Deterministic Parsing using PCFGs

Mark-Jan Nederhof and **Martin McCaffery**
School of Computer Science
University of St Andrews, UK

Abstract

We propose the design of deterministic constituent parsers that choose parser actions according to the probabilities of parses of a given probabilistic context-free grammar. Several variants are presented. One of these deterministically constructs a parse structure while postponing commitment to labels. We investigate theoretical time complexities and report experiments.

1 Introduction

Transition-based dependency parsing (Yamada and Matsumoto, 2003; Nivre, 2008) has attracted considerable attention, not only due to its high accuracy but also due to its small running time. The latter is often realized through determinism, i.e. for each configuration a unique next action is chosen. The action may be a shift of the next word onto the stack, or it may be the addition of a dependency link between words.

Because of the determinism, the running time is often linear or close to linear; most of the time and space resources are spent on deciding the next parser action. Generalizations that allow nondeterminism, while maintaining polynomial running time, were proposed by (Huang and Sagae, 2010; Kuhlmann et al., 2011).

This work has influenced, and has been influenced by, similar developments in constituent parsing. The challenge here is to deterministically choose a shift or reduce action. As in the case of dependency parsing, solutions to this problem are often expressed in terms of classifiers of some kind. Common approaches involve maximum entropy (Ratnaparkhi, 1997; Tsuruoka and Tsujii, 2005), decision trees (Wong and Wu, 1999; Kalt, 2004), and support vector machines (Sagae and Lavie, 2005).

The programming-languages community recognized early on that large classes of grammars allow deterministic, i.e. linear-time, parsing, provided parsing decisions are postponed as long as possible. This has led to (deterministic) LR(k) parsing (Knuth, 1965; Sippu and Soisalon-Soininen, 1990), which is a form of shift-reduce parsing. Here the parser needs to commit to a grammar rule only after all input covered by the right-hand side of that rule has been processed, while it may consult the next k symbols (the lookahead). LR is the optimal, i.e. most deterministic, parsing strategy that has this property. Deterministic LR parsing has also been considered relevant to psycholinguistics (Shieber, 1983).

Nondeterministic variants of LR(k) parsing, for use in natural language processing, have been proposed as well, some using tabulation to ensure polynomial running time in the length of the input string (Tomita, 1988; Billot and Lang, 1989). However, nondeterministic LR(k) parsing is potentially as expensive as, and possibly more expensive than, traditional tabular parsing algorithms such as CKY parsing (Younger, 1967; Aho and Ullman, 1972), as shown by for example (Shann, 1991); greater values of k make matters worse (Lankhorst, 1991). For this reason, LR parsing is sometimes enhanced by attaching probabilities to transitions (Briscoe and Carroll, 1993), which allows pruning of the search space (Lavie and Tomita, 1993). This by itself is not uncontroversial, for several reasons. First, the space of probability distributions expressible by a LR automaton is incomparable to that expressible by a CFG (Nederhof and Satta, 2004). Second, because an LR automaton may have many more transitions than rules, more training data may be needed to accurately estimate all parameters.

The approach we propose here retains some important properties of the above work on LR parsing. First, parser actions are delayed as long as

Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, pages 338–347,
Gothenburg, Sweden, April 26-30 2014.

possible, under the constraint that a rule is committed to no later than when the input covered by its right-hand side has been processed. Second, the parser action that is performed at each step is the most likely one, given the left context, the lookahead, and a probability distribution over parses given by a PCFG.

There are two differences with traditional LR parsing however. First, there is no explicit representation of LR states, and second, probabilities of actions are computed dynamically from a PCFG rather than retrieved as part of static transitions. In particular, this is unlike some other early approaches to probabilistic LR parsing such as (Ng and Tomita, 1991).

The mathematical framework is reminiscent of that used to compute prefix probabilities (Jelinek and Lafferty, 1991; Stolcke, 1995). One major difference is that instead of a prefix string, we now have a stack, which does not need to be parsed. In the first instance, this seems to make our problem easier. For our purposes however, we need to add new mechanisms in order to take lookahead into consideration.

It is known, e.g. from (Cer et al., 2010; Candito et al., 2010), that constituent parsing can be used effectively to achieve dependency parsing. It is therefore to be expected that our algorithms can be used for dependency parsing as well. The parsing steps of shift-reduce parsing with a binary grammar are in fact very close to those of many dependency parsing models. The major difference is, again, that instead of general-purpose classifiers to determine the next step, we would rely directly on a PCFG.

The emphasis of this paper is on deriving the necessary equations to build several variants of deterministic shift-reduce parsers, all guided by a PCFG. We also offer experimental results.

2 Shift-reduce parsing

In this section, we summarize the theory of LR parsing. As usual, a *context-free grammar* (CFG) is represented by a 4-tuple $(\Sigma,\ N,\ S,\ P)$, where Σ and N are two disjoint finite sets of *terminals* and *nonterminals*, respectively, $S \in N$ is the *start symbol*, and P is a finite set of *rules*, each of the form $A \to \alpha$, where $A \in N$ and $\alpha \in (\Sigma \cup N)^*$. By *grammar symbol* we mean a terminal or nonterminal. We use symbols $A, B, C, \ldots$ for nonterminals, $a, b, c, \ldots$ for terminals, $v, w, x, \ldots$ for strings of terminals, X for grammar symbols, and $\alpha, \beta, \gamma, \ldots$ for strings of grammar symbols. For technical reasons, a CFG is often *augmented* by an additional rule $S^\dagger \to S\$$, where $S^\dagger \notin N$ and $\$ \notin \Sigma$. The symbol $\$$ acts as an end-of-sentence marker.

As usual, we have a (right-most) 'derives' relation $\Rightarrow_{rm}$, $\Rightarrow^*_{rm}$ denotes derivation in zero or more steps, and $\Rightarrow^+_{rm}$ denotes derivation in one or more steps. If d is a string of rules $\pi_1 \cdots \pi_k$, then $\alpha \stackrel{d}{\Rightarrow}_{rm} \beta$ means that β can be derived from α by applying this list of rules in right-most order. A string α such that $S \Rightarrow^*_{rm} \alpha$ is called a *right-sentential form*.

The last rule $A \to \beta$ used in a derivation $S \Rightarrow^+_{rm} \alpha$ together with the position of (the relevant occurrence of) β in α we call the *handle* of the derivation. In more detail, such a derivation can be written as $S = \underline{A_0} \Rightarrow_{rm} \alpha_1 A_1 \underline{\beta_1} \Rightarrow^*_{rm} \alpha_1 \underline{A_1} v_1 \Rightarrow_{rm} \alpha_1 \alpha_2 A_2 \underline{\beta_2} v_2 \Rightarrow^*_{rm} \ldots \Rightarrow^*_{rm} \alpha_1 \cdots \alpha_{k-1} \underline{A_{k-1}} v_{k-1} \cdots v_1 \Rightarrow_{rm} \alpha_1 \cdots \alpha_{k-1} \beta v_{k-1} \cdots v_1$, where $k \geq 1$, and $A_{i-1} \to \alpha_i A_i \beta_i$ $(1 \leq i < k)$ and $A_{k-1} \to \beta$ are in P. The underlined symbols are those that are (recursively) rewritten to terminal strings within the following relation $\Rightarrow_{rm}$ or $\Rightarrow^*_{rm}$. The handle here is $A_{k-1} \to \beta$, together with the position of β in the right-sentential form, just after $\alpha_1 \cdots \alpha_{k-1}$. A prefix of $\alpha_1 \cdots \alpha_{k-1}\beta$ is called a *viable prefix* in the derivation.

Given an input string w, a *shift-reduce parser* finds a right-most derivation of w, but in reverse order, identifying the last rules first. It manipulates configurations of the form $(\alpha, v\$)$, where α is a viable prefix (in at least one derivation) and v is a suffix of w. The initial configuration is $(\varepsilon, w\$)$, where ε is the empty string. The two allowable steps are $(\alpha, av\$) \vdash (\alpha a, v\$)$, which is called a *shift*, and $(\alpha\beta, v\$) \vdash (\alpha A, v\$)$ where $A \to \beta$ is in P, which is called a *reduce*. Acceptance happens upon reaching a configuration $(S, \$)$.

A *1-item* has the form $[A \to \alpha \bullet \beta, a]$, where $A \to \alpha\beta$ is a rule. The bullet separates the right-hand side into two parts, the first of which has been matched to processed input. The symbol $a \in \Sigma \cup \{\$\}$ is called the *follower*.

In order to decide whether to apply a shift or reduce after reaching a configuration $(X_1 \cdots X_k, w)$, one may construct the sets $I_0, \ldots, I_k$, inductively defined as follows, with $0 \leq i \leq k$:

- if $S \to \sigma$ in P, then $[S \to \bullet\ \sigma, \$] \in I_0$,

- if $[A \rightarrow \alpha \bullet B\beta, a] \in I_i$, $B \rightarrow \gamma$ in P, and $\beta \Rightarrow^*_{rm} x$, then $[B \rightarrow \bullet\ \gamma, b] \in I_i$, where $b = 1 : xa$,
- if $[A \rightarrow \alpha \bullet X_i\beta, a] \in I_{i-1}$ then $[A \rightarrow \alpha X_i \bullet \beta, a] \in I_i$.

(The expression $1 : y$ denotes a if $y = az$, for some a and z; we leave it undefined for $y = \varepsilon$.) Exhaustive application of the second clause above will be referred to as the *closure* of a set of items.

It is not difficult to show that if $[A \rightarrow \alpha \bullet, a] \in I_k$, then α is of the form $X_{j+1} \cdots X_k$, some j, and $A \rightarrow \alpha$ at position $j + 1$ is the handle of at least one derivation $S \Rightarrow^*_{rm} X_1 \cdots X_k ax$, some x. If furthermore $a = 1 : w$, where $1 : w$ is called the *lookahead* of the current configuration $(X_1 \cdots X_k, w)$, then this justifies a reduce with $A \rightarrow \alpha$, as a step that potentially leads to a complete derivation; this is only 'potentially' because the actual remaining input w may be unlike ax, apart from the matching one-symbol lookahead.

Similarly, if $[A \rightarrow \alpha \bullet a\beta, b] \in I_k$, then $\alpha = X_{j+1} \cdots X_k$, some j, and if furthermore $a = 1 : w$, then a shift of symbol a is a justifiable step. Potentially, if a is followed by some x such that $\beta \Rightarrow^*_{rm} x$, then we may eventually obtain a stack $X_1 \cdots X_j \alpha a \beta$, which is a prefix of a right-sentential form, with the handle being $A \rightarrow \alpha a \beta$ at position $j + 1$.

For a fixed grammar, the collection of all possible sets of 1-items that may arise in processing any viable prefix is a finite set. The technique of *LR(1) parsing* relies on a precomputation of all such sets of items, each of which is turned into a state of the *LR(1) automaton*. The initial state consists of $closure(\{[S \rightarrow \bullet\ \sigma, \$] \mid S \rightarrow \sigma \in P\})$. The automaton has a transition labeled X from I to J if $goto(I, X) = J$, where $goto(I, X) = closure(\{[A \rightarrow \alpha X \bullet \beta, a] \mid [A \rightarrow \alpha \bullet X\beta, a] \in I\})$. In the present study, we do not precompute all possible states of the LR(1) automaton, as this would require prohibitive amounts of time and memory. Instead, our parsers are best understood as computing LR states *dynamically*, while furthermore attaching probabilities to individual items.

In the sequel we will assume that all rules either have the (lexical) form $A \rightarrow a$, the (binary) form $A \rightarrow BC$, or the (unary) form $A \rightarrow B$. This means that $A \Rightarrow^*_{rm} \varepsilon$ is not possible for any A. The end-of-sentence marker is now introduced by two augmented rules $S^\dagger \rightarrow SS^\$$ and $S^\$ \rightarrow \$$.

3 Probabilistic shift-reduce parsing

A *probabilistic* CFG (PCFG) is a 5-tuple (Σ, N, S, P, p), where the extra element p maps rules to probabilities. The probability of a derivation $\alpha \stackrel{d}{\Rightarrow}_{rm} \beta$, with $d = \pi_1 \cdots \pi_k$, is defined to be $p(d) = \prod_i p(\pi_i)$. The probability $p(w)$ of a string w is defined to be the sum of $p(d)$ for all d with $S \stackrel{d}{\Rightarrow}_{rm} w$.

We assume *properness*, i.e. $\sum_{\pi=A\rightarrow\alpha} p(\pi) = 1$ for all A, and *consistency*, i.e. $\sum_w p(w) = 1$. Properness and consistency together imply that for each nonterminal A, the sum of $p(d)$ for all d with $\exists_w A \stackrel{d}{\Rightarrow}_{rm} w$ equals 1. We will further assume an augmented PCFG with extra rules $S^\dagger \rightarrow SS^\$$ and $S^\$ \rightarrow \$$ both having probability 1.

Consider a viable prefix $A_1 \cdots A_k$ on the stack of a shift-reduce parser, and lookahead a. Each right-most derivation in which the handle is $A \rightarrow A_{k-1}A_k$ at position $k - 1$ must be of the form sketched in Figure 1.

Because of properness and consistency, we may assume that all possible subderivations generating strings entirely to the right of the lookahead have probabilities summing to 1. To compactly express the remaining probabilities, we need additional notation. First we define:

$$\mathcal{V}(C, D) \quad = \sum_{d\,:\,\exists_w C \stackrel{d}{\Rightarrow}_{rm} Dw} p(d)$$

for any pair of nonterminals C and D. This will be used later to 'factor out' a common term in a (potentially infinite) sum of probabilities of subderivations; the w in the expression above corresponds to a substring of the unknown input beyond the lookahead. In order to compute such values, we fix an ordering of the nonterminals by $N = \{C_1, \ldots, C_r\}$, with $r = |N|$. We then construct a matrix M, such that $M_{i,j} = \sum_{\pi=C_i\rightarrow C_j\alpha} p(\pi)$. In words, we sum the probabilities of all rules that have left-hand side C_i and a right-hand side beginning with C_j.

A downward path in a parse tree from an occurrence of C to an occurrence of D, restricted to following always the first child, can be of any length n, including $n = 0$ if $C = D$. This means we need to obtain the matrix $M^* = \sum_{0\leq n} M^n$, and $\mathcal{V}(C_i, C_j) = M^*_{i,j}$ for all i and j. Fortunately, $M^*_{i,j}$ can be effectively computed as $(I - M)^{-1}$, where I is the identity matrix of size r and the superscript denotes matrix inversion.

We further define:

$$\mathcal{U}(C,D) \;=\; \sum_{d\,:\,C \stackrel{d}{\Rightarrow}_{rm} D} p(d)$$

much as above, but restricting attention to unit rules.

The expected number of times a handle $A \rightarrow A_{k-1}A_k$ at position $k-1$ occurs in a right-most derivation with viable prefix $A_1 \cdots A_k$ and lookahead a is now given by:

$$\mathcal{E}(A_1 \cdots A_k, a, A \rightarrow A_{k-1}A_k) \;=$$
$$\sum_{\substack{S^\dagger = E_0, \ldots, E_{k-2}, F_1, \ldots, F_{k-1} = A,\\ F, E, B, B', m : 0 \leq m < k-1}}$$
$$\prod_{i:\ 1 \leq i \leq m} \mathcal{V}(E_{i-1}, F_i) \cdot p(F_i \rightarrow A_i E_i) \cdot$$
$$\mathcal{V}(E_m, F) \cdot p(F \rightarrow EB) \cdot \mathcal{U}(E, F_{m+1}) \cdot$$
$$\prod_{i:\ m < i < k-1} p(F_i \rightarrow A_i E_i) \cdot \mathcal{U}(E_i, F_{i+1}) \cdot$$
$$p(F_{k-1} \rightarrow A_{k-1}A_k) \cdot \mathcal{V}(B, B') \cdot p(B' \rightarrow a)$$

Note that the value above is not a probability and may exceed 1. This is because the same viable prefix may occur several times in a single right-most derivation.

At first sight, the computation of $\mathcal{E}$ seems to require an exponential number of steps in k. However, we can use an idea similar to that commonly used for computation of forward probabilities for HMMs (Rabiner, 1989). We first define $\mathcal{F}$:

$$\mathcal{F}(\varepsilon, E) \;=\; \begin{cases} 1 \text{ if } E = S^\dagger \\ 0 \text{ otherwise} \end{cases}$$
$$\mathcal{F}(\alpha A, E) \;=\; \sum_{E', \pi = F \rightarrow AE} \mathcal{F}(\alpha, E') \cdot \mathcal{V}(E', F) \cdot p(\pi)$$

This corresponds to the part of the definition of $\mathcal{E}$ involving $A_1, \ldots, A_m$, $E_0, \ldots, E_m$ and $F_1, \ldots, F_m$. We build on this by defining:

$$\mathcal{G}(\alpha, E, B) \;=\; \sum_{E', \pi = F \rightarrow EB} \mathcal{F}(\alpha, E') \cdot \mathcal{V}(E', F) \cdot p(\pi)$$

One more recursive function is needed for what was $A_{m+1}, \ldots, A_{k-2}$, $E_{m+1}, \ldots, E_{k-2}$ and $F_{m+1}, \ldots, F_{k-2}$ in the earlier definition of $\mathcal{E}$:

$$\mathcal{H}(\varepsilon, E, B) = \mathcal{G}(\varepsilon, E, B)$$
$$\mathcal{H}(\alpha A, E, B) = \sum_{E', \pi = F \rightarrow AE} \mathcal{H}(\alpha, E', B) \cdot \mathcal{U}(E', F) \cdot p(\pi) + \mathcal{G}(\alpha A, E, B)$$

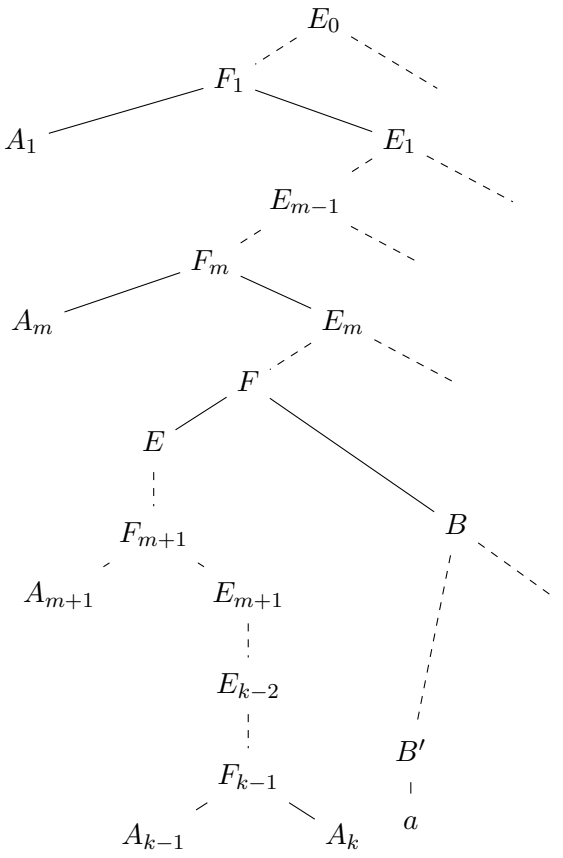

Figure 1: Right-most derivation leading to $F_{k-1} \rightarrow A_{k-1}A_k$ in viable prefix $A_1 \cdots A_k$ with lookahead a.

Finally, we can express $\mathcal{E}$ in terms of these recursive functions, considering the more general case of any rule $\pi = F \rightarrow \beta$:

$$\mathcal{E}(\alpha\beta, a, F \rightarrow \beta) \;=$$
$$\sum_{E,B} \mathcal{H}(\alpha, E, B) \cdot \mathcal{U}(E, F) \cdot p(\pi) \cdot \mathcal{L}(B, a)$$
$$\mathcal{E}(\alpha, a, F \rightarrow \beta) \;=\; 0 \quad \text{if } \neg\exists_\gamma\ \alpha = \gamma\beta$$

where:

$$\mathcal{L}(B, a) \;=\; \sum_{\pi = B' \rightarrow a} \mathcal{V}(B, B') \cdot p(\pi)$$

The expected number of times the handle is to be found to the right of α, with the stack being α and the lookahead symbol being a, is:

$$\mathcal{E}(\alpha, a, \textit{shift}) \;=\; \sum_{B} \mathcal{F}(\alpha, B) \cdot \mathcal{L}(B, a)$$

The expected number of times we see a stack α with lookahead a is:

$$\mathcal{E}(\alpha, a) \;=\; \mathcal{E}(\alpha, a, \textit{shift}) + \sum_{\pi} \mathcal{E}(\alpha, a, \pi)$$

The probability that a reduce with rule π is the correct action when the stack is α and the lookahead is a is naturally $\mathcal{E}(\alpha, a, \pi)/\mathcal{E}(\alpha, a)$ and the probability that a shift is the correct action is $\mathcal{E}(\alpha, a, \mathit{shift})/\mathcal{E}(\alpha, a)$. For determining the most likely action we do not need to compute $\mathcal{E}(\alpha, a)$; it suffices to identify the maximum value among $\mathcal{E}(\alpha, a, \mathit{shift})$ and $\mathcal{E}(\alpha, a, \pi)$ for each rule π.

A deterministic shift-reduce parser can now be constructed that always chooses the most likely next action. For a given input string, the number of actions performed by this parser is linear in the input length.

A call of $\mathcal{E}$ may lead to a number of recursive calls of $\mathcal{F}$ and $\mathcal{H}$ that is linear in the stack size and thereby in the input length. Note however that by remembering the values returned by these function between parser actions, one can ensure that each additional element pushed on the stack requires a bounded number of additional calls of the auxiliary functions. Because only linearly many elements are pushed on the stack, the time complexity becomes linear in the input length.

Complexity analysis seems less favorable if we consider the number of nonterminals. The definitions of $\mathcal{G}$ and $\mathcal{H}$ each involve four nonterminals excluding the stack symbol A, so that the time complexity is $\mathcal{O}(|w| \cdot |N|^4)$, where $|w|$ is the length of the input w. A finer analysis gives $\mathcal{O}(|w| \cdot (|N| \cdot |P| + |N|^2 \cdot \|P\|))$, where $\|P\|$ is the maximum for all A of the number of rules of the form $F \rightarrow AE$. By splitting up $\mathcal{G}$ and $\mathcal{H}$ into smaller functions, we obtain complexity $\mathcal{O}(|w| \cdot |N|^3)$, which can still be prohibitive.

Therefore we have implemented an alternative that has a time complexity that is only quadratic in the size of the grammar, at the expense of a quadratic complexity in the length of the input string, as detailed in Appendix A. This is still better in practice if the number of nonterminals is much greater than the length of the input string, as in the case of the grammars we investigated.

4 Structural determinism

We have assumed so far that a deterministic shift-reduce parser chooses a unique next action in each configuration, an action being a shift or reduce. Implicit in this was that if the next action is a reduce, then also a unique rule is chosen. However, if we assume for now that all non-lexical rules are binary, then we can easily generalize the parsing algorithm to consider *all* possible rules whose right-hand sides match the top-most two stack elements, and postpone commitment to any of the nonterminals in the left-hand sides. This requires that stack elements now contain *sets* of grammar symbols. Each of these is associated with the probability of the most likely subderivation consistent with the relevant substring of the input.

Each reduce with a binary rule is implicitly followed by zero or more reduces with unary rules. Similarly, each shift is implicitly followed by a reduce with a lexical rule and zero or more reduces with unary rules; see also (Graham et al., 1980). This uses a precompiled table similar to $\mathcal{U}$, but using maximization in place of summation, defined by:

$$\mathcal{U}_{\max}(C, D) = \max_{d : C \stackrel{d}{\Rightarrow}_{rm} D} p(d)$$

More concretely, configurations have the form $(Z_1 \ldots Z_k, v\$)$, $k \geq 0$, where each Z_i $(1 \leq i \leq k)$ is a set of pairs (A, p), where A is a nonterminal and p is a (non-zero) probability; each A occurs at most once in Z_i. A shift turns $(\alpha, av\$)$ into $(\alpha Z, v\$)$, where Z consists of all pairs (E, p) such that $p = \max_F \mathcal{U}_{\max}(E, F) \cdot p(F \rightarrow a)$. A generalized binary reduce now turns $(\alpha Z_1 Z_2, v\$)$ into $(\alpha Z, v\$)$, where Z consists of all pairs (E, p) such that:

$$p = \max_{\substack{\pi = F \rightarrow A_1 A_2, \\ (A_1, p_1) \in Z_1, (A_2, p_2) \in Z_2}} \mathcal{U}_{\max}(E, F) \cdot p(\pi) \cdot p_1 \cdot p_2$$

We characterize this parsing procedure as *structurally deterministic*, as an unlabeled structure is built deterministically in the first instance. The exact choices of rules can be postponed until after reaching the end of the sentence. Then follows a straightforward process of 'backtracing', which builds the derivation that led to the computed probability associated with the start symbol.

The time complexity is now $\mathcal{O}(|w| \cdot |N|^5)$ in the most straightforward implementation, but we can reduce this to quadratic in the size of the grammar provided we allow an additional factor $|w|$ as before. For more details see Appendix B.

5 Other variants

One way to improve accuracy is to increase the size of the lookahead, beyond the current 1, comparable to the generalization from LR(1) to LR(k) parsing. The formulas are given in Appendix C.

Yet another variant investigates only the topmost n stack symbols when choosing the next parser action. In combination with Appendix A, this brings the time complexity down again to linear time in the length of the input string. The required changes to the formulas are given in Appendix D. There is a slight similarity to (Schuler, 2009), in that no stack elements beyond a bounded depth are considered at each parsing step, but in our case the stack can still have arbitrary height.

Whereas we have concentrated on determinism in this paper, one can also introduce a limited degree of nondeterminism and allow some of the most promising configurations at each input position to compete, applying techniques such as beam search (Roark, 2001; Zhang and Clark, 2009; Zhu et al., 2013), best-first search (Sagae and Lavie, 2006), or A* search (Klein and Manning, 2003) in order to keep the running time low. For comparing different configurations, one would need to multiply the values $\mathcal{E}(\alpha, a)$ as in Section 3 by the probabilities of the subderivations associated with occurrences of grammar symbols in stack α.

Further variants are obtained by replacing the parsing strategy. One obvious candidate is left-corner parsing (Rosenkrantz and Lewis II, 1970), which is considerably simpler than LR parsing. The resulting algorithm would be very different from the left-corner models of e.g. (Henderson, 2003), which rely on neural networks instead of PCFGs.

6 Experiments

We used the WSJ treebank from OntoNotes 4.0 (Hovy et al., 2006), with Sections 2-21 for training and the 2228 sentences of up to 40 words from Section 23 for testing. Grammars with different sizes, and in the required binary form, were extracted by using the tools from the Berkeley parser (Petrov et al., 2006), with between 1 and 6 split-merge cycles. These tools offer a framework for handling unknown words, which we have adopted.

The implementation of the parsing algorithms is in C++, running on a desktop with four 3.1GHz Intel Core i5 CPUs. The main algorithm is that of Appendix C, with lookahead k between 1 and 3, also in combination with structural determinism (Appendix B), which is indicated here by *sd*. The variant that consults the stack down to bounded depth n (Appendix D) will only be reported for $k = 1$ and $n = 5$.

Bracketing recall, precision and F-measure, are computed using evalb, with settings as in (Collins, 1997), except that punctuation was deleted.[1] Table 1 reports results.

A nonterminal B in the stack may occur in a small number of rules of the form $A \rightarrow BC$. The C of one such rule is needed next in order to allow a reduction. If future input does not deliver this C, then parsing may fail. This problem becomes more severe as nonterminals become more specific, which is what happens with an increase of the number of split-merge cycles. Even more failures are introduced by removing the ability to consult the complete stack, which explains the poor results in the case of $k = 1, n = 5$; lower values of n lead to even more failures, and higher values further increase the running time. That the running time exceeds that of $k = 1$ is explained by the fact that with the variant from Appendix D, every pop or push requires a complete recomputation of all function values.

Parse failures can be almost completely eliminated however by choosing higher values of k and by using structural determinism. A combination thereof leads to high accuracy, not far below that of the Viterbi parses. Note that one cannot expect the accuracy of our deterministic parsers to *exceed* that of Viterbi parses. Both rely on the same model (a PCFG), but the first is forced to make local decisions without access to the input string that follows the bounded lookahead.

7 Conclusions

We have shown that deterministic parsers can be constructed from a given PCFG. Much of the accuracy of the grammar can be retained by choosing a large lookahead in combination with 'structural determinism', which postpones commitment to nonterminals until the end of the input is reached.

Parsers of this nature potentially run in linear time in the length of the input, but our parsers are better implemented to run in quadratic time. In terms of the grammar size, the experiments suggest that the number of rules is the dominating factor. The size of the lookahead strongly affects running time. The extra time costs of structural determinism are compensated by an increase in accuracy and a sharp decrease of the parse failures.

[1] Evalb otherwise stumbles over e.g. a part of speech consisting of two single quotes in the parsed file, against a part of speech 'POS' in the gold file, for an input token consisting of a single quote.

Table 1: Total time required (seconds), number of parse failures, recall, precision, F-measure, for deterministic parsing, compared to the Viterbi parses as computed with the Berkeley parser.

	time	fail	R	P	F1
1-split-merge (12,059 rules)					
$k=1$	43	11	67.20	66.67	66.94
$k=2$	99	0	70.74	71.01	70.88
$k=3$	199	0	71.41	71.85	71.63
$k=1, sd$	62	0	68.12	68.52	68.32
$k=2, sd$	135	0	70.98	71.72	71.35
$k=3, sd$	253	0	71.31	72.50	71.90
$k=1, n=5$	56	170	66.19	65.67	65.93
Viterbi		0	72.45	74.55	73.49
2-split-merge (32,994 rules)					
$k=1$	120	33	72.65	70.50	71.56
$k=2$	275	1	78.44	77.26	77.84
$k=3$	568	0	79.81	79.27	79.54
$k=1, sd$	196	0	74.78	74.96	74.87
$k=2, sd$	439	0	79.96	80.40	80.18
$k=3, sd$	770	0	80.49	81.20	80.85
$k=1, n=5$	146	247	72.27	70.34	71.29
Viterbi		0	82.16	82.69	82.43
3-split-merge (95,647 rules)					
$k=1$	305	75	74.39	72.33	73.35
$k=2$	770	3	81.32	80.35	80.83
$k=3$	1,596	0	82.78	82.35	82.56
$k=1, sd$	757	0	78.11	78.37	78.24
$k=2, sd$	1,531	0	82.85	83.39	83.12
$k=3, sd$	2,595	0	83.66	84.25	83.96
$k=1, n=5$	404	401	74.52	72.39	73.44
Viterbi		0	85.38	86.03	85.71

	time	fail	R	P	F1
4-split-merge (269,162 rules)					
$k=1$	870	115	75.69	73.30	74.48
$k=2$	2,257	1	83.48	82.35	82.91
$k=3$	4,380	1	84.95	84.06	84.51
$k=1, sd$	2,336	1	80.82	80.65	80.74
$k=2, sd$	4,747	0	85.52	85.64	85.58
$k=3, sd$	7,728	0	86.62	86.82	86.72
$k=1, n=5$	1,152	508	76.21	73.92	75.05
Viterbi		0	87.95	88.10	88.02
5-split-merge (716,575 rules)					
$k=1$	3,166	172	76.17	73.44	74.78
$k=2$	7,476	2	84.14	82.80	83.46
$k=3$	14,231	1	86.05	85.24	85.64
$k=1, sd$	7,427	1	81.99	81.44	81.72
$k=2, sd$	14,587	0	86.89	87.00	86.95
$k=3, sd$	24,553	0	87.67	87.82	87.74
$k=1, n=5$	4,572	559	77.65	75.13	76.37
Viterbi		0	88.65	89.00	88.83
6-split-merge (1,947,915 rules)					
$k=1$	7,741	274	76.60	74.08	75.32
$k=2$	19,440	5	84.60	83.17	83.88
$k=3$	35,712	0	86.02	85.07	85.54
$k=1, sd$	19,530	1	82.64	81.95	82.29
$k=2, sd$	39,615	0	87.36	87.20	87.28
$k=3, sd$	64,906	0	88.16	88.26	88.21
$k=1, n=5$	10,897	652	77.89	75.57	76.71
Viterbi		0	88.69	88.99	88.84

There are many advantages over other approaches to deterministic parsing that rely on general-purpose classifiers. First, some state-of-the-art language models are readily available as PCFGs. Second, most classifiers require treebanks, whereas our algorithms are also applicable to PCFGs that were obtained in any other way, for example through intersection of language models. Lastly, our algorithms fit within well understood automata theory.

Acknowledgments We thank the reviewers.

A Formulas for quadratic time complexity

The following are the formulas that correspond to the first implemented variant. Relative to Section 3, some auxiliary functions are broken up, and associating the lookahead a with an appropriate nonterminal B is now done in $\mathcal{G}$:

$$\begin{aligned}
\mathcal{F}(\varepsilon, E) &= \begin{cases} 1 \text{ if } E = S^\dagger \\ 0 \text{ otherwise} \end{cases} \\
\mathcal{F}(\alpha A, E) &= \sum_{\pi = F \to AE} \mathcal{F}'(\alpha, F) \cdot p(\pi) \\
\mathcal{F}'(\alpha, F) &= \sum_{E} \mathcal{F}(\alpha, E) \cdot \mathcal{V}(E, F) \\
\mathcal{G}(\alpha, E, a) &= \sum_{F} \mathcal{F}'(\alpha, F) \cdot \mathcal{G}'(F, E, a) \\
\mathcal{G}'(F, E, a) &= \sum_{\pi = F \to EB} p(\pi) \cdot \mathcal{L}(B, a) \\
\mathcal{H}(\varepsilon, E, a) &= \mathcal{G}(\varepsilon, E, a) \\
\mathcal{H}(\alpha A, E, a) &= \sum_{\pi = F \to AE} \mathcal{H}'(\alpha, F, a) \cdot p(\pi) \\
&\quad + \mathcal{G}(\alpha A, E, a) \\
\mathcal{H}'(\alpha, F, a) &= \sum_{E} \mathcal{H}(\alpha, E, a) \cdot \mathcal{U}(E, F)
\end{aligned}$$

$$\begin{aligned}
\mathcal{E}(\alpha\beta, a, F \to \beta) &= \mathcal{H}'(\alpha, F, a) \cdot p(F \to \beta) \\
\mathcal{E}(\alpha, a, F \to \beta) &= 0 \text{ if } \neg\exists_\gamma\, \alpha = \gamma\beta \\
\mathcal{E}(\alpha A, a, \mathit{shift}) &= \mathcal{G}(\alpha, A, a) \\
\mathcal{E}(\varepsilon, a, \mathit{shift}) &= \mathcal{L}(S^\dagger, a)
\end{aligned}$$

These equations correspond to a time complexity of $\mathcal{O}(|w|^2 \cdot |N|^2 + |w| \cdot |P|)$. Each definition except that of $\mathcal{G}'$ involves one stack (of linear size) and, at most, one terminal plus two arbitrary nonterminals. The full grammar is only considered once for every input position, in the definition of $\mathcal{G}'$.

The values are stored as vectors and matrices. For example, for each distinct lookahead symbol a, there is a (sparse) matrix containing the value of $\mathcal{G}'(F, E, a)$ at a row and a column uniquely identified by F and E, respectively.

B Formulas for structural determinism

For the variant from Section 4, we need to change only two definitions of auxiliary functions:

$$\mathcal{F}(\alpha Z, E) = \sum_{(A,p)\in Z, \pi = F \to AE} \mathcal{F}'(\alpha, F) \cdot p(\pi) \cdot p$$

$$\begin{aligned}
\mathcal{H}(\alpha Z, E, a) = &\sum_{(A,p)\in Z, \pi = F \to AE} \mathcal{H}'(\alpha, F, a) \cdot p(\pi) \cdot p \\
&+ \mathcal{G}(\alpha Z, E, a)
\end{aligned}$$

The only actions are shift and generalized binary reduce red. The definition of $\mathcal{E}$ becomes:

$$\mathcal{E}(\alpha Z_1 Z_2, a, \mathit{red}) = \sum_{\substack{(A_1,p_1)\in Z_1, (A_2,p_2)\in Z_2 \\ \pi = F \to A_1 A_2}} \mathcal{H}'(\alpha, F, a) \cdot p(\pi) \cdot p_1 \cdot p_2$$

$$\mathcal{E}(\alpha Z, a, \mathit{shift}) = \sum_{(A,p)\in Z} \mathcal{G}(\alpha, A, a) \cdot p$$

The time complexity now increases to $\mathcal{O}(|w|^2 \cdot (|N|^2 + |P|))$ due to the new $\mathcal{H}$.

C Formulas for larger lookahead

In order to handle k symbols of lookahead (Section 5) some technical problems are best avoided by having k copies of the end-of-sentence marker appended behind the input string, with a corresponding augmentation of the grammar. We generalize $\mathcal{L}(B, v)$ to be the sum of $p(d)$ for all d such that $B \overset{d}{\Rightarrow}_{rm} vx$, some x. We let $\mathcal{I}(B, v)$ be the sum of $p(d)$ for all d such that $B \overset{d}{\Rightarrow}_{rm} v$. If $\mathcal{I}$ is given for all prefixes of a fixed lookahead string of length k (this requires cubic time in k), we can compute $\mathcal{L}$ in linear time for all suffixes of the same string:

$$\begin{aligned}
\mathcal{L}(B, v) &= \sum_{B'} \mathcal{V}(B, B') \cdot \mathcal{L}'(B', v) \\
\mathcal{L}'(B, v) &= \sum_{\substack{\pi = B \to B_1 B_2, v_1, v_2: \\ v = v_1 v_2, 1 \le |v_1|, 1 \le |v_2|}} p(\pi) \cdot \mathcal{I}(B_1, v_1) \cdot \mathcal{L}(B_2, v_2) \quad \text{if } |v| > 1 \\
\mathcal{L}'(B, a) &= \sum_{\pi = B \to a} p(\pi)
\end{aligned}$$

The function $\mathcal{H}$ is generalized straightforwardly by letting it pass on a string v ($1 \le |v| \le k$) instead of a single terminal a. The same holds for $\mathcal{E}$. The function $\mathcal{G}$ requires a slightly bigger modification, leading back to $\mathcal{H}$ if not all of the lookahead has been matched yet:

$$\begin{aligned}
\mathcal{G}(\alpha, E, v) = &\sum_F \mathcal{F}'(\alpha, F) \cdot \mathcal{G}'(F, E, v) \; + \\
&\sum_{F, v_1, v_2 : v = v_1 v_2, |v_2| > 0} \mathcal{H}'(\alpha, F, v_2) \cdot \mathcal{G}''(F, E, v_1) \\
\mathcal{G}'(F, E, v) = &\sum_{\pi = F \to EB} p(\pi) \cdot \mathcal{L}(B, v) \\
\mathcal{G}''(F, E, v) = &\sum_{\pi = F \to EB} p(\pi) \cdot \mathcal{I}(B, v)
\end{aligned}$$

The time complexity is now $\mathcal{O}(k \cdot |w|^2 \cdot |N|^2 + k^3 \cdot |w| \cdot |P|)$.

D Investigation of top-most n stack symbols only

As discussed in Section 5, we want to predict the next parser action without consulting any symbols in α, when the current stack is $\alpha\beta$, with $|\beta| = n$. This is achieved by approximating $\mathcal{F}(\alpha, E)$ by the outside value of E, that is, the sum of $p(d)$ for all d such that $\exists_{\alpha,w} S \overset{d}{\Rightarrow}_{rm} \alpha E w$. Similarly, $\mathcal{H}'(\alpha, F, v)$ is approximated by $\sum_E \mathcal{G}(\alpha, E, v) \cdot \mathcal{W}(E, F)$ where:

$$\mathcal{W}(C, D) = \sum_{d\,:\,\exists_\delta C \overset{d}{\Rightarrow}_{rm} \delta D} p(d)$$

The time complexity (with lookahead k) is now $\mathcal{O}(k \cdot n \cdot |w| \cdot |N|^2 + k^3 \cdot |w| \cdot |P|)$.

References

A.V. Aho and J.D. Ullman. 1972. *Parsing*, volume 1 of *The Theory of Parsing, Translation and Compiling*. Prentice-Hall, Englewood Cliffs, N.J.

S. Billot and B. Lang. 1989. The structure of shared forests in ambiguous parsing. In *27th Annual Meeting of the ACL, Proceedings of the Conference*, pages 143–151, Vancouver, British Columbia, Canada, June.

T. Briscoe and J. Carroll. 1993. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–59.

M. Candito, J. Nivre, P. Denis, and E. Henestroza Anguiano. 2010. Benchmarking of statistical dependency parsers for French. In *The 23rd International Conference on Computational Linguistics*, pages 108–116, Beijing, China, August.

D. Cer, M.-C. de Marneffe, D. Jurafsky, and C. Manning. 2010. Parsing to Stanford dependencies: Trade-offs between speed and accuracy. In *LREC 2010: Seventh International Conference on Language Resources and Evaluation, Proceedings*, pages 1628–1632, Valletta , Malta, May.

M. Collins. 1997. Three generative, lexicalised models for statistical parsing. In *35th Annual Meeting of the ACL, Proceedings of the Conference*, pages 16–23, Madrid, Spain, July.

S.L. Graham, M.A. Harrison, and W.L. Ruzzo. 1980. An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, 2:415–462.

J. Henderson. 2003. Generative versus discriminative models for statistical left-corner parsing. In *8th International Workshop on Parsing Technologies*, pages 115–126, LORIA, Nancy, France, April.

E. Hovy, M. Marcus, M. Palmer, L. Ramshaw, and R. Weischedel. 2006. OntoNotes: The 90% solution. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 57–60, New York, USA, June.

L. Huang and K. Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the ACL*, pages 1077–1086, Uppsala, Sweden, July.

F. Jelinek and J.D. Lafferty. 1991. Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, 17(3):315–323.

T. Kalt. 2004. Induction of greedy controllers for deterministic treebank parsers. In *Conference on Empirical Methods in Natural Language Processing*, pages 17–24, Barcelona, Spain, July.

D. Klein and C.D. Manning. 2003. A* parsing: Fast exact Viterbi parse selection. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the ACL*, pages 40–47, Edmonton, Canada, May–June.

D.E. Knuth. 1965. On the translation of languages from left to right. *Information and Control*, 8:607–639.

M. Kuhlmann, C. Gómez-Rodríguez, and G. Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *49th Annual Meeting of the ACL, Proceedings of the Conference*, pages 673–682, Portland, Oregon, June.

M. Lankhorst. 1991. An empirical comparison of generalized LR tables. In R. Heemels, A. Nijholt, and K. Sikkel, editors, *Tomita's Algorithm: Extensions and Applications*, Proc. of the first Twente Workshop on Language Technology, pages 87–93. University of Twente, September.

A. Lavie and M. Tomita. 1993. GLR* – an efficient noise-skipping parsing algorithm for context free grammars. In *Third International Workshop on Parsing Technologies*, pages 123–134, Tilburg (The Netherlands) and Durbuy (Belgium), August.

M.-J. Nederhof and G. Satta. 2004. An alternative method of training probabilistic LR parsers. In *42nd Annual Meeting of the ACL, Proceedings of the Conference*, pages 551–558, Barcelona, Spain, July.

S.-K. Ng and M. Tomita. 1991. Probabilistic LR parsing for general context-free grammars. In *Proc. of the Second International Workshop on Parsing Technologies*, pages 154–163, Cancun, Mexico, February.

J. Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

S. Petrov, L. Barrett, R. Thibaux, and D. Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL*, pages 433–440, Sydney, Australia, July.

L.R. Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February.

A. Ratnaparkhi. 1997. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, pages 1–10, Providence, Rhode Island, USA, August.

B. Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276.

D.J. Rosenkrantz and P.M. Lewis II. 1970. Deterministic left corner parsing. In *IEEE Conference Record of the 11th Annual Symposium on Switching and Automata Theory*, pages 139–152.

K. Sagae and A. Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technologies*, pages 125–132, Vancouver, British Columbia, Canada, October.

K. Sagae and A. Lavie. 2006. A best-first probabilistic shift-reduce parser. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL*, pages 691–698, Sydney, Australia, July.

W. Schuler. 2009. Positive results for parsing with a bounded stack using a model-based right-corner transform. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the ACL*, pages 344–352, Boulder, Colorado, May–June.

P. Shann. 1991. Experiments with GLR and chart parsing. In M. Tomita, editor, *Generalized LR Parsing*, chapter 2, pages 17–34. Kluwer Academic Publishers.

S.M. Shieber. 1983. Sentence disambiguation by a shift-reduce parsing technique. In *21st Annual Meeting of the ACL, Proceedings of the Conference*, pages 113–118, Cambridge, Massachusetts, July.

S. Sippu and E. Soisalon-Soininen. 1990. *Parsing Theory, Vol. II: LR(k) and LL(k) Parsing*, volume 20 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag.

A. Stolcke. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):167–201.

M. Tomita. 1988. Graph-structured stack and natural language parsing. In *26th Annual Meeting of the ACL, Proceedings of the Conference*, pages 249–257, Buffalo, New York, June.

Y. Tsuruoka and J. Tsujii. 2005. Chunk parsing revisited. In *Proceedings of the Ninth International Workshop on Parsing Technologies*, pages 133–140, Vancouver, British Columbia, Canada, October.

A. Wong and D. Wu. 1999. Learning a lightweight robust deterministic parser. In *Sixth European Conference on Speech Communication and Technology*, pages 2047–2050.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *8th International Workshop on Parsing Technologies*, pages 195–206, LORIA, Nancy, France, April.

D.H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10:189–208.

Y. Zhang and S. Clark. 2009. Transition-based parsing of the Chinese treebank using a global discriminative model. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 162–171, Paris, France, October.

M. Zhu, Y. Zhang, W. Chen, M. Zhang, and J. Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *51st Annual Meeting of the ACL, Proceedings of the Conference*, volume 1, pages 434–443, Sofia, Bulgaria, August.