

(titlepage)

Journal

Computer Speech and Language

Title

Language modeling with probabilistic left corner parsing

Authors

Dong Hoon Van Uytsel and Dirk Van Compernelle
Electrical Engineering Department
Katholieke Universiteit Leuven, Belgium

Running title

PLCG-based language model

Keywords

language modeling, stochastic parsing, automatic speech recognition

(address page)

Address of correspondence

Dong Hoon Van Uytsel
K.U.Leuven/ESAT/PSI
Kasteelpark Arenberg 10
B-3001 Leuven (Heverlee)

(abstract page)

Abstract

We present a novel language model, suitable for large-vocabulary continuous speech recognition, based on parsing with a probabilistic left corner grammar (PLCG). The PLCG probabilities are conditioned on local and non-local features of the partial parse tree, and some of these features are lexical. They are not derived from another stochastic grammar, but directly induced from a treebank, a corpus of text sentences, annotated with parse trees. A context-enriched constituent represents all partial parse trees that are equivalent with respect to the probability of the next parse move. For computational efficiency the parsing problem is represented as a traversal through a compact stochastic network of constituents connected by PLCG moves. The efficiency of the algorithm is due to the fact that the network consists of recursively nested, shared subnetworks. The PLCG-based language model results from accumulating the probabilities of all (partial) paths through this network. Next word probabilities can be computed synchronously with the probabilistic left corner parsing algorithm in one single pass from left to right. They are guaranteed to be normalized, even when pruning less likely paths. Finally, it is shown experimentally that the PLCG-based language model is a competitive alternative to other syntax-based language models, both in efficiency and in accuracy.

Language modeling with probabilistic left corner parsing^{*}

Dong Hoon Van Uytsel^{a,*} Dirk Van Compernelle^a

^aESAT, Katholieke Universiteit Leuven
Kasteelpark Arenberg 10, 3001 Leuven (Heverlee), Belgium

Abstract

We present a novel language model, suitable for large-vocabulary continuous speech recognition, based on parsing with a probabilistic left corner grammar (PLCG). The PLCG probabilities are conditioned on local and non-local features of the partial parse tree, and some of these features are lexical. They are not derived from another stochastic grammar, but directly induced from a treebank, a corpus of text sentences, annotated with parse trees. A context-enriched constituent represents all partial parse trees that are equivalent with respect to the probability of the next parse move. For computational efficiency the parsing problem is represented as a traversal through a compact stochastic network of constituents connected by PLCG moves. The efficiency of the algorithm is due to the fact that the network consists of recursively nested, shared subnetworks. The PLCG-based language model results from accumulating the probabilities of all (partial) paths through this network. Next word probabilities can be computed synchronously with the probabilistic left corner parsing algorithm in one single pass from left to right. They are guaranteed to be normalized, even when pruning less likely paths. Finally, it is shown experimentally that the PLCG-based language model is a competitive alternative to other syntax-based language models, both in efficiency and accuracy.

Key words: language modeling, stochastic parsing, automatic speech recognition

^{*} Part of the presented work was supported by the K.U.Leuven Research Fund.

^{*} Corresponding author.

Email address: donghoon@esat.kuleuven.ac.be (Dong Hoon Van Uytsel).

1 Introduction

1.1 Problem definition

The basic aim of the work described in this article is to improve language modeling for statistical large-vocabulary continuous speech recognition (LVCSR) by *syntactically structuring* the language model probabilities. For an introduction to speech recognition and language modeling, we refer to texts such as (Young, 1996; Jelinek, 1997; Huang et al., 2001). A number of syntactically structured LMs exist (Chelba and Jelinek, 1999; Roark, 2001; Bod, 2000). In this paper we propose a more efficient and better performing alternative based on left corner parsing.

A LM is a probability mass function estimating the probability of a given sentence. In many situations, it is desirable to have an explicit left-to-right factorization of the sentence probability into a product of conditional probabilities of each word given the words preceding it. These conditional probabilities are estimated by a *conditional* LM (CLM). A CLM automatically translates to a LM, but the formulation of an equivalent CLM starting from a LM is not always possible nor evident.

The most widely used CLM is the word-based n -gram; it can be combined with other common CLMs such as class-based n -grams, cache and trigger models using statistical model combination techniques; a good and recent survey with results from large scale experiments was published by Goodman (2001).

However, all of these popular CLMs fail to draw generalization power from *syntactically* structured dependencies within a sentence: n -grams always assume the same simple structure, while cache and trigger models do not assume any structure. While it cannot be denied that there are many non-structural and local patterns in a language, we believe that syntax is a very important factor in the sentence generation process. Syntax-based CLMs are therefore expected to become a powerful complement to other conventional language modeling techniques.

In restricted application domains, syntax-based (C)LMs are commonly used since it is possible to write a grammar by hand in these cases. However, syntax-based (C)LMs were for a long time inappropriate for LVCSR, unlike simple word-based trigram models. We see at least four reasons of failure:

- (1) The grammars used were context-free.
- (2) There was no use of fine-grained, particularly lexical, features.
- (3) Grammars were hand-built and had insufficient coverage.
- (4) Enlarging grammars to increase coverage causes overgeneration.

The first two problems result in bad probability estimates. The third problem causes zero probability estimates for perfectly acceptable sentences. The last problem may

be responsible for long parse times and underestimation of the probability of accurate parse trees.

A first encouraging advance was realized when Chelba and Jelinek (1999) (henceforth abbreviated C&J) proposed a CLM based on a stochastic shift-reduce parser. Their model marginally improves on a word trigram on the Wall Street Journal task. Its design and strong emphasis on lexical dependencies are reminiscent of the dependency LM (Stolcke et al., 1997); the model can be considered a generalization of the word-based trigram CLM. Problems 1 and 2 mentioned above are avoided by non-local stochastic conditioning on lexical features: the parser uses probabilistic shift and reduce moves that are conditioned on the two most recent ‘exposed heads’ (an exposed head is the lexical head of a phrase covered by a subtree that is not yet a subtree in another tree). We also believe that using conditional probabilities favors internally consistent parse trees, which mitigates problem 3. Problem 4 is tackled by initializing conditional shift and reduce probabilities directly from a treebank (a hand- or machine-parsed text collection) and applying conventional statistical smoothing techniques.

However, the C&J model is inefficient in both memory and time, which restricts its use to rescoring n-best lists or very thin word lattices. Perhaps more importantly, available computational resources severely limit the amount of training text that can be used to reestimate the model. We see two sources of inefficiency:

1. *Following paths leading to improbable derivations.* C&J’s parser grows isolated subtrees from the bottom up without (stochastic) top-down filtering. In the equivalent derivation tree representation, too many paths are extended that lead to improbable paths in the end; they should have been pruned in an earlier stage.

2. *Searching a tree-shaped derivation space.* The derivation space is tree-structured. Hence identical paths occurring at different places in the tree are treated separately. Efficiency could be gained by a representation that avoids repetition of identical paths. A dynamic programming approach for C&J was presented by Jelinek and Chelba (1999), but this paper did not contain empirical results. The model described in (Jelinek and Chelba, 1999) was later implemented and improved by Van Aelten and Hogenhout (2000).

Roark’s top-down parsing LM (Roark, 2001) does not suffer from inefficiency 1.: parse trees are grown from the top down with look-ahead to the next word. So it can decide early which rules should not be expanded. However, the *rule probabilities themselves* cannot be conditioned on the lookahead-word, since that would violate the chosen chain-rule decomposition of the derivation probability. Aggressive pruning of partial paths by comparison of their probabilities is therefore likely to introduce too many search errors. A disadvantage of Roark’s model w.r.t. C&J’s model is that next-word probabilities are systematically underestimated, while C&J are able to renormalize them.

Roark’s derivation space representation is similar to C&J’s, sharing the same inefficiency. Besides, this representation makes it difficult — if not impossible — to find an accurate EM reestimation algorithm. Chelba however found and used a working *approximative* EM reestimation algorithm (Chelba, 2000).

On the other hand, the tree representation of derivation space gives Roark’s model flexibility in extracting conditioning information from the path between the root and the current node using ‘tree-walking’ functions.

1.2 Methods and overview

In this paper we present a syntax-based language model that combines the strengths of C&J’s and Roark’s models. It is based on a novel efficient implementation of *stochastic left corner parsing* using rich non-local and lexicalized probabilistic conditioning of the parser moves.

Several efficiency improvements to the original left corner parsing algorithm have been published, but most of them are not easily generalized to a probabilistic framework. The key to our left corner parsing algorithm is the representation of several equivalent partial derivations by one context-enriched constituent, which holds all information that conditions the probability of the next parser move; by a careful definition of the constituent context, inspired by C&J’s, we can ensure that the resulting constituent, including its context, is completely defined by the original constituent and the parser move.

The ensemble of conditional probabilities of all parser moves defines a probability mass function over the space of left corner derivations, and will be called a probabilistic left corner grammar (PLCG).

In this framework, the set of all (partial) derivations that generate a (partial) input sentence is represented as a directed acyclic network of constituents connected by parser moves. The PLCG-based language model is essentially a dynamic programming parsing algorithm that traverses the network, accumulating the probabilities of all partial paths through the network. Duplication of work is avoided by avoiding duplication of constituents.

Due to the compact network representation, we lose the flexibility of choosing arbitrary context features for conditioning; we will verify experimentally that this limitation does not have to degrade performance as compared with Roark’s results.

The PLCG-based language model not only produces the probability of the whole input sentence, but also probabilities of partial sentences, synchronously with the parsing; therefore, it allows online computation of conditional language model probabilities. These probabilities are guaranteed to be normalized, even in the face

of pruning, because they can be rewritten as a weighted average of shift probabilities.

The result is a relatively efficient grammar-based language model which is proven to compare favorably with other recent competing grammar-based language models for speech recognition purposes.

The rest of this paper is organized as follows. Sec. 2 introduces specific definitions and notation. Starting from a standard formulation of left corner parsing, the framework of context-enriched constituents is gradually developed in Sec. 3. Based on this framework, Sec. 4 develops an efficient probabilistic left corner parsing algorithm. Sec. 5 modifies this algorithm to produce language model probabilities. Methods and results of our experiments are reported in Sec. 6. Sec. 7 concludes.

2 Definitions and notation

2.1 Language, push-down automata, and language modeling

In statistical speech recognition, natural language is modeled as a stream of word tokens. A *vocabulary* or *input alphabet* V is a set of word tokens; if the set size is fixed, then the vocabulary is *closed*, otherwise it is *open*. In this paper tokens are typeset in a typewriter typeface: we distinguish the *variable* w from the token “w”.

Sequences of variables and/or constants are written separated by a comma and enclosed in parentheses, e.g. (x, y, z) , or concatenated, like xyz . Greek lowercase letters (α, β, \dots) denote sequences; (Y, β) and $Y\beta$ denote the same sequence starting with Y . The empty sequence $()$ is denoted by ϵ . The expression w_i^j is a short notation for the sequence $(w_i, w_{i+1}, \dots, w_j)$, or for ϵ if $j < i$. An initial substring of a sequence is called a *prefix*. The set of sequences of length n containing elements from the same set V is denoted by V^n . The union $\{\epsilon\} \cup V \cup V^2 \cup \dots$ is denoted by V^* .

In Sec. 3, we will use left corner push-down automata (PDA) to develop the concept of PLCG-based language modeling. Our notation of PDA transitions follows (Hopcroft et al., 2001, Ch. 6), except that we leave out the state from the instantaneous description (cf. below) since the left corner PDAs are stateless. A *stateless PDA* is a 4-tuple (V, \mathcal{S}, m, q_0) where V is a finite alphabet of input symbols, \mathcal{S} is a finite alphabet of stack symbols, m is a transition function and q_0 is the initial stack symbol. We define m with rules that describe possible transitions from one instantaneous description to another one, formalized as a relation \vdash . An instantaneous description of the PDA is written as (w_i^j, π) , where w_i^j denotes the input sequence left to be consumed, and π denotes the contents of the stack. The stack is written

as a sequence with the stack top at its left end and the stack bottom at its right end. The operation

$$(w_i^j, \alpha\pi) \vdash (w_{i+1}^j, \beta\pi)$$

consumes the next input symbol (w_i) and replaces α on the stack with β . Note that w_i can be empty, in which case the operation does not consume input. Also α and/or β can be empty.

Let w_0, w_1, \dots, w_n be word tokens in a vocabulary V and let V also contain two special word tokens signaling the beginning and end of a sentence, namely $\langle s \rangle$ and $\langle /s \rangle$. A sequence w_0^n where $w_0 = \langle s \rangle$ and $w_1, \dots, w_n \in V \setminus \{\langle s \rangle, \langle /s \rangle\}$ is called a *sentence prefix*. If w_0^{n-1} is a sentence prefix and $w_n = \langle /s \rangle$ then w_0^n is a *sentence*.

A *statistical language model* (LM) is a probability mass function (*pmf*) $p(w_0^n)$ over the space of sentences. A *conditional LM* (CLM) is a set of conditional *pmfs* over V of the form $p(w_{i+1} = w | w_0^i = h)$. Note that a CLM defines a LM:

$$p'(w_0^n) = \prod_{i=1}^n p(w_i | w_0^{i-1})$$

The latter expression is called a *left-to-right factorization* of $p'(w_0^n)$. Through recursive updates of prefix probabilities $P(w_0^i) = P(w_0^{i-1})p(w_i | w_0^{i-1})$, a CLM allows a tight integration of the language model within the search engine of a recognizer, which is beneficial for time and space efficiency.

2.2 Stochastic grammars and parsing

A *context-free grammar* (CFG) Γ is a 4-tuple (N, V, P, S) , where N is a finite set of non-terminal category labels, V is a finite set of terminal category labels (the vocabulary), $P = \{(A \rightarrow \alpha) | A \in N, \alpha \in (N \cup V)^*\}$ is a finite set of context-free production rules and $S \in N$ is the axiom or start symbol. Latin capitals (A, B, C, \dots) represent non-terminals, latin lowercase letters (a, b, c, \dots) represent terminals, while lowercase greek letters ($\alpha, \beta, \gamma, \dots$) represent *sequences* of terminals and/or non-terminals.

We will use the *left corner relation*. For a CFG $\Gamma = (N, V, P, S)$, X is a left corner of Y if and only if there is some α for which there is a rule $Y \rightarrow X\alpha$ in P . We write this as $X \angle Y$.

The *derives* relation, written $\stackrel{\Gamma}{\Rightarrow}$ or \Rightarrow if Γ is clear from the context, is defined as follows: $X_0^n \Rightarrow X_0^{i-1} \alpha X_{i+1}^n$ if and only if there is a rule $(X_i \rightarrow \alpha) \in P$.

A *local tree* $t = (t_1^m)_X$ is either an empty sequence or a sequence of pointers to other local trees t_1, \dots, t_m , labeled with a category X . The function $R(t) = X$ returns the

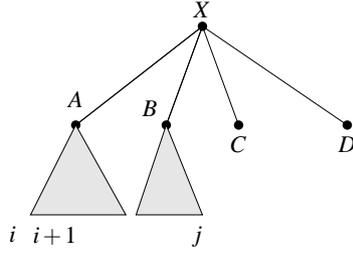


Fig. 1. A graphical representation of a constituent $q = [{}_iAB {}_jCD]_X$.

label of t , and by extension, $R(t_1^m) = R(t_1) \dots R(t_m)$. If $m \geq 1$, the local tree $t = (t_1^m)_X$ is only valid if there is a grammar rule $R(t) \rightarrow R(t_1^m)$. Otherwise t is a terminal node and $R(t)$ must be a terminal category. The *yield* of a local tree t , denoted by $Y(t)$, is equal to $R(t)$ if t is a terminal node; otherwise it is the concatenation of the yields of its children. A *parse tree* T is a set of connected local trees, and $Y(T) = Y(t)$ if t is the top local tree of T .

Note that there are multiple top-down derivations corresponding with one parse tree, since the order in which non-terminals are rewritten is arbitrary. The *leftmost derivation* is the top-down derivation that always rewrites the *leftmost* non-terminal (the corresponding relation is written \Rightarrow_L). There is a unique correspondence between a parse tree and its leftmost derivation.

The concept of a *constituent* links a local tree with its yield. It can be understood as a local tree of which the yield is partially or completely verified to correspond with a part of the input sentence. In this paper, we define a *constituent* $q = [{}_i\alpha {}_j\beta]_X$ of a parse tree T for which $Y(T) = w_1^n$, as the proposition that there is a local tree $(t_1^m)_X \in T$ for which $R(t_1^k) = \alpha$, $R(t_{k+1}^m) = \beta$ and $Y(t_1) \dots Y(t_k) = w_{i+1}^j$. Fig. 1 gives a graphical representation of a constituent $q = [{}_iAB {}_jCD]_X$.

We say that q is a constituent of w_1^n if there is at least 1 parse tree T where $Y(T) = w_1^n$ and q is a constituent of T . If $\beta = \varepsilon$, q is called *resolved*. Otherwise q is *unresolved*. For notational convenience, we define two functions $\text{pos}(q)$ and $\text{cat}(q)$, returning the position and the category of q , respectively. For instance, if $q = [{}_i\alpha {}_j\beta]_X$, then $\text{pos}(q) = j$ and $\text{cat}(q) = X$.

2.3 Parsing and language models

In natural language grammars, the generation of a sentence is initiated by a start symbol, usually called S. The sentence is complete as soon as a parse tree is found with S as the top category and the input sentence as its yield. Language modeling however, traditionally initiates a sentence with a start-of-sentence symbol, e.g. $\langle s \rangle$, and completes it with the emission of an end-of-sentence symbol, e.g. $\langle /s \rangle$.

These two different views can be made compatible by making the original local

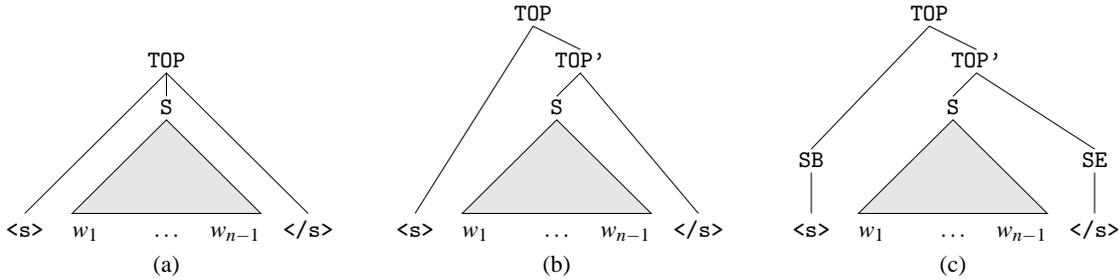


Fig. 2. Marking sentence boundaries in parsing-based language modeling.

parse tree $t_S = (\dots)_S$ a daughter of a local tree $(t_I, t_S, t_M)_{TOP}$ (cf. Fig. 2(a)). The yield of the resulting tree is w_0^n , where $w_0 = \langle s \rangle$, $w_n = \langle /s \rangle$ and the original input sentence was w_1^{n-1} . If the grammar or parser only handles binary trees, this can be accounted for with a TOP' tree as in Fig. 2(b). If the framework requires pre-terminals (e.g. parts-of-speech) between the input words and the actual parse tree, two pre-terminals SB (sentence begin) and SE (sentence end) can also be added, as in Fig. 2(c).

3 Probabilistic left corner parsing

In this section we gradually develop a novel formulation of probabilistic left corner parsing using context-enriched constituents. A context-enriched constituent holds all information that conditions the probability of a next parser move; by a careful definition of the constituent context, we can ensure that the resulting constituent, including its context, is completely defined by the original constituent and the parser move. This forms the key to the parsing algorithm developed in Sec. 4.

First we briefly review left corner derivation and introduce the left corner automaton. Its stochastic generalization is described next, followed by its extension involving lexicalized categories and extensive probabilistic conditioning of the move probabilities. Finally we introduce the concept of PLCG submodels and define the PLCG.

3.1 Non-probabilistic left corner parsing

Non-probabilistic left corner parsing is known as an efficient CFG parsing technique. It is often attributed to Rosenkrantz and Lewis II (1970), although a similar idea was already used by the SBT parser (Griffiths and Petrick, 1965). Several enhancements and extensions of the original method were proposed later (Matsumoto et al., 1983; Wirén, 1987; Nederhof, 1993; Moore, 2000). Left corner parsing is a special case of head corner parsing (Kay, 1989; van Noord, 1997). Although head corner parsing lies conceptually closer to modern linguistic theories, left corner

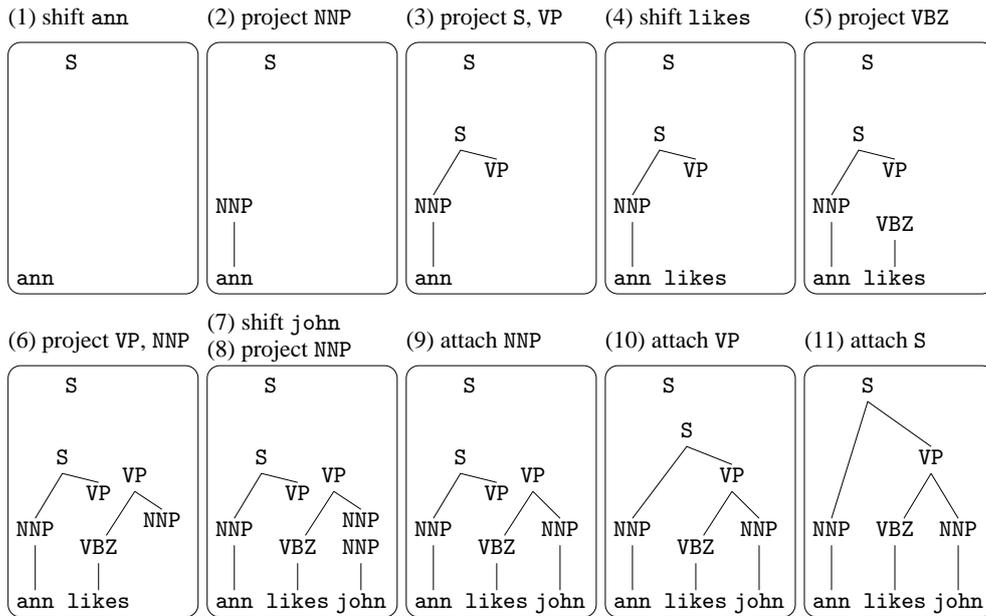


Fig. 3. Left corner derivation of a small parse tree.

parsing seems more suitable for integration in a statistical language model operating from left to right.

In Sec. 2.2 we saw that a parse tree uniquely corresponds with its leftmost derivation. However, there are many other canonical derivation schemes, *left corner derivation* being one of them. In left corner derivation, the generation of a full parse tree of a given category is initiated by predicting (*shifting*) the next word token (the leaf node at its bottom left), which is considered a (trivial) full parse tree. A full parse tree can be used for *projection* or be *attached* to another partial parse tree. Projecting from a full parse tree means constructing a local tree on top of the full parse tree, such that the full parse tree is the leftmost child node of the local tree; the other child nodes have a label but no children. The resulting partial tree is completed by ‘plugging in’ (*attaching*) other full parse trees of the appropriate categories at the child nodes. These full parse trees have each been generated by recursive application of this strategy.

To get a feel for left corner derivation, we detailed the left corner derivation of a small parse tree on the sentence ‘ann likes john’ in Fig. 3.

A left corner parser can be elegantly and concisely described as a non-deterministic stateless PDA (V, \mathcal{S}, m, q_I) , where

- (1) V is the set of terminals (word tokens);
- (2) \mathcal{S} is the set of constituents;
- (3) $q_I = [0 \ \varepsilon_0 \ S]$ is the initial stack element;
- (4) m is defined by the following operations:

(a) The SHIFT operation:

$$(w_{j+1}^n, ([i \alpha_j Y \beta]_X, \pi)) \vdash (w_{j+2}^n, ([j w_{j+1} w_{j+1}]_w, [i \alpha_j Y \beta]_X, \pi)) \quad (1)$$

for any $i, j, \alpha, \beta, X, Y$ and π (which denotes the rest of the stack). In words, the shift operation takes the next word w_{j+1} , builds a new constituent with it and pushes it on the stack. It is applicable whenever the stack top is an unresolved constituent.

(b) A number of PROJECT(U, δ) operations:

$$(w_{k+1}^n, ([j \alpha_k]_X, [i \beta_j Y \gamma]_Z, \pi)) \vdash (w_{k+1}^n, ([j X_k \delta]_U, [i \beta_j Y \gamma]_Z, \pi)) \quad (2)$$

for any $i, j, k, \alpha, \beta, \gamma, X, Y, Z$ and π . In words, a PROJECT(U, δ) operation consumes no input and replaces the stack top with a new constituent. The stack top must be a resolved constituent and figures as the leftmost child of the new constituent.

(c) The ATTACH operation:

$$(w_{k+1}^n, ([j \alpha_k]_X, [i \beta_j X \gamma]_Z, \pi)) \vdash (w_{k+1}^n, ([j \beta X_k \gamma]_Z, \pi)) \quad (3)$$

for any $i, j, k, \alpha, \beta, \gamma, X, Z$, and π . In words, the ATTACH operation is applicable if the category of the stack top X equals the first unresolved daughter of the constituent below it. The stack top is popped, and the constituent below it is modified by marking the daughter X as resolved.

(d) The PDA terminates successfully by empty stack if all input is consumed and the stack only contains the final constituent $q_F = [{}_0 S_n]$:

$$(\varepsilon, ([{}_0 S_n])) \vdash (\varepsilon, \varepsilon) \quad (4)$$

A deterministic left corner parser can be constructed from the PDA described above with a general method described by (Lang, 1974).

Note that dead paths can be abandoned early by checking whether $w \angle^* Y$ when shifting w (cf. Eq. (1)) and whether $U \angle^* Y$ when projecting U in order to ultimately obtain a constituent of category Y (cf. Eq. (2)). These checks, constituting *top-down filtering*, can be implemented as a simple look-up in a pre-compiled table.

3.2 Probabilistic left corner parsing: previous art

In a probabilistic CFG (PCFG), each production rule $A \rightarrow \alpha$ is annotated with the conditional probability $P(\alpha|A)$ that the given lefthand side A produces the righthand side α . Although this probability is successfully handled in various PCFG parsers, its definition is most logical from the viewpoint of top-down derivation.

Starting from a more general view on derivation, conceptually simpler probabilistic parsers are obtained as follows.

Consider a derivation m_1^p , which is a sequence of p moves m_1, m_2, \dots, m_p . Then $P(m_1^p) = \prod_{i=1}^p P(m_i | m_1^{i-1})$. We call $P(m_1^p)$ a *derivation probability* or *path probability* and $P(m_i | m_1^{i-1})$ a *move probability* or *transition probability*. The space of derivations can be organized as a prefix tree in which a path from the root to one of the leaves represents one derivation. Simple probabilistic parsing is realized by searching the most probable path (or all ‘sufficiently’ probable paths) in the derivation prefix tree.

If a derivation uniquely corresponds with a parse tree, then the probability of the parse tree can be computed as the probability of its derivation and $P(m_i | m_1^{i-1})$ is equivalent with $P(m_i | t(m_1^{i-1}))$ where $t(m_1^{i-1})$ is the partial parse tree generated with m_1^{i-1} .

A probabilistic left corner automaton (PLCA) is now obtained straightforwardly by annotating the original stack rules with conditional move probabilities $P(m_i = \text{SHIFT}(w) | m_1^{i-1})$, $P(m_i = \text{PROJECT}(U, \delta) | m_1^{i-1})$ and $P(m_i = \text{ATTACH} | m_1^{i-1})$, where m_1^{i-1} denotes the preceding moves. If one is only interested in the best parse tree of a given sentence, there is no need to treat the $\text{SHIFT}(w)$ move probabilistically where w is the next word to be shifted, since it is possible when and only when the top of the stack is unresolved. However, in order to obtain an estimate of the probability of the input sentence (instead of a parse tree) the $\text{SHIFT}(w)$ move has to be predicted probabilistically. Of course,

$$\sum_{w \in V} P(\text{SHIFT}(w) | m_1^{i-1}) + \sum_{\substack{U \in N \\ \delta \in (N \cup V)^*}} P(\text{PROJECT}(U, \delta) | m_1^{i-1}) + P(\text{ATTACH} | m_1^{i-1}) = 1 \quad (5)$$

must hold for each derivation prefix m_1^{i-1} .

Previously described PLCAs can be recognized as simplifications of the above general description. For example, the PLCA described by Manning and Carpenter (1997) can be regarded as the result of three simplification steps of m_1^{i-1} :

- (1) The derivation prefix m_1^{i-1} is replaced by the stack it produces.
- (2) The stack is interpreted as one of the lefthand sides of Eqs. 1–3. That is, the prediction of a $\text{SHIFT}(w)$ is conditioned only on the top of the stack, while a $\text{PROJECT}(U, \delta)$ and an ATTACH are conditioned only on the top and the subtop.
- (3) Referring to the Eqs. 1–3, the $\text{SHIFT}(w)$ move is conditioned only on Y . The $\text{PROJECT}(U, \delta)$ move is conditioned only on X and Y . The ATTACH move is conditioned only on X .

The behavior of simplified PLCAs such as Manning and Carpenter’s is simulated

by standard chart parsers using left corner transformed PCFGs. However, direct estimation of the PLCA move probabilities from a treebank can apply statistical smoothing techniques straightforwardly. Obtaining equivalent results with a PCFG is not possible with the standard left corner transform.

3.3 Probabilistic left corner parsing: extension

The standard PLCA described in the previous section is now extended in two ways.

3.3.1 Constituent context

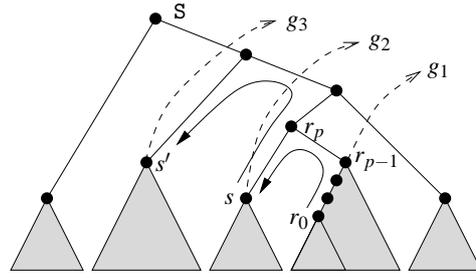
We want to relax the independence assumptions on the move probabilities. At the same time however, we seek to simplify the PLCA rules Eqs. 1–3 to rules that only specify the top constituent on the stack, which will allow easy model reestimation in the spirit of the Baum-Welch algorithm. Both goals are realized by integrating all conditioning features in the top stack element:

- We define a *local tree context* \vec{g} as a set of features of a parse tree with respect to a local tree in that parse tree.
- We redefine a constituent $x = [{}_i \alpha {}_j \beta | \vec{g}]_X$ as the set of parse trees that contain a local tree $t = (t_1^m)_X$ for which $R(t_1^m) = \alpha\beta$ and $\alpha \Rightarrow^* w_{i+1}^j$ and this local tree has a local tree context \vec{g} .

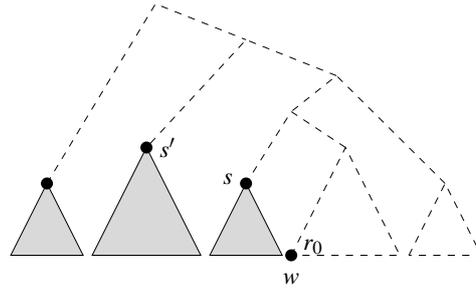
For a local tree context to be useful in parsing, it should be recognizable from the part of the parse tree that is already constructed at the time the local tree which it conditions is being constructed.

The local tree context used in the C&J model consists of the categories of the two rightmost isolated trees (i.e., they are not yet a subtree of another tree). We give an alternative definition here, based on the full parse tree (see Fig. 4(a)), because it will show the equivalence of C&J with the local tree context in our PLCA-based model. Our local tree context \vec{g} consists of three features (g_1, g_2, g_3) . Given a local tree t , its local tree context feature g_2 is found by calling $t = r_0$ and considering the sequence $(r_0 = t, r_1, \dots, r_p)$ for which r_{i-1} is the first child of r_i for $i = 1 \dots p-1$ and r_{p-1} is a daughter, but not the first, of r_p . If the first daughter of r_p is denoted by s , then $g_2 = R(s)$. g_3 is defined as the g_2 context feature of s : $g_3 = R(s')$, where s' is found in the same way from s as s was found from r_0 . Finally g_1 is defined as $R(r_{p-1})$. As shown in Fig. 4(c), r_{p-1} is not yet available when constructing r_0 ; however, its label $R(r_{p-1})$ has already been predicted by the projection of r_p from s .

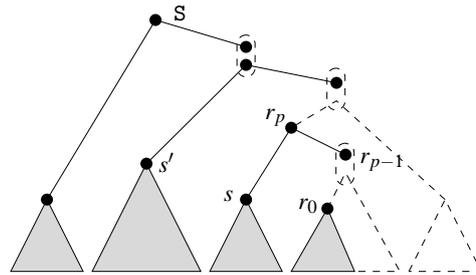
As will be shown below, this definition of \vec{g} is well-chosen in the sense that a context-enriched constituent, resulting from an operation on an originating con-



(a)



(b)



(c)

Fig. 4. (g_1, g_2, g_3) context of a local tree $t = r_0$. (a) Definition of g_1, g_2 and g_3 on a full parse tree. (b) In the C&J model: at the time w is shifted, only s and s' are known. (c) In the PLCA model: at the time r_0 is constructed, s and s' are available; r_{p-1} is not yet known, but $R(r_{p-1})$ is. (Dashed lines indicate parts of the parse tree that await completion. The dashed boxes indicate pending attach moves.)

stituent, is *completely* determined by this operation and the originating constituent.

Fig. 4(b) shows the equivalence with the probabilistic conditioning of the shift-reduce parsing mechanism used by the C&J model. This parser creates binary parse trees only. The part of the parse tree that has not yet been predicted, at the time r_0 is being constructed, is drawn with dashed lines. Only s and s' are available, there is no such node as r_{p-1} yet. This leads to an essential difference with our model: there is no context feature g_1 . (In the original C&J model, $g_3 = R(s')$ and $g_2 = R(s)$ condition the construction of r_0 only if r_0 is a leaf node (shift move). C&J's reduce steps are independent from s' , presumably for practical reasons, but it is theoretically possible to include s' too.)

We now redefine the PLCA with context-conditioned constituents as stack elements. Given an input sentence w_0^n :

- The stack is initialized with (cf. Fig. 2(c))

$$q_I = [{}_0 \text{ SB } {}_1 \text{ TOP}' | \text{TOP}, \text{SB}, \text{SB}]_{\text{TOP}}$$

The second and third element of the context do not correspond with our context definition; they are just used to avoid additional symbols.

- The stack is emptied and the PLCA terminates successfully if it only contains (cf. Fig. 2(c))

$$q_F = [{}_0 \text{ SB } \text{ TOP}' {}_n | \text{TOP}, \text{SB}, \text{SB}]_{\text{TOP}}$$

- The original SHIFT operation Eq. (1) is modified as:

$$(w_{j+1}^n, ([{}_i A \alpha {}_j Y \beta | \vec{g}]_X, \pi)) \quad \vdash \quad (w_{j+2}^n, ([{}_j w_{j+1} {}_{j+1} | \vec{h}]_{w_{j+1}}, [{}_i A \alpha {}_j Y \beta | \vec{g}]_X, \pi))$$

where $\vec{g} = (g_1, g_2, g_3)$ and $\vec{h} = (Y, A, g_2)$. The construction of \vec{h} can be understood by looking at Fig. 4(c): a new constituent, $[{}_j w_{j+1} {}_{j+1} | \vec{h}]_{w_{j+1}}$, corresponding with r_0 , is created and pushed on the stack when an unresolved constituent, $[{}_i A \alpha {}_j Y \beta | \vec{g}]_X$, corresponding with r_p , was encountered on top of the stack. The first unresolved daughter constituent of r_p is h_1 and equals Y . h_2 is found as the category of the leftmost daughter of r_p , which is A . The g_2 context feature is inherited as h_3 : it corresponds with the category of s' on the same figure.

- The original form of a PROJECT(U, δ) move Eq. (2) becomes:

$$(w_{k+1}^n, ([{}_j \alpha {}_k | \vec{g}]_X, \pi)) \quad \vdash \quad (w_{k+1}^n, ([{}_j X {}_k \delta | \vec{g}]_U, \pi))$$

where $\vec{g} = (Y, B, h_2)$. Again referring to Fig. 4(c), if $[{}_i \alpha {}_j | \vec{g}]_X$ would correspond with r_0 , then $[{}_j X {}_k \delta | \vec{g}]_U$ would correspond with r_1 and $p > 1$. Hence \vec{g} is simply inherited. Note that in case $X = Y$, the ATTACH is applicable too (see next).

- Finally, the original ATTACH operation Eq. (3) becomes:

$$(w_{k+1}^n, ([{}_j \gamma {}_k | \vec{h}]_Y, [{}_i A \alpha {}_j Y \beta | \vec{g}]_X, \pi)) \quad \vdash \quad (w_{k+1}^n, ([{}_i A \alpha Y {}_k \beta | \vec{g}]_X, \pi))$$

where $\vec{h} = (Y, A, g_2)$. For convenience, we consider attachment as a special projection: $\text{ATTACH} \equiv \text{PROJECT}(\text{ATT})$ and require $P(\text{PROJECT}(\text{ATT}) | [{}_j \gamma {}_k | \vec{h}]_Y) = 0$ if $h_1 \neq Y$.

3.3.2 Lexicalized categories

It is empirically found that syntactic categories, however fine-grained they are, lose important information that statistically influences certain structural pattern preferences. A common and simple approach is to keep record of a lexical feature besides

a syntactic category feature, leading to *lexicalized* grammars. For instance, in a typical lexicalized grammar, the lexical feature of the noun phrase ‘an evil monster’ would be ‘monster’.

We apply the same principle to the PLCA. We introduce some more notation for this purpose: we will use bold Latin capitals to denote a composite category label consisting of a syntactic category and a lexical category separated with a slash (/); bold Greek letters denote sequences of these. A terminal w is replaced with a composite W/w using a dummy W syntactic feature.

We also need to specify how the lexical feature is determined: we assume that the lexical feature of the mother constituent can be determined as a function of its syntactic category and the composite categories of its daughters, and call this function $\text{head}(\cdot)$. This assumption seems sufficient for English, but it may be too restrictive for other languages.¹

Given the input sentence w_0^n , the format of the stack rules of our extended PLCA is then again adapted to the following final form.

- The stack is initialized with

$$q_I = [{}_0 \text{SB}/\langle s \rangle \text{ TOP}' | \text{TOP}, \text{SB}/\langle s \rangle, \text{SB}/\langle s \rangle]_{\text{TOP}} \quad (6)$$

- The stack is emptied and the PLCA terminates successfully if it only contains

$$q_F = [{}_0 \text{SB}/\langle s \rangle \text{ TOP}' \text{ }_n | \text{TOP}, \text{SB}/\langle s \rangle, \text{SB}/\langle s \rangle]_{\text{TOP}} \quad (7)$$

- A $\text{SHIFT}(w_{j+1})$ move applies

$$(w_{j+1}^n, ([{}_i \mathbf{A} \alpha_j Y \beta | \vec{g}]_X, \pi)) \vdash (w_{j+2}^n, ([{}_j \mathbf{W}_{j+1} | \vec{h}]_W, [{}_i \mathbf{A} \alpha_j Y \beta | \vec{g}]_X, \pi)) \quad (8)$$

with a probability $P(\text{SHIFT}(w_{j+1}) | [{}_i \mathbf{A} \alpha_j Y \beta | \vec{g}]_X)$ where

$$\begin{aligned} \vec{h} &= (Y, \mathbf{A}, \mathbf{g}_2) \\ \mathbf{W} &= W/w_{j+1} \\ \sum_{w \in V} P(\text{SHIFT}(w) | [{}_i \mathbf{A} \alpha_j Y \beta | \vec{g}]_X) &= 1 \end{aligned}$$

- A move $\text{PROJECT}(U, \delta)$ applies

$$(w_{k+1}^n, ([{}_j \alpha_k | \vec{g}]_X, \pi)) \vdash (w_{k+1}^n, ([{}_j \mathbf{X}_k \delta | \vec{g}]_U, \pi)) \quad (9)$$

¹ In our experiments, we even assumed that the lexical category of a constituent is equal to the lexical category of its daughter at head position, and that this head position can be precomputed for each grammar rule.

with a probability $P(\text{PROJECT}(U, \delta)|[{}_j \alpha_k | \vec{g}]_X)$ where

$$\begin{aligned} \mathbf{X} &= X/\text{head}(X, \alpha) \\ \sum_{U, \delta} P(\text{PROJECT}(U, \delta)|[{}_j \alpha_k | \vec{g}]_X) &= 1 - P(\text{ATTACH}|[{}_j \alpha_k | \vec{g}]_X) \end{aligned}$$

- The ATTACH move applies

$$\left(w_{k+1}^n, ([{}_j \gamma_k | \vec{h}]_Y, [{}_i \mathbf{A} \alpha_j Y \beta | \vec{g}]_X, \pi) \right) \vdash \left(w_{k+1}^n, ([{}_i \mathbf{A} \alpha \mathbf{Y} \beta | \vec{g}]_X, \pi) \right) \quad (10)$$

with a probability $P(\text{ATTACH}|[{}_j \gamma_k | \vec{h}]_Y)$ where

$$\begin{aligned} \vec{h} &= (Y, \mathbf{A}, \mathbf{g}_2) \\ \mathbf{Y} &= Y/\text{head}(Y, \gamma) \\ P(\text{ATTACH}|[{}_j \gamma_k | \vec{h}]_Y) &= 0 \quad \text{if } Y \neq h_1 \end{aligned}$$

Again, one may opt to regard ATTACH equivalent with PROJECT(ATT).

We will henceforth assume that conditional moves are statistically independent from α (the categories of the resolved daughters except the first one). Its role in the evaluation of $\text{head}(X, \alpha)$ can be accounted for in other ways if we can assume that the $\text{head}(\cdot)$ function selects a head position and returns the lexical feature of the head daughter: for instance, the head position is looked up when the syntactic features of the mother and the sisters are projected, and the lexical head of the sister at head position is propagated as soon as it is being attached. So we can replace α with a wildcard '*', which enhances the time and space efficiency of the parser.²

3.4 Inducing a probabilistic left corner grammar from a treebank

3.4.1 Submodels

The move probabilities are conditioned on the topmost constituent of the PLCA stack. In our work, these probabilities are estimated from training data. Given that there is always a limit to available matching training data, we will assume independence of PLCA moves from most of the features of the topmost constituent. Only a few of them can effectively serve for probabilistic conditioning. A typical parameterization is (cf. Eqs. 23–24):

$$P(\text{SHIFT}(w)|[{}_i \mathbf{A} * {}_j Y \beta | \vec{h}]_X) = p_s(w|Y, \mathbf{A}, \mathbf{h}_2) \quad (11)$$

$$P(\text{PROJECT}(U, \delta)|[{}_j \mathbf{A} * {}_k | \vec{g}]_X) = p_p(U, \delta|g_1, X, \mathbf{A}, \mathbf{g}_2) \quad (12)$$

$$P(\text{ATTACH}|[{}_j \mathbf{A} * {}_k | \vec{g}]_X) = p_a(\text{ATT}|g_1, X, \mathbf{A}, \mathbf{g}_2) \quad (13)$$

² This is related with, but not equal to an optimization proposed by Leermakers (1992), who would also discard the leftmost daughter \mathbf{A} .

p_s, p_p and p_a are ensembles of conditional *pmfs* and will be called *submodels* in this paper.

After fixing the parameterization, the submodels are initialized using methods similar with C&J, as explained next. The training corpus is a human-annotated or machine-generated treebank. Each tree is decomposed into its PLCA derivation steps. That is, given the above parameterization, the treebank is transformed in a stream of independent joint events of the types

$$\begin{aligned} &(\text{SHIFT}(w), Y, \mathbf{A}, \mathbf{h}_2) \\ &(\text{PROJECT}(U, \delta), g_1, X, \mathbf{A}, \mathbf{g}_2) \\ &(\text{ATTACH}, g_1, X, \mathbf{A}, \mathbf{g}_2) \end{aligned}$$

The shift events observed by the C&J model would rather look as $(\text{SHIFT}(w), \mathbf{A}, \mathbf{h}_2)$, that is, without Y .

Then, submodels are estimated from relative frequencies, using standard language modeling techniques such as smoothing, interpolation and back-off. For instance, a p_s smoothed by linear interpolation with a lower order version of it, say \tilde{p}_s , may look like:

$$\tilde{p}_s(w|Y, \mathbf{A}, \mathbf{h}_2) = d \times \frac{C(\text{SHIFT}(w), Y, \mathbf{A}, \mathbf{h}_2)}{\sum_{v \in V} C(\text{SHIFT}(v), Y, \mathbf{A}, \mathbf{h}_2)} + (1 - d) \times \tilde{p}_s(w|Y, \mathbf{A})$$

where $C(x)$ is the observed frequency of the event x in the training corpus, and d is an interpolation factor.

Note 1 *Linguistically speaking it does not make sense to assign an attach probability greater than 0 if $X \neq g_1$; in our experiments, we always disallowed attachment if $X \neq g_1$. In afterthought however, for the language model it may be useful to still allow attachment in this case; one would rely on the reestimation procedure (cf. Sec. 5.3) to find the optimal estimate, which may be greater than 0.*

Note 2 *It may be useful to have different parameterizations for projections at different levels in the parse tree.*

3.4.2 Probabilistic Left Corner Grammar

We can now define a probabilistic left corner grammar (PLCG) as a 5-tuple $\Gamma = (N, V, p_s, p_p, p_a)$ where N is a finite set of non-terminal category labels including W , V is a finite set of terminal category labels (the vocabulary), p_s is a shift submodel, p_t is a project submodel and p_a is an attach submodel.

Note that there is no need to specify a finite set of rules in a PLCG: all necessary information is given by the submodels. Given a PLCG, the construction of a corresponding PLCA is trivial.

The submodels are initialized with a treebank using statistical smoothing methods. All trees are decomposed in their PLCA derivation sequences, and submodel parameters are estimated as smoothed frequency statistics of PLCA derivation steps.

An additional advantage of estimating the submodels directly from a treebank is the flexibility in the statistical estimation process; for instance, various smoothing methods can be applied straightforwardly without complicating the estimation algorithm.

An equivalent model could be obtained by estimating a PCFG from a left corner transformed treebank and applying top-down parsing with the PCFG, instead of native left corner parsing (Johnson, 1998; Roark and Johnson, 1999). The left corner transform blows up the number of category tags and production rules, a fortiori with non-local conditioning on lexical context features. This necessitates proper tying of the conditional rule probabilities, which may have been overlooked by Roark and Johnson.

Alternatively, the submodels could be computed from a PCFG with a method similar to Stolcke’s method to compute a probabilistic transitive left corner relation from a PCFG (Stolcke, 1995). It is however impractical for large PCFGs, especially with lexicalization and encoding of context in the category labels.

3.5 *A toy example*

Table 1 shows one possible execution trace of an artificial PLCA generating the sentence `<s> ann likes john </s>`. The 3rd column is the conditional probability of the move (in the 2nd column), given the constituent at the stack top (in the 1st column).

4 **PLCG parsing in a compact network**

In this section we develop an efficient synchronous PLCG parsing algorithm. This algorithm is formulated as a traversal through a directed acyclic network of (context-enriched) constituents, connected by elementary parser moves. Each path through this network corresponds with a derivation. Following the simple, yet powerful principle of dynamic programming, duplication of parsing work is avoided by avoiding duplicate constituents. In the resulting compact network one can identify subnetworks situated between a SHIFT move and its corresponding ATTACH move; such a subnetwork realizes a previously projected constituent, and is shared by all other derivations projecting the same constituent. Each such subnetwork recursively contains other subnetworks down to the level of elementary W constituents.

Table 1

A PLCA trace generating the sentence `<s> ann likes john </s>`.

constituent at stack top	move	prob
$[_0 \text{SB}/\langle s \rangle \text{ } _1 \text{TOP}' \text{TOP}, \text{SB}/\langle s \rangle, \text{SB}/\langle s \rangle]_{\text{TOP}}$	SHIFT(<code>ann</code>)	0.1
$[_1 \text{W}/\text{ann} \text{ } _2 \text{TOP}', \text{SB}/\langle s \rangle, \text{SB}/\langle s \rangle]_{\text{W}}$	PROJECT(<code>NNP</code> , ϵ)	1.0
$[_1 \text{W}/\text{ann} \text{ } _2 \text{TOP}', \text{SB}/\langle s \rangle, \text{SB}/\langle s \rangle]_{\text{NNP}}$	PROJECT(<code>S</code> , <code>VP</code>)	0.7
$[_1 \text{NNP}/\text{ann} \text{ } _2 \text{VP} \text{TOP}', \text{SB}/\langle s \rangle, \text{SB}/\langle s \rangle]_{\text{S}}$	SHIFT(<code>likes</code>)	0.2
$[_2 \text{W}/\text{likes} \text{ } _3 \text{VP}, \text{NNP}/\text{ann}, \text{SB}/\langle s \rangle]_{\text{W}}$	PROJECT(<code>VBZ</code> , ϵ)	1.0
$[_2 \text{W}/\text{likes} \text{ } _3 \text{VP}, \text{NNP}/\text{ann}, \text{SB}/\langle s \rangle]_{\text{VBZ}}$	PROJECT(<code>VP</code> , <code>NNP</code>)	0.5
$[_2 \text{VBZ}/\text{likes} \text{ } _3 \text{NNP} \text{VP}, \text{NNP}/\text{ann}, \text{SB}/\langle s \rangle]_{\text{VP}}$	SHIFT(<code>john</code>)	0.1
$[_3 \text{W}/\text{john} \text{ } _4 \text{NNP}, \text{VBZ}/\text{likes}, \text{NNP}/\text{ann}]_{\text{W}}$	PROJECT(<code>NNP</code> , ϵ)	1.0
$[_3 \text{W}/\text{john} \text{ } _4 \text{NNP}, \text{VBZ}/\text{likes}, \text{NNP}/\text{ann}]_{\text{NNP}}$	ATTACH	1.0
$[_2 \text{VBZ}/\text{likes} \text{ } _3 \text{NNP}/\text{john} \text{ } _4 \text{VP}, \text{NNP}/\text{ann}, \text{SB}/\langle s \rangle]_{\text{VP}}$	ATTACH	1.0
$[_1 \text{NNP}/\text{ann} \text{ } \text{VP}/\text{likes} \text{ } _4 \text{TOP}', \text{SB}/\langle s \rangle, \text{SB}/\langle s \rangle]_{\text{S}}$	PROJECT(<code>TOP'</code> , <code>SE</code>)	0.8
$[_1 \text{S}/\text{likes} \text{ } _4 \text{SE} \text{TOP}', \text{SB}/\langle s \rangle, \text{SB}/\langle s \rangle]_{\text{TOP}'}$	SHIFT(<code></s></code>)	1.0
$[_4 \text{W}/\langle s \rangle \text{ } _5 \text{SE}, \text{S}/\text{likes}, \text{SB}/\langle s \rangle]_{\text{W}}$	PROJECT(<code>SE</code> , ϵ)	1.0
$[_4 \text{W}/\langle s \rangle \text{ } _5 \text{SE}, \text{S}/\text{likes}, \text{SB}/\langle s \rangle]_{\text{SE}}$	ATTACH	1.0
$[_1 \text{S}/\text{likes} \text{ } \text{SE}/\langle s \rangle \text{ } _5 \text{TOP}', \text{SB}/\langle s \rangle, \text{SB}/\langle s \rangle]_{\text{TOP}'}$	ATTACH	1.0
$[_0 \text{SB}/\langle s \rangle \text{ } \text{TOP}' / \langle s \rangle \text{ } _5 \text{TOP}, \text{SB}/\langle s \rangle, \text{SB}/\langle s \rangle]_{\text{TOP}}$		
Total derivation probability		0.00056

This nesting of shared subnetworks makes our parsing algorithm efficient.

Being ultimately interested in language modeling, we pay special attention to how the sum of the probabilities of distinct parsing paths can be obtained efficiently. Not only probability sums of full paths, but also those of partial paths are needed to compute conditional language model probabilities.

4.1 The PLCG network

We now formally move from a PLCA view on the PLCG to a network view. First, it is important to realize that the contents of the PLCA stack can be reconstructed at any time if the preceding moves are known and that in the previous section we designed the PLCA such that the next moves are conditioned on the current top of the stack only. This leads to a linear directed graph representation of the generation of a parse tree by a PLCA where nodes are labeled with the top of the stack (instead

of the complete stack) and arcs are labeled with a move. Henceforth, we call this linear graph a *path*. The probability of the path is the product of the conditional probabilities of its moves, so it equals the probability of the corresponding parse tree.

Given a PLCG, the associated network is the union graph of all paths with a probability greater than 0. This network could be a path prefix tree, and as such it would be equivalent with a derivation tree. However, in this paper we prefer to make this network as compact as possible. This is obtained by the requirement that there are no two nodes with the same label:

A *PLCG network* $\mathcal{G} = (\mathcal{G}^n, \mathcal{G}^a)$ is a directed acyclic graph consisting of a set of nodes \mathcal{G}^n and a set of directed arcs $\mathcal{G}^a \subset \mathcal{G}^n \times \mathcal{G}^n$. Each node $q \in \mathcal{G}^n$ is labeled with a constituent and there are no two nodes with the same label. Each arc is labeled with a move and the conditional probability of that move given the label of the source node. The w_0^n -constrained network $\mathcal{G}_{w_0^n}$ is the subgraph of \mathcal{G} that includes all paths that generate parse trees that yield w_0^n with a probability greater than 0, but no other paths. There is at most one arc between two nodes.

For convenience of notation, we treat nodes and arcs as completely equivalent with their labels. For instance, $q = [{}_i Y * {}_j \beta | \vec{h}]_X$ denotes that the label of the node q is $[{}_i Y * {}_j \beta | \vec{h}]_X$. And $(q_i, q_j) = \text{ATTACH}(q_k)$ denotes that the arc from q_i to q_j is labeled $\text{ATTACH}(q_k)$. The conditional move probability is written as $P(q_j | q_i)$.

Using a parse tree representation adapted for language modeling (cf. Fig. 2(c)), the *initial node* is q_I (cf. Eq. (6)), and the *final node* is q_F (cf. Eq. (7)), where n is the number of input words, excluding $\langle s \rangle$ but including $\langle /s \rangle$.

A *path* is a linear subgraph of \mathcal{G} , represented as a sequence of moves or a sequence of constituents. The *path set* denoted by $\langle q_i, q_j \rangle$ is the set of all paths starting from q_i and arriving in q_j . ℓ is a *complete path* if $\ell \in \langle q_I, q_F \rangle$. We say that $q_i \prec q_j$ (q_i precedes q_j) if $\langle q_i, q_j \rangle \neq \emptyset$.

We define a *path concatenation* operator for convenience. The concatenation of two paths ℓ_1 and ℓ_2 , written as $\ell_1 \ell_2$, is the union graph of ℓ_1 and ℓ_2 . It is only defined if ℓ_1 arrives in the node where ℓ_2 departs. For path sets,

$$\langle q_1, q_2 \rangle \langle q_2, q_3 \rangle \doteq \{((q_1, q_2))\ell | \ell \in \langle q_2, q_3 \rangle\} \quad (14)$$

$$\langle q_1, q_2 \rangle (q_2, q_3) \doteq \{\ell((q_2, q_3)) | \ell \in \langle q_1, q_2 \rangle\} \quad (15)$$

$$\langle q_1, q_2 \rangle \langle q_2, q_3 \rangle \doteq \{\ell_1 \ell_2 | \ell_1 \in \langle q_1, q_2 \rangle, \ell_2 \in \langle q_2, q_3 \rangle\} \quad (16)$$

Note that if $\ell_1 \ell_2$ is defined, ℓ_1 and ℓ_2 have only one node in common; if there were another common node, \mathcal{G} would contain a cycle, which is in contradiction with the definition.

As an illustration, Fig. 5 shows a fragment of a severely pruned PLCG network. It

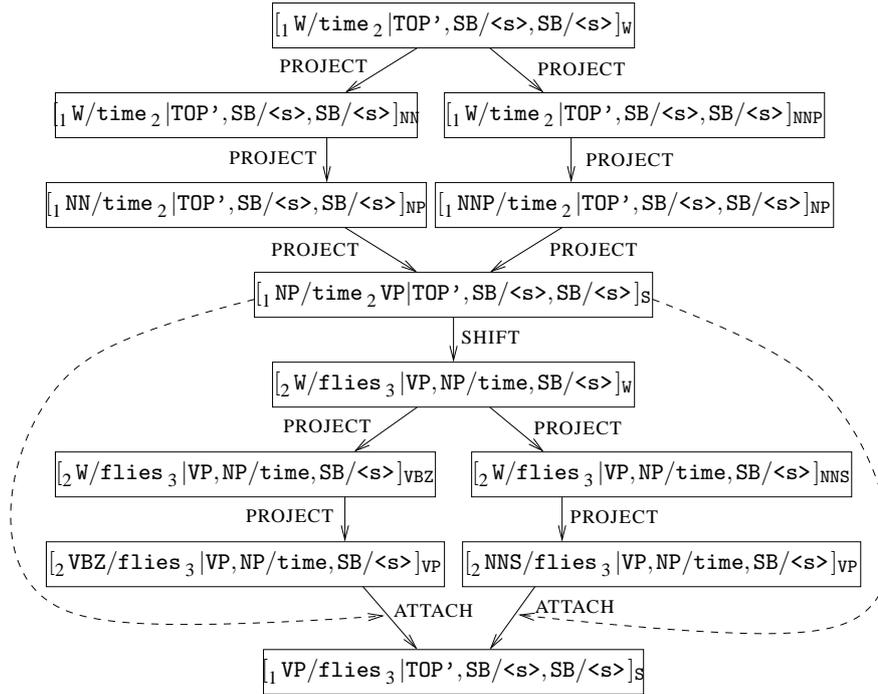


Fig. 5. Fragment of a (severely pruned) PLCG network constrained to $\langle s \rangle$ time flies $\langle /s \rangle$. A dashed arrow connects an ATTACH with its argument.

contains 4 partial paths.

4.2 Computing sums of path probabilities

One node in the PLCG network corresponds with multiple partial derivations. This section explains how the sum of their probabilities — which will be called the *forward probability* of the node — can be computed efficiently.

Efficiency is in principle obtained by the *dynamic programming* (DP) principle of storing (tabulating) and maximally re-using intermediate results. In PCFG chart parsing, a well-known DP parsing algorithm, the “intermediate result” is stored as a chart item annotated with its probability. The item is constructed from previously constructed daughter items, and its probability is computed from the probabilities of its daughter items. In the PLCG network, the “intermediate result” is constructed as a network node annotated with its forward probability. The node and its forward probability is constructed from previously constructed nodes and their probabilities; there must be a valid PLCG move between the node and each of its predecessors. Duplication of work is avoided by requiring that each node be unique.

Actually, the computation of yet another probability, the *inner probability*, is necessary in the PLCG network, because of the “attach constraint”, which we will explain first.

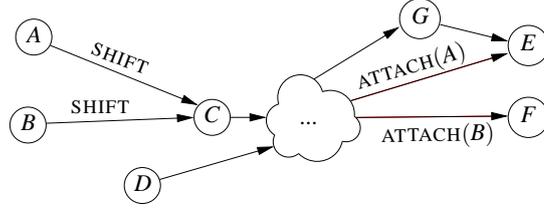


Fig. 6. The PLCG network is not Markov due to the constraint that a path containing $\text{ATTACH}(q)$ must visit q . The sequence (A, C, \dots, F) does not constitute a valid path. Neither does (B, C, \dots, E) . On the other hand (B, C, \dots, F) and (B, C, \dots, G, E) are valid paths.

4.2.1 The attach constraint

A move probability is conditioned only on its source node q . However, there is one subtlety to this. Within one single path, an ATTACH move refers implicitly, but unambiguously, to an *attachment* node: the node labeled with the constituent that is being completed by the constituent at the source node of the ATTACH move. However, in a graph, the ATTACH move cannot extend a path prefix that does not contain a corresponding attachment node. We refer to this non-Markov property as the *attach constraint*. In order to be able to decide whether the attach constraint is satisfied, the ATTACH move is henceforth augmented with the corresponding attachment node as its argument. Fig. 6 should make this more clear.

Fortunately, within the same path attach constraints are *nested*, which means that any other legal subpath between q and $\text{ATTACH}(q)$ found in the PLCG network can be substituted to form another legal full path. This property is formally expressed as follows. The proof is given in Appendix A.

Lemma 1 *Let q_1 and q_2 be nodes in a PLCG network \mathcal{G} where $q_1 \prec q_2$, $t \in \langle q_1, q_2 \rangle$ and $(q, q') = \text{ATTACH}(q')$ be a move in t . If $\text{ATTACH}(s)$ is a move in a path $u \in \langle q', q \rangle$ then $q'' \prec s$.*

The following lemma identifies a subgraph $\langle q^o, q \rangle$ that will play a role in the definition of the inner probability. The proof is also given in Appendix A.

Lemma 2 *If $q = [{}_i \mathbf{X} * {}_j \beta | \vec{h}]_Z$, then on every w_0^i -constrained path in $\langle q_I, q \rangle$ there is a node $q^o = [{}_i w_{i+1} | {}_{i+1} \vec{h}]_W$.*

The superscript ‘o’ will henceforth be used in this meaning.

4.2.2 Forward and inner probability

A language model returns an estimate of the probability of an input sentence w_0^n with $w_0 = \langle s \rangle$, $w_n = \langle /s \rangle$. A path $t \in \langle q_I, q_F \rangle$ uniquely corresponds with a derivation (w_0^n, T) ; therefore $P(w_0^n) = \sum_T P(w_0^n, T) = \sum_{t \in \langle q_I, q_F \rangle} P(t)$. It is impractical, if not infeasible, to enumerate all t . However, by defining forward and inner proba-

bilities, the sum can be obtained implicitly. The following treatment is inspired by work on PCFG parsing by Stolcke (1995).

Definition 3 (forward probability) *Given a constrained PLCG network $\mathcal{G}_{w_0^n}$. The forward probability of a node q is*

$$\mu(q) \doteq \sum_{t \in \langle q_I, q \rangle} P(t)$$

Definition 4 (inner probability) *Given a constrained PLCG network $\mathcal{G}_{w_0^n}$. Consider a node $q = [{}_i X * {}_j \beta | \vec{h}]_Z$, $Z \neq W$. The inner probability of q is defined as*

$$\nu(q) \doteq \sum_{t \in \langle q^o, q \rangle} P(t)$$

If $Z = W$, $q^o = q$ and $\nu(q) \doteq 1$.

Using forward probabilities, the problem of finding $P(w_0^n)$ is reformulated as the problem of finding $\mu(q_F)$. If $\mathcal{G}_{w_0^n}$ were Markov, the edges (q, q') for a given q only depend on q itself and are independent of the path that led to q . Thus

$$\langle q_I, q' \rangle = \bigcup_{q: (q, q') \in \mathcal{G}_{w_0^n}^a} \langle q_I, q \rangle (q, q')$$

and since the path sets under the union operator are disjoint,

$$\mu(q') = \sum_{q: (q, q') \in \mathcal{G}_{w_0^n}^a} \mu(q) P((q, q')).$$

So $\mu(q_F)$ could be obtained as follows: first initialize $\mu(q_I) = 1$, then visit all $q' \in \mathcal{G}_{w_0^n}^a$ in topological order, computing $\mu(q')$ from all $\mu(q)$ and $P((q, q'))$ for $(q, q') \in \mathcal{G}_{w_0^n}^a$.

The PLCG network, however, is not Markov due to the attach constraint, which complicates the computation of forward probabilities. The recursion formula involves inner probabilities:

$$\mu(q') = \sum_{q, q''} \mu(q'') P(q^o | q'') \nu(q) P(q' | q) + \sum_q \mu(q) P(q' | q) \quad (17)$$

$$\nu(q') = \sum_{q, q''} \nu(q'') P(q^o | q'') \nu(q) P(q' | q) + \sum_q \nu(q) P(q' | q) \quad (18)$$

where — in both equations — the first sum is over all q for which $(q, q') \in \mathcal{G}_{w_0^n}^a$ and $(q, q') = \text{ATTACH}(q'')$, the second sum is over all q for which $(q, q') \in \mathcal{G}_{w_0^n}^a$ and (q, q') is a PROJECT move. A derivation of Eq. (17) and Eq. (18) is given in Appendix B.

4.3 Synchronous parsing algorithm

The essential task of a PLCG parser is computing the inner and forward probabilities of the nodes in the constrained network. Conceptually, the PLCG network is static for a given PLCG, and only the forward and inner probabilities have to be recomputed for each new input sentence. However in practice it is only feasible to allocate nodes dynamically the first time they get visited. A node that does not exist in main memory is assumed to have a zero or ‘negligible’ forward probability for the current input sentence.

The recursion formulas Eq. (17) and Eq. (18) for the computation of the inner and forward probabilities are now translated into three parsing subroutines: PROJECT-NODE, ATTACH-NODE and SHIFT-NODE. These routines take a source node and a move as input. If the target node does not exist yet in main memory, they allocate it and initialize the inner and forward probabilities to 0, otherwise they just update the probabilities according to Eq. (17) and Eq. (18). They return the target node. Assume a source node $q = [{}_i \mathbf{Y} * {}_j \alpha | \vec{h}]_X$. Then

- (1) If α is not empty, say $\alpha = Z\beta$, SHIFT-NODE(q, w) retrieves or initializes q' and an arc (q, q') , with $q' = [{}_j \mathbb{W} / w_{j+1} {}_{j+1} | \vec{g}]_{\mathbb{W}}$ where $\vec{g} = (Z, \mathbf{Y}, \mathbf{h}_2)$. $v(q')$ is initialized to 1, $\mu(q')$ is incremented with $\mu(q)p_s(w|q)$.
- (2) If α is empty, PROJECT-NODE(q, U, δ) retrieves or initializes q' and an arc (q, q') , with $q' = [{}_i \mathbf{X} {}_j \delta | \vec{h}]_U$. $\mu(q')$ is incremented with $f\mu(q)$ and $v(q')$ is incremented with $f v(q)$, with $f = p_p(U, \delta|q)(1 - p_a(\text{ATT}|q))$.
- (3) Suppose α is empty and a matching $q'' = [{}_k \mathbf{h}_2 * {}_i h_1 \beta | \vec{g}]_Z$, $\mathbf{g}_2 = \mathbf{h}_3$ is found for some $Z, k, \beta, g_1, \mathbf{g}_3$. Then ATTACH-NODE(q, q'') retrieves or initializes q' and an arc (q, q') with $q' = [{}_k \mathbf{U} * {}_j \beta | \vec{g}]_Z$. Furthermore, according to Eq. (17), $\mu(q')$ is incremented with $\mu(q'')p_s(q''|q'')v(q)p_a(\text{ATT}|q)$. According to Eq. (18), $v(q')$ is incremented with $v(q'')p_s(q''|q'')v(q)p_a(\text{ATT}|q)$.

In order to allow the PLCG-based language model to operate in a conditional mode (i.e. to return the probability of a next word given the words preceding it) we develop a word-synchronous parsing algorithm: partial analyses, constrained by a left context w_0^i , are extended in parallel observing the constraints imposed by w_{i+1} . Fig. 7 outlines the parsing algorithm that we implemented for our experiments.

4.4 Pruning

Dynamic programming allows to account for more parsing paths than beam search with the same amount of computer time and memory. Still it usually remains necessary to prune the constrained network quite severely. The following fairly simple pruning scheme proved to provide a satisfying time/performance trade-off in our

```

initialize  $q_I$ 
for each  $j = 1, 2, \dots, n$ :
  for each  $i = j - 1, j - 2, \dots, 0$ :
    for each  $q$  for which  $\text{start}(q) = i, \text{pos}(q) = j$ :
      for each  $U, \delta$  for which  $p_p(U, \delta|q) > 0$ :
         $q' \leftarrow \text{PROJECT-NODE}(q, U, \delta)$ 
        if  $q'$  is resolved:
          schedule  $q'$  for further PROJECT-NODE/ATTACH-NODE
        if  $q$  is attachable:
          for each matching node  $m$ :
             $q' \leftarrow \text{ATTACH-NODE}(q, m)$ 
            if  $q'$  is resolved:
              schedule  $q'$  for further PROJECT-NODE/ATTACH-NODE
  if  $j = n$ , return
  for each  $i = 0, 1, \dots, j - 1$ :
    for each  $q$  for which  $\text{start}(q) = i, \text{pos}(q) = j$ :
       $q' \leftarrow \text{SHIFT-NODE}(q, w_j)$ 

```

Fig. 7. Word-synchronous PLCG parsing algorithm

experiments:

- (1) Consider nodes $\mathcal{Q}_{ij} = \{q | \text{start}(q) = i, \text{pos}(q) = j\}$ and let $M_{ij} = \max_{q \in \mathcal{Q}_{ij}} \mu(q)$. Then $q \in \mathcal{Q}_{ij}$ is pruned if $\mu(q) \cdot \rho < M_{ij}$.
- (2) ρ is an *adapted beamwidth*. If N_{ij} is the number of nodes in \mathcal{Q}_{ij} , then $\rho = \rho^o N_{ij}^{-\sigma}$. ρ^o is the *maximum beam width*, σ is the *beam narrowing factor*.
- (3) ρ^o and σ are specified by the user. Typical values are $\rho^o = 10^4$, $\sigma = 0.3$.

The beam narrowing factor guards the parser against a combinatorial explosion of execution time in situations where too many nodes have forward probabilities close to each other.³

³ The beam narrowing factor can be considered a generalization of Roark's pruning method in (Roark, 2001), which would correspond with $\sigma = 3$, or in (Roark and Johnson, 1999), which would correspond with $\sigma = 1$. One important difference, though, is that we have one beam per group of nodes \mathcal{Q}_{ij} , while Roark's approach would rather correspond with pruning globally over $\bigcup_i \mathcal{Q}_{ij}$.

5 The PLCG-based language model

In this section, we derive a LM and a CLM from our word-synchronous PLCG parsing algorithm. Unlike Roark’s top-down parsing language model, the CLM probabilities are guaranteed to be normalized even in the face of pruning, since they can be reformulated as a weighted normalized sum of shift probabilities; in fact the lost probability mass from pruned terms is compensated by scaling up the probability mass of the remaining terms.

The CLM offers a few advantages over the LM. First of all, it can be combined with other CLMs, thereby providing a rough but usually effective tool for smoothing and mixing in other knowledge sources.

Another advantage is that a CLM can be applied earlier in the search process than a LM, reducing the input/output delay and potentially increasing search efficiency — although in practice a multi-pass approach is still preferred where a simple CLM (for instance, a trigram) is applied in a first pass and the advanced CLM or LM rescores the remaining hypotheses in the second pass. This is because the limited reduction of the search effort by advanced CLMs in the first pass would not compensate for the added computational complexity.

5.1 The PLCG-based language model

The PLCG-based LM straightforwardly returns the forward probability of the final node after parsing: for an input sentence W

$$\mu(q_F) = \sum_{t \in \langle q_I, q_F \rangle} P(t) = \sum_{W, T} P(W, T) = P(W) \quad (19)$$

since each path through the W -constrained network corresponds with one PLCG derivation (W, T) .

In practice, pruning will cause $\mu(q_F)$ to systematically underestimate the sentence probability. Alternatively, one can compose the sentence probability with conditional probabilities using the chain rule $P(W) = \sum_{i=1}^n P(w_i | w_0^{i-1})$ where $W = w_0^n$. The conditional probabilities are emitted by the CLM, explained in the following section.

5.2 The PLCG-based conditional language model

In this section, expressions for the prefix probabilities are derived, which will lead to an efficient calculation of conditional probabilities as the ratio of prefix proba-

bilities. Due to pruning, the prefix probabilities lose probability mass, which would lead to an underestimate of the conditional probabilities. However, we will show how a particular partitioning of the constrained PLCG graph allows the ratio of prefix probabilities to be rewritten as a weighted, normalized average of shift probabilities, which is guaranteed to be normalized in the face of pruning.

As a first step, the prefix probability $P(w_0^j)$ is expressed as the sum of probabilities of all partial paths that generate w_0^j and end just before shifting w_{j+1} . This sum is efficiently computed as a sum of partial sums, where each partial sum is available as the forward probability of a node where a subset of the considered partial paths arrive. The set of these nodes is called \mathcal{H}_j below.

One can then see that, in the absence of pruning, the total probability of all partial paths that end just after shifting w_j equals the total probability of all partial paths that end just before shifting w_{j+1} : the probability mass is preserved, since it is only re-distributed. In other words, $P(w_0^j)$ is alternatively expressed as a sum of forward probabilities of nodes where these partial paths arrive. The set of these nodes is called \mathcal{W}_j below.

The following lemma allows a more formal treatment of the above intuitive reasoning. (The proof is trivial and therefore omitted.)

Lemma 5 *Assume the following two subsets of nodes of the constrained PLCG graph $\mathcal{G}_{w_0^n}$:*

$$\begin{aligned}\mathcal{H}_j &\doteq \{q \in \mathcal{G}_{w_0^n} \mid \text{pos}(q) = j, q \text{ is unresolved}\} \\ \mathcal{W}_j &\doteq \{q \in \mathcal{G}_{w_0^n} \mid \text{pos}(q) = j, \text{cat}(q) = W\}\end{aligned}$$

The sets $L_q = \{t \in \langle q_I, q_F \rangle : q \in t\}$, $q \in \mathcal{H}_j$, form a partition of $\langle q_I, q_F \rangle$ since each path in $\langle q_I, q_F \rangle$ traverses exactly one $q \in \mathcal{H}_j$. In the same way the sets $L_q = \{t \in \langle q_I, q_F \rangle : q \in t\}$, $q \in \mathcal{W}_j$, form a partition of $\langle q_I, q_F \rangle$.

The forward probability of a node $q \in \mathcal{H}_j$ can be interpreted as a real probability. Each path in $\langle q_I, q \rangle$ uniquely corresponds with one way of generating w_0^j and arriving in q . In other words $\mu(q)$, being the sum of probabilities of these paths, is the joint probability of w_0^j and q :

$$\mu(q) = P(w_0^j, q) \tag{20}$$

With a similar reasoning on \mathcal{W}_j and due to Lemma 5 we now obtain two simple formulas to compute *prefix probabilities*:

$$P(w_0^j) = \sum_{q \in \mathcal{H}_j} \mu(q) = \sum_{q \in \mathcal{W}_j} \mu(q) \tag{21}$$

and conditional probabilities:

$$P(w_{j+1}|w_0^j) = \frac{P(w_0^{j+1})}{P(w_0^j)} = \frac{\sum_{q \in \mathcal{H}_{j+1}} \mu(q)}{\sum_{q \in \mathcal{H}_j} \mu(q)} = \frac{\sum_{q \in \mathcal{H}_j} \mu(q) P_s(w_j|q)}{\sum_{q \in \mathcal{H}_j} \mu(q)} \quad (22)$$

So we come to the intuitively appealing conclusion that $P(w_{j+1}|w_0^j)$ is a normalized and weighted average of the shift probabilities $p_s(w_{j+1}|q)$ with weights $\mu(q)$.

Contrary to the computed whole-sentence probability (Eq. (19)) and the prefix probabilities (Eq. (21)), Eq. (22) remains a proper probability under path pruning. Another important advantage is that the conditional probabilities can be emitted on-line during parsing, namely at the end of the main loop (Fig. 7).

5.3 Maximum-likelihood training

In Sec. 3.4.1, we proposed to initialize the move probabilities of a PLCG-based LM with smoothed relative observation frequencies in a treebank. This initialization approximatively optimizes the likelihood of the *treebank*. For a language model however, it makes more sense to optimize the likelihood of the *plain text*, i.e. the treebank minus the trees.

We developed a full Expectation-Maximization(EM) training procedure for the PLCG-based LM, involving the efficient computation of expected move frequencies from the constrained PLCG graph. For the sake of brevity, we omit the detailed derivations of the reestimation formulas here; they are described in (Van Uytsel et al., 2001; Van Uytsel and Van Compernelle, 2003).

6 Experiments

Experiments were conducted in two stages. In the first stage, small models were trained on the Penn Treebank corpus. We used these models to test our software, find appropriate submodel parameterizations and compare test set perplexities with other competing models. In the second stage we worked towards a more realistic large-vocabulary speech recognition setting: large models were trained and reestimated on the BLLIP WSJ corpus, and tested on their performance in n-best list rescoring.

6.1 Experiments with Penn Treebank models

6.1.1 Modeling

6.1.1.1 Data The Penn Treebank (Marcus et al., 1993) (PTB) is a collection of text available from the LDC including material from the ATIS domain, the Wall Street Journal (WSJ), the Brown and Switchboard corpus. The text is annotated with hand-corrected labeled parse trees, part-of-speech tags, function labels (such as subject, location, etc.), anaphora and disfluency markers (the latter for Switchboard only). A list of the labels and their meanings can be found in (Marcus et al., 1993).

In our experiments, we only used the WSJ portion of the Penn Treebank (version 3) and only kept the parse trees with the syntactic constituent labels and the pre-terminals (parts-of-speech) from the annotation. Sections 00–20 were used for training, while sections 21–22 were reserved for testing during development and sections 23–24 were used for final testing. The train set contains 42,073 sentences worth of 1,046,082 running tokens (after tokenization, preserving punctuation). The development set contains 3,371 sentences (83,524 tokens) and the test set contains 3,759 sentences (93,185 tokens).

The labeled parse trees went through a number of preprocessing steps, involving headword annotation and binarization. We hereby attempted to match the preprocessing used in Chelba’s experiments as closely as possible, in order to allow a reasonable comparison of performance results with C&J. All the details can be found in (Chelba, 2000).

6.1.1.2 Parameterization From any node $q = [{}_i \mathbf{Y} * {}_j \beta | \vec{h}]_X$ in the network, conditional move probabilities $P(\text{SHIFT}(w)|q)$, $P(\text{PROJECT}(U, \delta)|q)$ and $P(\text{ATTACH}|q)$ have to be given by the language model. The condition q has 10 categorical attributes (given that the considered parse trees are binary). Obviously, this is more than can possibly be used in estimating the conditional probabilities from data in practical situations. We therefore need to find a good model parameterization, i.e. determine which attributes of q are informative enough, and in which order of significance. The latter is necessary because we apply back-off in order to account for data sparsity.

Assuming that $q = [{}_i \mathbf{X} * {}_j \beta | \vec{h}]_{\mathbf{Z}}$, $\mathbf{X} = X/x$, $\mathbf{Z} = Z/z$, and $\vec{h} = (G, L_1/\ell_1, L_2/\ell_2)$, we

have chosen the following parameterizations:

$$P(\text{SHIFT}(w)|q) \simeq \begin{cases} p_s(w|\beta, x, \ell_1) & \text{if } \beta \neq \varepsilon \\ 0 & \text{otherwise} \end{cases} \quad (23)$$

$$P(\text{PROJECT}(U, \delta)|q) \simeq \begin{cases} p_p(U, \delta|G, Z, X, z) & \text{if } \beta = \varepsilon \text{ and } Z \neq W \\ p_t(U, \delta|z, G, L_1) & \text{if } \beta = \varepsilon \text{ and } Z = W \\ 0 & \text{otherwise} \end{cases} \quad (24)$$

$$P(\text{ATTACH}|q) \simeq \begin{cases} p_a(\text{ATTACH}|G, Z, X, z) & \text{if } \beta = \varepsilon \\ 0 & \text{otherwise} \end{cases} \quad (25)$$

The conditioning attributes are ordered from most to least significant: for instance, $p_s(w|\beta, x, \ell_1)$ is smoothed with a back-off distribution $p'_s(w|\beta, x)$, which is smoothed with a back-off distribution $p''_s(w|\beta)$, etc. Note that we employ a different parameterization for projections from W constituents (essentially, part-of-speech tagging); we call p_t the *tagger* submodel.

If p_p and p_a have the same parameterization, as in our case, p_p and p_a can be combined conveniently in one model p_{pa} :

$$p_{pa}(\text{ATT}|G, Z, X, z) = p_a(\text{ATTACH}|G, Z, X, z) \quad (26)$$

$$p_{pa}(U, \delta|G, Z, X, z) = p_p(U, \delta|G, Z, X, z)(1 - p_{pa}(\text{ATT}|G, Z, X, z)), \quad (27)$$

which considers ATT just as a special (U, δ) .

Parameterizations Eq. (23), Eq. (24) and Eq. (25) were determined manually by optimizing the *conditional perplexities* (CPPL) on the development set (sections 21–22) obtained with an initial model trained on sections 00–20 with a certain parameterization. The concept of CPPL was introduced by (Chelba, 2000). For the shift model $p_s(w|\beta, x, \ell_1)$, for instance, it can be defined as

$$\text{CPPL} = \exp \frac{\sum_{w, \beta, x, \ell_1} c_D(\text{SHIFT}(w), \beta, x, \ell_1) \ln p_s(w|\beta, x, \ell_1)}{\sum_{w, \beta, x, \ell_1} c_D(\text{SHIFT}(w), \beta, x, \ell_1)}, \quad (28)$$

where D is a measurement corpus of parse trees, decomposed into LC parser moves, and $c_D(\text{SHIFT}(w), \beta, x, \ell_1)$ is the frequency in D of the $\text{SHIFT}(w)$ move from any node $[_i X / x^* _j \beta | G, L_1 / \ell_1, L_2 / \ell_2]_{\mathbf{Z}}$ for some $\mathbf{Z}, i, j, X, G, L_1, L_2, \ell_2$.

The CPPLs of the selected parameterizations on the development set are listed in Table 2. One way of interpreting the CPPL of p_s is as a lower bound for the perplexity that can be achieved by the PLCG-based language model. It would be the perplexity of a language model that always builds correct parse trees with probability 1.

In an attempt to automate the parameterization process, we have also considered the following greedy optimization procedure:

Table 2

Conditional perplexities of selected submodel parameterizations on the PTB development set.

submodel	smoothing	CPPL
$p_s(w \beta, x, \ell_1)$	Good-Turing	35.2
	Kneser-Ney	31.1
$p_{pa}(U, \delta G, Z, X, z)$	Good-Turing	1.88
	Kneser-Ney	1.80
$p_t(U, \delta z, G, L_1)$	Good-Turing	1.20
	Kneser-Ney	1.15

Table 3

Evaluated submodel smoothing techniques.

abbreviation	discounting	back-off strategy	reference
GT	Good-Turing	non-linear back-off	(Katz, 1987)
KN	absolute	linear Kneser-Ney	(Chen and Goodman, 1998)
DI	linear, context-dep.	linear deleted interp.	(Jelinek and Mercer, 1980)

- (1) Start with an empty context.
- (2) For each parameter that is not yet in the context, evaluate the decrement of the CPPL on the development set by adding it as the least significant item to the context.
- (3) If the largest decrement of the CPPL is larger than a preset threshold value, add the corresponding parameter to the context as the least significant item. Else terminate.
- (4) Go back to step 2.

This procedure was repeated for each submodel with Kneser-Ney smoothing and Good-Turing discounting. The results were identical to the manually selected parameterizations, except for the project model, for which in both smoothing schemes a parameterization $p_p(U, \delta|Z, G, z, X)$ was found, which performed slightly worse than Eq. (24).

6.1.1.3 Initial models Once submodel parameterizations are fixed, each tree-annotated sentence from the train set is decomposed in its elementary LC derivation steps. Each step corresponds with an n -gram event. p_s , p_{pa} and p_t are initialized using conventional n -gram language modeling techniques. We compared three smoothing techniques, as listed in Table 3.

Although the designer may choose a *different* smoothing technique for each submodel, we did not explore this additional degree of freedom in our experiments.

Table 4

Influence of pruning parameters on PPL and execution time. Table entries are in the format PPL/PPLi/Time, where PPLi is obtained after interpolation with the baseline 3-gram model, and time is total execution time on a PIII/930MHz PC. Both the baseline 3-gram model and the PLCG-based LM are trained with KN smoothing and tested on the verbalized punctuation data.

	$\rho^o = 10^3$	$\rho^o = 10^{3.5}$	$\rho^o = 10^4$	$\rho^o = 10^{4.5}$
$\sigma = .7$	130.5/100.7/0:13	113.2/97.5/0:20	106.4/96.0/0:35	104.9/95.5/1:06
$\sigma = .5$	112.4/97.6/0:21	105.8/96.0/0:37	103.4/95.5/1:18	102.6/95.3/3:02
$\sigma = .3$	106.0/96.2/0:37	103.2/95.5/1:21	102.4/95.3/3:23	102.5/95.3/9:25

6.1.1.4 Baseline model We have trained word-based 3-gram models on the running text of the PTB train set using the smoothing techniques listed in Table 3 for comparison with the corresponding PLCG-based LMs. The 3-gram models are also used for interpolation with the PLCG-based LM.

6.1.2 Measurements

6.1.2.1 Data As the measurement set, we selected sections 23 and 24 of the Penn Treebank. This test set is very common in large-scale stochastic parsing literature. We have prepared a ‘verbalized punctuation’ (vp) and a ‘non-verbalized punctuation’ (nvp) version, to be used with their corresponding models.

6.1.2.2 Influence of pruning The pruning parameters ρ^o and σ (cf. Sec. 4.4) have an important influence on the runtime behavior of the PLCG-based LM. Small ρ^o and large σ lead to fast execution but inaccurate evaluation.

In Table 4, test set perplexities and execution times measured with varying pruning settings are collected in a matrix. We interpret the measured PPL at very loose pruning settings as the ‘real’ PPL, and the difference with the real PPL at practical pruning settings as the inaccuracy of the evaluation. There seems to be a rather strong dependence between ρ^o and σ , given a target PPL/time trade-off. In the next experiments, we fixed $\rho^o = 10^{3.5}$ and $\sigma = .5$.

6.1.2.3 Influence of smoothing Table 5 reports test set PPLs of differently smoothed PLCG-based LMs (cf. Table 3). The PLCG-based LMs used in these experiments were not reestimated.

One notes that the DI and GT smoothed models do not significantly differ in performance; KN smoothing, however, outperforms GT and DI smoothing with a significant PPL reduction of roughly 10%.

Table 5

Test set PPL of differently smoothed PLCG-based LMs. The ‘3g+PLCG’ rows are obtained by interpolating the PLCG-based LM interpolated with the baseline 3-gram LM where the interpolation weight of the baseline 3-gram LM is .4.

PTB 23–24 nvp	DI	GT	KN
baseline 3-gram	194	191	173
PLCG-based LM	175	174	154
3g+PLCG	164	161	145
PTB 23–24 vp	DI	GT	KN
baseline 3-gram	129	126	114
PLCG-based LM	118	119	106
3g+PLCG	108	107	96

We also conclude from the vp experiment series that the unreeestimated PLCG-based LM improves on the 3-gram PPL by roughly 6%, and by 15% when interpolated with the 3-gram. For the nvp series, the measured PPL improvements are 10% and 15%, respectively.

6.1.2.4 Influence of reestimation Reestimating the model on the same PTB train corpus was unsuccessful. Our results are collected in Table 6. The train set PPL drops and converges after 1 iteration. The development and test set PPLs, however, increase significantly at the first iteration with GT smoothing. The test PPLs obtained with KN smoothing seem rather random; in any case, they do not justify the additional training effort.

We do not know the exact reason of the failing reestimation. A possible cause is that KN and GT smoothing are actually non-continuous techniques; using them here for counteracting EM over-training is not theoretically justified. DI smoothing does not have that problem, but it remains to be seen whether DI smoothed reestimation can make any difference, given the inferior performance of DI smoothing in perplexity measurements.

6.1.2.5 Comparison with other syntax-based LMs Our standard choice of the train and test set enabled us to compare the PPL performance of the PLCG-based LM quantitatively with other recent grammar-based language models.

Results extracted from publications and from our own experiments are collected in Table 7. The table separates results obtained with different smoothing techniques in different columns, since smoothing affects PPL considerably, as noted above. In addition, we recalculated test set PPLs with <unk> probabilities *included*, to

Table 6

Influence of reestimation on PPL.

PTB vp, KN	initial	after 1 it.	after 2 it.
Train set PPL	26.1	25.4	25.2
Dev set PPL	103.4	100.6	102.3
Test set PPL	106.0	107.1	109.0
PTB vp, GT	initial	after 1 it.	after 2 it.
Train set PPL	31.7	19.3	19.3
Dev set PPL	114.5	134.3	130.2
Test set PPL	119.0	144.2	140.1
PTB nvp, KN	initial	after 1 it.	after 2 it.
Train set PPL	32.5	31.9	31.9
Dev set PPL	151.5	152.4	156.0
Test set PPL	154.0	162.0	166.1
PTB nvp, GT		after 1 it.	after 2 it.
Train set PPL	40.8	22.3	23.3
Dev set PPL	169.4	201.7	198.0
Test set PPL	174.0	217.0	212.5

enable comparison with (Chelba, 2000; Roark, 2001; Charniak, 2001; Kim et al., 2001). (We believe, though, that the PPL excluding <unk> probabilities, as reported elsewhere in this article, is more predictive for speech recognition performance.)

The lowest PPL (including <unk>) in the table is 126. It is obtained with a KN smoothed PLCG-based LM, interpolated with the baseline word-based trigram. Chelba’s LM with KN smoothed probabilities (Kim et al., 2001) comes close to our best result (130 versus 126). This model is reestimated, while the PLCG-based LM is not. In other words, building a PLCG-based LM requires a tiny fraction of the time needed for building a C&J model that has comparable performance.

Charniak’s model (Charniak, 2001) reaches the same performance with DI smoothed probability distributions; it is an open question whether KN smoothing may improve Charniak’s model further. An important drawback of Charniak’s model is that it cannot be interpolated with other models at the word level.

Roark’s top-down parsing LM (Roark, 2001) shows some potential too; again, improved smoothing may reduce the PPL of 137, but we did not yet find such experiment results in literature. Moreover, our PPL of 126 was measured at a rather tight pruning setting; at equal execution speeds on comparable computers (30 words/s),

Table 7

Comparing PPLs (on PTB 23–24 nvp) obtained with other grammar-based LMs. The C&J models are reestimated with 3 EM iterations. The PLCG-based LM is not reestimated.

	DI		GT		KN	
PPL without <unk>	PPL	PPLi	PPL	PPLi	PPL	PPLi
word-based 3-gram	194	194	191	191	173	173
PLCG-based LM	175 ^a	164 ^a	174	161	154	145
C&J LM	187 ^b	174 ^a				
PPL with <unk>	PPL	PPLi	PPL	PPLi	PPL	PPLi
word-based 3-gram	167	167	166	166	156	156
PLCG-based LM	151	139	150	138	133	126
C&J LM	153 ^c	147 ^c			141 ^d	130 ^d
Roark LM	152 ^e	137 ^e				
Charniak LM	130 ^f	126 ^f				

^a experiments run by F. Van Aelten and K. Daneels at L&H; ^b obtained with a reimplementation (Van Aelten and Hogenhout, 2000) of (Chelba, 2000); ^c as reported in (Chelba, 2000, p. 49); ^d as reported in (Kim et al., 2001); ^e as reported in (Roark, 2001, p. 270); ^f as reported in (Charniak, 2001).

Roark’s model marked a PPL of 141.

6.2 Experiments with BLLIP-WSJ models

In a second series of experiments, we evaluate the PLCG-based LM in a more realistic dictation setting. Models are trained on a large corpus and employed for rescoring transcription hypotheses in a recognition task.

6.2.1 Modeling

The models are trained on the BLLIP-WSJ corpus plus sections 0–20 of the PTB. The BLLIP-WSJ corpus is the ACL/DCI Wall Street Journal ’87–’89 corpus that was machine-parsed by the BLLIP lab at Brown University and distributed by the LDC (Charniak, 2000). BLLIP-WSJ has an annotation style and tag set that is very similar to PTB’s, but is about 35 times larger. On the other hand, it contains more parsing errors.

Care was taken that the speech recognition test set was excluded from the training data. All the submodels and the baseline word-based 3-grams were slimmed down by omitting all maximum-order events that appeared less than twice in the train

Table 8

Word error rates on eval92 obtained with GT smoothed models. PLCG0 is not reestimated, PLCG2 is reestimated twice. '+' indicates linear interpolation.

model	WER
word 3-gram	7.98
PLCG0 + word 3-gram	7.26
PLCG2 + word 3-gram	7.03

corpus.

The submodels of the PLCG-based LM were parameterized in the same way as in the PTB experiment series. GT smoothing was used for initialization, yielding model PLCG0, as well as for two EM iterations, yielding models PLCG1 and PLCG2, respectively.

Additionally, we trained a GT smoothed word-based trigram on the BLLIP-WSJ. This model differs from the standard WSJ-trained trigrams in the tokenization (e.g., don't is replaced with do n't and numbers are replaced with N). Also, the BLLIP-WSJ does not contain all the data from the WSJ corpus because sentences that could not be machine-parsed in reasonable time were left out.

In cooperative work done at L&H, a DI smoothed class-based 4-gram model was trained with automatically generated word classes. This model was used to assess complementarity of the word class model with other grammar-based models.

6.2.2 Word error rate

The DARPA WSJ November 1992 LVCSR test suite (20k open vocabulary, verbalized punctuation) was used for testing recognition performance of the PLCG-based LM. We generated 100-best lists for both the evaluation (eval92) and development (dev92) test sets using a mainstream HMM-based Viterbi decoder equipped with the standard word trigram LM. The 100-best lists were first preprocessed to match the tokenization of the BLLIP-WSJ models. These lists were then rescored with the language models under scrutiny. The dev92 set was used for finding optimal model interpolation weights.

In Table 8 we have collected WER results with GT smoothed models. The un-reestimated PLCG-based LM yields a relative improvement of 9% with respect to the word 3-gram. We observed a small but consistent improvement by EM reestimation: the reestimated PLCG-based LM yields a relative improvement of 12% below the word 3-gram baseline WER. (This corrects the results reported in (Van Uytsel et al., 2001), which were affected by bugs in our rescoring code.)

Comparative experiments using DI smoothed models were done at the L&H lab by

Table 9

Word error rates on eval92 obtained with DI smoothed models. '+' indicates linear interpolation. The PLCG0 model is not reestimated, the C&J model is reestimated with 3 EM iterations. Measurements by Filip Van Aelten and Kristin Daneels at L&H.

model	WER
word 3-gram	7.88
word 3-gram + class 4-gram	7.37
C&J	7.47
C&J + word 3-gram	7.31
C&J + word 3-gram + class 4-gram	7.08
PLCG0	7.08
PLCG0 + word 3-gram	7.06
PLCG0 + word 3-gram + class 4-gram	6.91

Filip Van Aelten and Kristin Daneels. Their results are summarized in Table 9. It was observed that the PLCG model successfully complements a word 3-gram and a class 4-gram; the C&J model does so too, but its performance is slightly worse than the PLCG model's, although the significance of the difference is disputable.

7 Discussion

In this article we developed the PLCG-based LM from an efficient synchronous dynamic-programming PLCG parser using rich context nodes. The model emits next word probabilities in one single left-to-right pass. It is initialized on a treebank and can be further optimized on the text (surface) part of the same data or retrained on other unannotated text material. The latter possibility opens the way to adapting the language model to another domain, a perspective that sounds attractive because the knowledge contained within the PLCG-based language model is of a syntactic nature and therefore expected to be more generic than other common language model formalisms.

We found that the PLCG-based LM is a competitive alternative among the class of syntax-based LMs. Test set perplexities and word error rates compare favorably with, for instance, (Chelba, 2000), (Roark, 2001) and (Charniak, 2001). From the comparison of execution times, it is believed that the PLCG-based LM is more efficient than the other cited models. However, EM reestimation did not produce convincing improvements; but it was observed that the un-reestimated PLCG-based LM performed at least as well as the reestimated C&J model, while building the first model requires only a tiny fraction of the time needed to train the latter.

We believe that a great part of the efficiency is gained by representing the search space as a minimal network instead of a search tree, as in (Chelba and Jelinek, 1999) and (Roark, 2001). Given the same computational resources, more probabilistic analyses can be accounted for using the dynamic programming technique. A disadvantage of this, however, is that the parser is less flexible at extracting the relevant conditioning information from a partial analysis. We solved that problem by extending the nodes with all features that we expected to be informative in selecting the next parse move.

A final suggestion for future work is about the detail of the underlying grammar. Current syntax-based language models rely on an overly simplistic version of phrase structure grammar, namely the one that is most readily extracted from the Penn Treebank and its relatives. A greater degree of generalization can be obtained by processing syntactic features (such as tense, gender, number, finiteness), for instance in a unification-based style, instead of only one category label. The integration of a morpho-syntactic analysis stage within the parsing system could introduce those features into the syntactic analysis of the whole sentence; the linguistically sound treatment of previously unseen wordforms would come as a welcome side-effect.

Acknowledgements

We would like to thank the anonymous reviewers, Tom Laureys, Kris Demuynck and Patrick Wambacq for their comments on draft versions of this paper. We are indebted to Filip Van Aelten, who suggested to use context conditioning for left corner parsing and did part of the experiments, especially those involving deleted interpolation smoothing.

References

- Bod, R., 2000. Combining semantic and syntactic structure for language modeling. In: Proc. International Conference on Spoken Language Processing 2000. Vol. 3. Beijing, China, pp. 106–109.
- Charniak, E., 2000. A maximum-entropy inspired parser. In: Proc. 1st Meeting of the North American Chapter of the Association for Computational Linguistics. pp. 132–139.
- Charniak, E., 2001. Immediate-head parsing for language models. In: Proc. 39th Annual Meeting of the Association for Computational Linguistics (ACL'01).
- Chelba, C., 2000. Exploiting syntactic structure for natural language modeling. Ph.D. thesis, Johns Hopkins University.
- Chelba, C., Jelinek, F., 1999. Recognition performance of a structured language model. In: Proc. European Conference on Speech Communication and Technology 1999. Vol. 1. pp. 1567–1570.
- Chen, S., Goodman, J., August 1998. An empirical study of smoothing techniques for language modeling. Technical Report TR–10–98, Harvard University.
- Goodman, J. T., 2001. A bit of progress in language modeling. *Computer Speech and Language* 15 (4), 403–434.
- Griffiths, T. V., Petrick, S. R., 1965. On the relative efficiencies of context-free grammar recognizers. *Communications of the ACM* 8 (5), 289–300.
- Hopcroft, J. E., Motwani, R., Ullman, J., 2001. *Introduction to Automata Theory, Languages, and Computation*, 2nd Edition. Addison-Wesley.
- Huang, X., Acero, A., Hon, H.-W., 2001. *Spoken Language Processing*. URL file:///users/spraak/spch/tex/elib/huang:book01
- Jelinek, F., 1997. *Statistical Methods for Speech Recognition*. The MIT Press, Cambridge, MA.
- Jelinek, F., Chelba, C., 1999. Putting language into language modeling. In: Proc. European Conference on Speech Communication and Technology 1999. Vol. 1. pp. KN–1–6.
- Jelinek, F., Mercer, R. L., 1980. Interpolated estimation of Markov source parameters from sparse data. In: Geltsema, E. S., Kanal, L. N. (Eds.), *Pattern Recognition in Practice*. North Holland, Amsterdam.
- Johnson, M., 1998. Finite state approximation of constraint-based grammars using left-corner grammar transforms. In: Proc. 36th Annual Meeting of the Association for Computational Linguistics (COLING/ACL'98). pp. 619–623.
- Katz, S. M., Mar. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing* 35, 400–401.
- Kay, M., 1989. Head driven parsing. In: Proc. 1st International Workshop on Parsing Technologies. Pittsburgh, PA, USA.
- Kim, W., Khudanpur, S., Wu, J., 2001. Smoothing issues in the structured language model. In: Proc. European Conference on Speech Communication and Technology 2001. Aalborg, Danmark, pp. 717–720.
- Lang, B., 1974. Deterministic techniques for efficient non-deterministic parsers. In:

- Proc. 2nd Coll. Automata, Languages and Programming. Springer, Saarbrücken, pp. 255–269.
- Leermakers, R., 1992. A recursive ascent Earley parser. *Information Processing Letters* 41 (2), 87–91.
- Manning, C. D., Carpenter, B., 1997. Probabilistic parsing using left corner language models. In: Proc. 5th International Workshop on Parsing Technologies. pp. 147–158.
- Marcus, M. P., Santorini, B., Marcinkiewicz, M. A., 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics* 19 (2), 313–330.
- Matsumoto, Y., Kiyono, M., Tanaka, H., 1983. BUP: A bottom-up parser embedded in Prolog. *New Generation Computing* 1 (2), 145–158.
- Moore, R. C., 2000. Improved left-corner chart parsing for large context-free grammars. In: Proc. 6th International Workshop on Parsing Technologies. pp. 171–182.
- Nederhof, M.-J., 1993. Generalized left-corner parsing. In: Proc. 6th Conference of the European Chapter of the Association for Computational Linguistics (EACL'93). pp. 305–314.
- Roark, B., 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics* 27 (2), 249–276.
- Roark, B., Johnson, M., 1999. Efficient probabilistic top-down and left-corner parsing. In: Proc. 37th Annual Meeting of the Association for Computational Linguistics (ACL'99). pp. 421–428.
- Rosenkrantz, D. J., Lewis II, P. M., 1970. Deterministic left corner parsing (extended abstract). In: IEEE Conference Record of the 11th Annual Symposium on Switching and Automata Theory. pp. 139–152.
- Stolcke, A., 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics* 21 (2), 165–201.
- Stolcke, A., Chelba, C., Engle, D., Jimenez, V., Mangu, L., Printz, H., Ristad, E., Rosenfeld, R., Wu, D., Jelinek, F., Khudanpur, S., 1997. WS96 project report on dependency language modeling.
URL citeseer.nj.nec.com/article/stolcke97dependency.html
- Van Aelten, F., Hogenhout, M., 2000. Inside-outside reestimation of Chelba-Jelinek models. Technical Report L&H-SR-00-027, Lernout & Hauspie, Wemmel, Belgium.
- van Noord, G., 1997. An efficient implementation of the head-corner parser. *Computational Linguistics* 23 (3), 425–456.
- Van Uytsel, D. H., Van Compernelle, D., Jan. 2003. Language modeling with context-sensitive probabilistic left corner parsing. Internal Report PSI-SPCH-03-1, K.U.Leuven/ESAT/PSI.
URL <file:///users/spraak/spch/tex/papers/donghoon/ir03.1/ir.pdf>
- Van Uytsel, D. H., Van Compernelle, D., Wambacq, P., Dec. 2001. Maximum-likelihood training of the PLCG-based language model. In: Proc. IEEE Automatic Speech Recognition and Understanding Workshop 2001. Madonna di

- Campiglio, Italy, 4 pages. ISBN 0-7803-7343-X.
- Wirén, M., 1987. A comparison of rule-invocation strategies in context-free chart parsing. In: Proc. 3rd Conference of the European Chapter of the Association for Computational Linguistics (EACL'87). pp. 226–233.
- Young, S., Sep. 1996. A review of large-vocabulary continuous-speech recognition. IEEE Signal Processing Magazine , 45–57.

A Proof of Lemma 1 and Lemma 2

Lemma 6 *Let \mathcal{G} be a PLCG network and $q_1, q_2 \in \mathcal{G}^n$. Then $q_1 \prec q_2$ implies $\text{pos}(q_1) \leq \text{pos}(q_2)$. If q_1 and q_2 are connected by a path, then $\text{pos}(q_1) < \text{pos}(q_2)$ implies $q_1 \prec q_2$.*

PROOF. A SHIFT increments $\text{pos}(\cdot)$ with 1, while PROJECT and ATTACH() moves leave $\text{pos}(\cdot)$ unaltered. Thus if a path $t \in \langle q_1, q_2 \rangle$ contains n SHIFT moves, then $\text{pos}(q_2) = \text{pos}(q_1) + n \geq \text{pos}(q_1)$. Conversely, if q_1 and q_2 are connected by a path and $\text{pos}(q_1) < \text{pos}(q_2)$, then $q_1 \prec q_2$ or $q_2 \prec q_1$. Assume $q_2 \prec q_1$. Then $\text{pos}(q_2) \leq \text{pos}(q_1)$ which is in contradiction with the given. \square

Lemma 7 (Lemma 1) *Let q_1 and q_2 be nodes in a PLCG network \mathcal{G} where $q_1 \prec q_2$, $t \in \langle q_1, q_2 \rangle$ and $(q, q') = \text{ATTACH}(q'')$ be a move in t . If $\text{ATTACH}(s)$ is a move in a path $u \in \langle q'', q \rangle$ then $q'' \prec s$.*

PROOF. The first move (q'', q^o) of u is a SHIFT move, since q'' is incomplete. So $\text{start}(q^o) = \text{pos}(q'')$. Now suppose (q_i, q_j) is a move on u . If (q_i, q_j) is a SHIFT move, then $\text{start}(q_i) < \text{start}(q_j)$. If (q_i, q_j) is a PROJECT move, then $\text{start}(q_i) = \text{start}(q_j)$. If (q_i, q_j) is an ATTACH() move, first consider the case that it is the first ATTACH() move in u , so that $\text{start}(r) \geq \text{pos}(q'')$ for all $r \in u$ for which $q'' \prec r \preceq q_i$. Assume $(q_i, q_j) = \text{ATTACH}(s), s \neq q''$. Then $\text{pos}(q'') \leq \text{pos}(s) < \text{pos}(q_i)$, since $\text{start}(q_i) = \text{pos}(s)$, $\text{start}(q_i) < \text{pos}(q_i)$ and $\text{start}(q_i) \geq \text{pos}(q'')$. By Lemma 6, $q'' \prec s \prec q_i$, which was to be proven. In the other case that (q_i, q_j) is not the first ATTACH() move, repeat the above reasoning on $\langle q_k, q_i \rangle$ instead of $\langle q'', q \rangle$. \square

Lemma 8 (Lemma 2) *If $q = [{}_i \mathbf{X} * {}_j \beta | \vec{h}]_Z$, then on every w_0^i -constrained path in $\langle q_I, q \rangle$ there is a node $q^o = [{}_i w_{i+1} | \vec{h}]_W$.*

PROOF. Any w_0^i -constrained path must contain a node $r = [{}_i w_{i+1} | \vec{g}]_W$. If the given q is on the same path, then r and q can be connected by PROJECT moves only, since SHIFT and ATTACH moves do not preserve the $\text{start}(\cdot)$ property. PROJECT moves preserve the local tree context. Hence $\vec{h} = \vec{g}$ and thus $r = q^o$. \square

B Derivation of recursion formulas Eq. (17) and Eq. (18) for forward and inner probabilities

Let a constrained PLCG network $\mathcal{G}_{w_0^n}$ be given. Suppose we want to compute $\mu(q')$ and $v(q')$ when $\mu(q)$ and $v(q)$ is already known for each $q : q \prec q'$.

A move $(q, q') = \text{ATTACH}(q'')$ does not contribute a term $\mu(q)P(q'|q)$ to $\mu(q')$, since not all paths $\langle q_I, q \rangle$ extend to paths in $\langle q_I, q' \rangle$. Using *inner probabilities* we can ensure that only paths $\{t \in \langle q_I, q \rangle : q'' \in t\}$ contribute to $\mu(q')$.

Let q' be the node for which we want to compute $\mu(q')$. Then

$$\langle q_I, q' \rangle = \left\{ \bigcup_{q, q''} \langle q_I, q'' \rangle (q'', q^o) \langle q^o, q \rangle (q, q') \right\} \cup \left\{ \bigcup_q \langle q_I, q \rangle (q, q') \right\} \quad (\text{B.1})$$

where the first union is over q, q'' for which $(q, q') = \text{ATTACH}(q'')$ for some q'' and the second union is over q for which (q, q') is not an ATTACH move. The above expansion is justified by Lemma 1 which states that any path in $\langle q_I, q'' \rangle$ concatenates with $(q'', q^o) \langle q^o, q \rangle$ to a valid path in $\langle q_I, q \rangle$, given that $(q, q') = \text{ATTACH}(q'')$: $\langle q_I, q'' \rangle (q'', q^o) \langle q^o, q \rangle$ exactly covers the paths of $\langle q_I, q \rangle$ that visit q'' . In terms of probabilities:

$$\mu(q') = \sum_{q, q''} \left\{ \mu(q'')P(q^o|q'')P(q'|q) \sum_{t \in \langle q^o, q \rangle} P(t) \right\} + \sum_q \mu(q)P(q'|q) \quad (\text{B.2})$$

where the first sum is over all q for which $(q, q') \in \mathcal{G}_{w_0^a}$ and $(q, q') = \text{ATTACH}(q'')$, the second sum is over all q for which $(q, q') \in \mathcal{G}_{w_0^a}$ and (q, q') is a PROJECT move.

The sum within the first summand is an inner probability. Hence Eq. (17) follows:

$$\mu(q') = \sum_{q, q''} \mu(q'')P(q^o|q'')\nu(q)P(q'|q) + \sum_q \mu(q)P(q'|q) \quad (\text{B.3})$$

The derivation of a recursive formula for the inner probability is analogous with the one above, where the node q^o substitutes q_I . One obtains Eq. (18):

$$\nu(q') = \sum_{q, q''} \nu(q'')P(q^o|q'')\nu(q)P(q'|q) + \sum_q \nu(q)P(q'|q) \quad (\text{B.4})$$

for a node q' .