of higher order. Still the time for longer sentences (> 20 tokens) is quite high. In Figure 5 the parsing time is averaged over different sentences of the same length. This time we

[1] We do not pre-process the sentence with a part of speech tagger and the tags are computed in the parser.

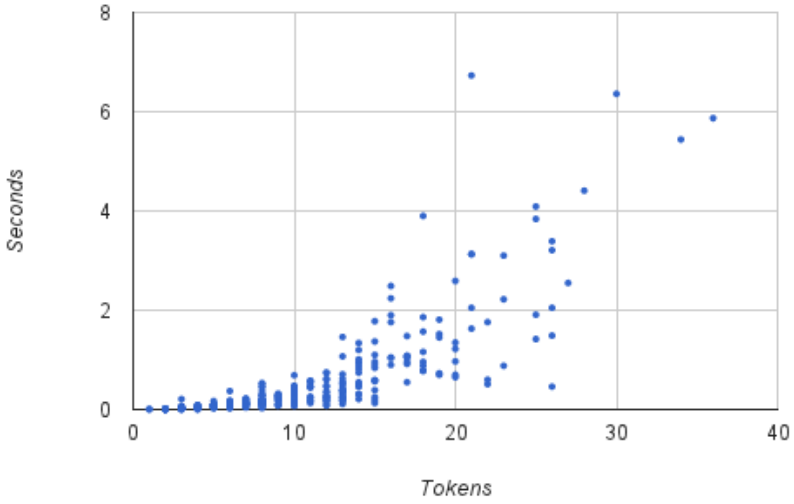

Figure 4: Parsing Time in Seconds per Token

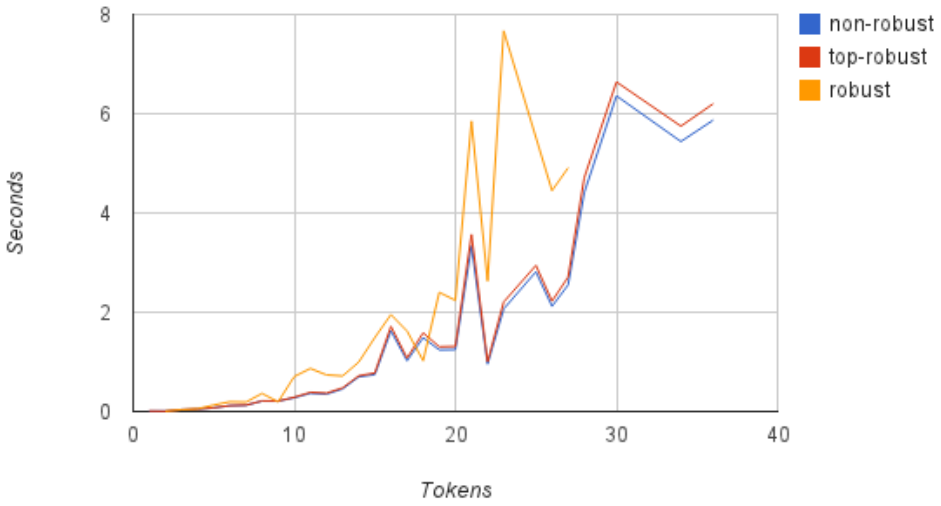

Figure 5: Average Parsing Time in Seconds per Token

showed three different curves. The lowest one (`non-robust`) shows the complexity when only the normal parsing rules are used, and the robustness is switched off. Slightly above it is the curve with partial robustness where metavariables are allowed only on the top-level (`top-robust`). This is essentially the same kind of robustness that Stolcke (1995) used, and it corresponds to parsing only chunks in the sentence. Far above these two curves is the curve where full robustness is allowed (`robust`). The difference is quite significant and in fact we had to remove some sentences during the evaluation of the full robustness since we did not have enough memory. Fortunately there is an easy explanation. The constant factor is a function of the size of the grammar and the introduction of robustness is essentially equivalent to extending the grammar. The META ROBUST rule can predict any category from any other category, and the only factor which reduces the number of combinations is the associated weight for the item. Combinations with high weight can be delayed indefinitely or never explored if a complete tree with lower weight is found. Fortunately it must be possible to eliminate most of this combinations, if we had bottom up filtering in the prediction. Currently neither the PREDICT nor the META PREDICT rule takes into account the value of the current token[2]. The filtering will benefit the normal prediction too, but it will be even more important for the meta prediction.

Although the time for parsing long sentences might sound pessimistic, it is actually a major advancement compared with the non-statistical version of the parser in Angelov (2009). Previously the algorithm was used only with small application grammars with mostly unambiguous lexicons. In this scenario it performed quite well, but now we are faced with another level of complexity. We found that the new statistical parser can find the best tree by processing only about 10% of the items that will have to be processed if the non-statistical parser was used.

9 Conclusion

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 was implemented and evaluated as part of the runtime for the Grammatical Framework (Ranta, 2004), but the essence of the work is theoretical and it could be reused in other platforms based on related context-sensitive formalisms. The main future directions that we identified are an extension to the algorithm which will let us to use lexicalized statistical models (Collins, 2003), and an implementation for bottom up filtering with PMCFG. Furthermore, it would be interesting to develop better heuristics for A^* search.

[2] Actually we use very simple bottom up filtering for the prediction with pre-terminal categories but this does not help with the phrase level categories.

References

Angelov, K. (2009). Incremental parsing with parallel multiple context-free grammars. In *European Chapter of the Association for Computational Linguistics*.

Angelov, K. (2011). *The Mechanics of the Grammatical Framework*. PhD thesis, Chalmers University of Technology.

Collins, M. (2003). Head-driven statistical models for natural language parsing. *Comput. Linguist.*, 29(4):589–637.

Earley, J. (1970). An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94–102.

Kallmeyer, L. and Maier, W. (2010). Data-driven parsing with probabilistic linear context-free rewriting systems. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 537–545, Stroudsburg, PA, USA. Association for Computational Linguistics.

Kato, Y., Seki, H., and Kasami, T. (2006). Stochastic multiple context-free grammar for RNA pseudoknot modeling. In *Proceedings of the Eighth International Workshop on Tree Adjoining Grammar and Related Formalisms*, TAGRF '06, pages 57–64, Stroudsburg, PA, USA. Association for Computational Linguistics.

Klein, D. and Manning, C. D. (2003). A^* parsing: fast exact Viterbi parse selection. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 40–47, Stroudsburg, PA, USA. Association for Computational Linguistics.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19:313–330.

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

Ranta, A. (2004). Grammatical Framework: A Type-Theoretical Grammar Formalism. *Journal of Functional Programming*, 14(2):145–189.

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

Seki, H., Matsumura, T., Fujii, M., and Kasami, T. (1991). On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229.

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