

Language Modeling with Parallel Multiple Context-Free Grammars

Krasimir Angelov

Chalmers University of Technology

Göteborg, Sweden

krasimir@chalmers.se

Abstract

This paper outlines and solves some theoretical issues related to the probabilistic language modeling with parallel multiple context-free grammars.

1 Introduction

Parallel Multiple Context-Free Grammar (PMCFG) (Seki et al., 1991) is one of the grammar formalisms that have been proposed for the syntax of natural languages. It is an extension of context-free grammar (CFG) where the right hand side of the production rule is a tuple of strings instead of only one string. Using tuples the grammar can model discontinuous constituents which make it more powerful than context-free grammar. In the same time PMCFG has the advantage to be parseable in polynomial time which makes it attractive from computational point of view.

In Angelov (2009) an efficient parsing algorithm for PMCFG is described which has the advantage to be incremental. It reads the input token by token and after each token only those partial parse trees that are consistent with the accepted input are kept in the parse chart.

The incremental nature of the algorithm is a unique feature which makes it possible to use PMCFG for language modeling for a first time. Traditionally language modeling is predominantly based on hidden Markov model but works based on probabilistic context-free grammars (PCFG) are also known (Stolcke, 1995). Another related work is Kato et al. (2006) which describes parsing algorithm for stochastic MCFG¹ but the algorithm is not incremental and cannot be used for language modeling.

The problem that is addressed in this paper is how to incorporate probabilistic model in the incremental PMCFG parser. Both the algorithms in Angelov (2009) and Stolcke (1995) are extensions of the classical algorithm in Earley (1970) for context-free grammars. In the first work the Earley's algorithm is extended to work with context-sensitive grammar and in the second work the same algorithm is extended with probabilistic model. The question is how to combine them and the answer turned out to be nontrivial.

¹MCFG is closely related formalism which is only mildly context-sensitive and so is weaker than PMCFG

In the next two sections we briefly describe the incremental PMCFG parsing first and the language modeling with context-free grammars next. In section 4 we describe how the two algorithms could be combined.

2 Incremental Parsing with PMCFG

The advantage of the PMCFG formalism could be illustrated with this example grammar:

$VT \rightarrow \text{turn-off}[]$; 0.3
$VT \rightarrow \text{turn-down}[]$; 0.2
$VT \rightarrow \text{swith-off}[]$; 0.5
$VP \rightarrow \text{ComplSep}[VT, NP]$; 0.1
$VP \rightarrow \text{ComplCont}[VT, NP]$; 0.9
$\text{turn-off} := ("turn", "off")$	
$\text{turn-down} := ("turn", "down")$	
$\text{swith-off} := ("swith", "off")$	
$\text{ComplSep} := \langle (1; 1) \langle 2; 1 \rangle \langle 1; 2 \rangle \rangle$	
$\text{ComplCont} := \langle (1; 1) \langle 1; 2 \rangle \langle 2; 1 \rangle \rangle$	

The grammar has list of productions of the form:

$$X \rightarrow f[X_1, \dots, X_n]$$

where X, X_1, \dots, X_n are nonterminals and f is a function symbol. Every nonterminal X_i could derive a tuple of strings and the already derived tuples are combined with the function f in the production to produce a new tuple for the nonterminal X .

The example grammar has three phrasal transitive verbs: *turn off*, *turn down* and *swith off*. Each verb is represented by a function with corresponding name and a production rule which applies the function to empty list of arguments to derive a tuple for the category VT . Since these phrasal verbs could be discontinuous they are represented with tuples containing two strings. One for the verb itself and one for the parts *off* and *down*. The functions *ComplSep* and *ComplCont* are applied to nonterminals VT and NP and produce VP . The difference is that *ComplSep* splits the two parts of the verb and inserts the noun phrase of the subject between them, while *ComplCont* places the two parts together and adds the subject at the end. The notation $\langle d; i \rangle$ in the function definitions means take constituent number i from the tuple of argument with number d . If there were rules that produce the phrase *the lights* then using the first function the phrase *turn the lights off* could

be produced while the second will produce *turn off the lights*.

The same grammar could be approximated with the following PCFG:

$VT^1 \rightarrow \text{turn}$;0.5
$VT^1 \rightarrow \text{swith}$;0.5
$VT^2 \rightarrow \text{off}$;0.8
$VT^2 \rightarrow \text{down}$;0.2
$VP \rightarrow VT^1 NP VT^2$;0.1
$VP \rightarrow VT^1 VT^2 NP$;0.9

It is approximation because the PCFG also allows the phrase *switch down* which was not allowed in the original grammar.

The incremental PMCFG parser is a variant of the Earley parser for CFG but enforces the additional restrictions by adding new nonterminals and productions during the parsing. If the phrase *swith the TV off* have to be parsed then after the token *swith* it will create a new nonterminal VT' and one production for it:

$$VT' \rightarrow \text{swith-off}[]$$

This is the same production as in the original grammar but now the new nonterminal does not have the rules for *turn off* and *turn down*. Later when the parser tries to recognize the second part of the verb i.e. *off* it will use the new restricted nonterminal which does not have *down* as a possible second part.

The details about how exactly the algorithm works are described in Angelov (2009).

The differences in the formalisms also affect the probability distribution in the language. The probabilities of the PMCFG productions are cited after each production. The corresponding probabilities for the CFG productions are the same except for the rules for *turn* and *off*. They are obtained by summing over all productions in PMCFG that contain the corresponding constituent. The probability contribution of the verb *swith off* in the phrase *swith the TV off* is 0.5 according to the PMCFG model because this is application of only one production. In the CFG model it is $0.5 \cdot 0.8 = 0.4$ because it requires the application of two distinct rules. In the same time the impossible verb *switch down* is as probable as *turn down* in CFG while it is prohibited in PMCFG.

3 Language Modeling with CFG

In Stolcke (1995) a probabilistic variant of the Earley algorithm is described which could be used for language modeling. The algorithm could compute the probability of each prefix of string in a context-free language. This allows to compute the probability distribution among the set of next possible words given the

already know prefix:

$$P(w_i | w_0 \dots w_{i-1}) = \frac{P(w_0 \dots w_i)}{P(w_0 \dots w_{i-1})}$$

At the heart of the algorithm are two matrices – the left-corner matrix and the unit-production matrix.

A nonterminal Y is a left corner of another nonterminal X if for some β there is a production:

$$X \rightarrow Y\beta$$

The relation is abbreviated as:

$$X \Rightarrow_L Y$$

The transitive closure of the relation is defined as:

$$X \overset{*}{\Rightarrow}_L Y \equiv (X \Rightarrow_L Y) \vee [(X \Rightarrow_L Z) \wedge (Z \overset{*}{\Rightarrow}_L Y)]$$

The left-corner matrix $R(X \overset{*}{\Rightarrow}_L Y)$ is indexed by the nonterminals X and Y and contains the probability that X is transitive left-corner of Y . The matrix is precomputed once for each grammar using the nontransitive left-corner matrix $P(X \Rightarrow_L Y)$. Stolcke (1995) has proved that:

$$R(X \overset{*}{\Rightarrow}_L Y) = (I - P(X \Rightarrow_L Y))^{-1}$$

The unit-production matrix $R(X \overset{*}{\Rightarrow} Y)$ is computed in similar way but in the computation only the unit productions are taken into account.

Following Shieber et al. (1995) the Earley algorithm could be defined as deductive system operating with items of the form:

$$[{}_j^k X \rightarrow \lambda \bullet \mu]$$

The item denotes that the input range from position j to k was partially parsed with production $X \rightarrow \lambda\mu$ where λ is the part of the production which is already recognized and μ is the part which is not recognized yet.

In the probabilistic version of the algorithm every item is annotated with two probabilities:

forward probability α - The total probability of the partial parse trees to which the item belongs.

inner probability γ - The total probability of the subtrees rooted by the item.

The items with probabilities will be written as:

$$[{}_j^k X \rightarrow \lambda \bullet \mu \mid \alpha\gamma]$$

The algorithm has three derivation rules. The first rule is PREDICT:

$$\begin{array}{l} \text{PREDICT} \\ [{}_j^k X \rightarrow \lambda \bullet Z \mu \mid \alpha\gamma] \\ [{}_k^k Y \rightarrow \bullet \nu \mid \alpha'\gamma'] \end{array}$$

where the new probabilities are computed as:

$$\begin{aligned}\alpha' &+= \alpha \cdot R(Z \xrightarrow{*}_L Y) P(Y \rightarrow \nu) \\ \gamma' &= P(Y \rightarrow \nu)\end{aligned}$$

and $P(Y \rightarrow \nu)$ is the probability of the production used in the prediction. The operator $+=$ means that if there is more than one way to derive the same item then the corresponding probabilities are accumulated. The rule could be applied only if the nonterminal Z and Y are such that $R(Z \xrightarrow{*}_L Y) \neq 0$.

The SCAN rule just moves the dot if it is before a terminal which matches the current terminal in the input:

$$\text{SCAN} \quad \frac{[{}_j^k X \rightarrow \lambda \bullet s \mu \mid \alpha \gamma]}{[{}_j^{k+1} X \rightarrow \lambda s \bullet \mu \mid \alpha \gamma]} \quad s = w_{k+1}$$

The last rule COMBINE combines one complete item with another where the dot is before the same nonterminal.

$$\text{COMBINE} \quad \frac{[{}_j^u X \rightarrow \lambda \bullet Z \mu \mid \alpha \gamma] \quad [{}_u^k Y \rightarrow \nu \bullet \mid \alpha'' \gamma'']}{[{}_j^k X \rightarrow \lambda Z \bullet \mu \mid \alpha' \gamma']}$$

the probabilities in the derived item are computed as:

$$\begin{aligned}\alpha' &+= \alpha \cdot \gamma'' R(Z \xrightarrow{*} Y) \\ \gamma' &+= \gamma \cdot \gamma'' R(Z \xrightarrow{*} Y)\end{aligned}$$

Again the rule could be applied only if the probability $R(Z \xrightarrow{*} Y) \neq 0$.

4 Language Modeling with PMCFG

The parsing algorithm in Angelov (2009) is similar to the Earley's algorithm and the intuition is that Stolke's algorithm could be applied to PMCFG as well. It is not a problem in theory but in practice the naive implementation will be far too slow. The problem is that Stolke's algorithm presupposes that the matrices $R(Z \xrightarrow{*}_L Y)$ and $R(Z \xrightarrow{*} Y)$ are precomputed in advance. This is possible with CFG but not with PMCFG because the later is represented as dynamic CFG which is updated at runtime. It is possible to recompute the matrices after each update but this is far too expensive. The computation involves global analyses of the grammar and also matrix inversion.

Fortunately there is a work around. As Stolke noted the matrices are necessary only if the grammar uses left recursion. Otherwise this simplified versions of PREDICT and COMBINE could be used:

$$\text{PREDICT} \quad \frac{[{}_j^k X \rightarrow \lambda \bullet Y \mu \mid \alpha \gamma]}{[{}_k^k Y \rightarrow \bullet \nu \mid \alpha' \gamma']} \left\{ \begin{array}{l} \alpha' += \alpha \cdot P(Y \rightarrow \nu) \\ \gamma' = P(Y \rightarrow \nu) \end{array} \right.$$

$$\text{COMBINE} \quad \frac{[{}_j^u X \rightarrow \lambda \bullet Y \mu \mid \alpha \gamma] \quad [{}_u^k Y \rightarrow \nu \bullet \mid \alpha'' \gamma'']}{[{}_j^k X \rightarrow \lambda Y \bullet \mu \mid \alpha' \gamma']} \left\{ \begin{array}{l} \alpha' += \alpha \text{ other elements of the loop.} \\ \gamma' += \gamma \text{ and the last element of the sequence from this follows} \end{array} \right.$$

The roles of the matrices are just to account for the infinitely many ways to arrive from nonterminal Z to nonterminal Y , via the left-corner relation, when there are left-recursion loops. These loops could be determined statically for the initial grammar and we will show that the new productions that are added at runtime could produce new loops only in a special cases which could also be determined statically.

Before to define the left-corner relation in PMCFG it is helpful to introduce the procedure for deriving the context-free approximation of a PMCFG. If X is a PMCFG nonterminal with n constituents then in the CFG approximation it is split into n nonterminals: $X^1 \dots X^n$. The productions for these nonterminals are derived as follows - if there is a production and a function:

$$\begin{aligned}X &\rightarrow f[Y_1 \dots Y_m] \\ f &:= (\lambda_1, \dots \lambda_n)\end{aligned}$$

then

$$X^i \rightarrow \bar{\lambda}_i$$

where $\bar{\lambda}_i$ is derived from λ_i by replacing each $\langle d; r \rangle$ with the context-free nonterminal Y_d^r .

The left-corner relation for PMCFG is actually defined on the context-free nonterminals X^i and Y^j in the same way as for ordinary CFG. Since the incremental parser always works with the context-free approximation of the original grammar it could use the $R(X^i \xrightarrow{*}_L Y^j)$ and $R(X^i \xrightarrow{*} Y^j)$ matrices computed from the initial approximation of the grammar.

This is accurate only for nonterminals which exist in the original grammar. For new fresh nonterminals the simplified prediction and completion rules should be used but adjusted for the cases when there are loops. The loops could be detected using the following theorem:

Theorem 1 *If X_1 is a fresh nonterminal generated in the parser and the loop:*

$$X_1^{r_1} \Rightarrow_L X_2^{r_2} \Rightarrow_L \dots X_n^{r_n} \Rightarrow_L X_1^{r_1}$$

was created after the recognition of constituent r_1 , then all productions used in the loop are unit productions.

First note that X_1 is not in the original grammar because it is a fresh nonterminal generated from the parser. Since it is left corner of $X_2 \dots X_n$ they should also be fresh nonterminals. When a new nonterminal is created it is created for each unique tuple (A, l, j, k) where A is a nonterminal such that the range $j - k$ was successfully recognized with constituent l . When the nonterminals are in left-corner relation then their ranges should be in range-subrange relation. In the case for the loop the range for $X_2^{r_2}$ should be the same or a subrange of the range of $X_1^{r_1}$. The same applies for the other elements of the loop. Since X_1 is both the first and the last element of the sequence from this follows

that actually all ranges are equal. If there is a production:

$$X_i^{r_i} \rightarrow \lambda X_j^{r_j} \mu$$

and both $X_i^{r_i}$ and $X_j^{r_j}$ have the same ranges then both λ and μ derive the empty string i.e. this is an unit production².

The unit-production loops are easy to find – they are encoded in the $R(X^i \xRightarrow{*} Y^j)$ matrix. The set of productions that are used in a given unit-production loop is a new grammar which is subgrammar of the original. For every subgrammar a new $R(X^i \xRightarrow{*} Y^j)$ matrix could be computed.

This is illustrated with this example grammar:

$$\begin{aligned} X &\rightarrow f[Y] \\ X &\rightarrow h[Y] \\ Y &\rightarrow g[X] \end{aligned}$$

$$\begin{aligned} f &:= (\langle 1; 1 \rangle, \langle 1; 2 \rangle \lambda) \\ g &:= (\langle 1; 1 \rangle, \langle 1; 2 \rangle \mu) \\ h &:= (\langle 1; 1 \rangle \gamma, \epsilon) \end{aligned}$$

It has one unit-production loop $X^1 \Rightarrow Y^1 \Rightarrow X^1$ which is seen clearly in the CFG approximation of the grammar:

$$\begin{aligned} X^1 &\rightarrow Y^1 & (1) \\ X^1 &\rightarrow Y^1 \gamma & (2) \\ Y^1 &\rightarrow X^1 & (3) \\ X^2 &\rightarrow Y^2 \lambda & (4) \\ X^2 &\rightarrow \epsilon & (5) \\ Y^2 &\rightarrow X^2 \mu & (6) \end{aligned}$$

From this loop a subgrammar is extracted which contains only productions: 1,3,4 and 6. A new left-corner matrix could be computed based only on this subgrammar. Since there are only finite number of loops in the grammar the set of matrices could be computed in advance.

Using these matrices the parser could compute the right probabilities in the PREDICT rule. The COMPLETE rule is not a problem because the unit productions are always preserved so the original unit-production matrix could be reused.

5 Conclusion

This paper only outlines the theoretical issues that have to be solved in order to implement a probabilistic variant of the incremental PMCFG parser. A proper implementation of the algorithm will definitely clear up important details in the theory as well.

²Strictly speaking unit production is a production like $X \rightarrow Y$ but in Stolke's algorithm this is generalized when the $P(X \xRightarrow{*} Y)$ matrix is computed

Other important theoretical questions like the parameters estimation and the retrieval of the Viterbi parse are not discussed.

The parameters estimation is an interesting question and probably there is a variant of the inside-outside algorithm which could be applied to PMCFG as well.

The Viterbi parse is less interesting. It is always possible to find all parse trees for a given sentence and then to retrieve the most likely one from the chart in polynomial time. There are context-free parsing algorithms that keep in the chart only the most likely trees but this is not applicable to PMCFG. The reason is that in the incremental PMCFG parser the parse chart is just an extension of the grammar itself. If we trim down the grammar too early then some possible parse trees may not be found.

An algorithm for language modeling with PMCFG opens the door for experiments in speech recognition based on more sophisticated language models. Previous works in this direction (Bringert, 2007) were using approximations with context-free grammars and finite state automata.

References

- Krasimir Angelov. 2009. Incremental parsing with parallel multiple context-free grammars. In *European Chapter of the Association for Computational Linguistics*.
- Björn Bringert. 2007. Compiling Grammar-based Speech Application Components. Technical Report 40L, Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg.
- Jay Earley. 1970. An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94–102.
- Yuki Kato, Hiroyuki Seki, and Tadao Kasami. 2006. Stochastic multiple Context-Free Grammar for RNA pseudoknot modeling. In *Proceedings of the Eighth International Workshop on Tree Adjoining Grammar and Related Formalisms*, pages 57–64, Sidney, Australia, July. Association for Computational Linguistics.
- Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229, October.
- Stuart M. Shieber, Yves Schabes, and Fernando C. N. Pereira. 1995. Principles and Implementation of Deductive Parsing. *Journal of Logic Programming*, 24(1&2):3–36.
- Andreas Stolcke. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201.