Natural Language Engineering **10** (1): 1–24. © 2004 Cambridge University Press
DOI: 10.1017/S1351324903003152 Printed in the United Kingdom

Robust garden path parsing

BRIAN ROARK
AT&T Labs - Research, 180 Park Avenue, Building 103,
Room E145, Florham Park, NJ 07932-0971, USA
e-mail: roark@research.att.com

(*Received 23 April 2002; revised 17 September 2002*)

Abstract

This paper presents modifications to a standard probabilistic context-free grammar that enable a predictive parser to avoid garden pathing without resorting to any ad-hoc heuristic repair. The resulting parser is shown to apply efficiently to both newspaper text and telephone conversations with complete coverage and excellent accuracy. The distribution over trees is peaked enough to allow the parser to find parses efficiently, even with the much larger search space resulting from overgeneration. Empirical results are provided for both *Wall St. Journal* and Switchboard test corpora.

1 Introduction

Roark (2001a) presented a broad-coverage incremental parser that was used effectively as a probabilistic language model for speech recognition. While that parser achieved high coverage, it was not robust, since it failed to find a parse for some small percentage (around one percent) of strings in the *Wall St. Journal* test set, by virtue of its incremental pruning strategy. This paper outlines a modified probabilistic model that results in a robust parser, despite the 'garden path' incremental parsing approach. This model is shown is provide higher accuracy and complete coverage without an undue efficiency burden. The parser is then applied to noisier strings taken from the Switchboard corpus of telephone conversations, and is shown to provide high accuracy and complete coverage in this domain as well.

In the case of the Wall St. Journal, the failure to parse in Roark (2001a) was quite frequently due to rare uses of punctuation that happen not to have been observed in particular syntactic contexts in the training corpus. More generally, many of the rules in the treebank are very flat[1], so that long sequences of children categories are observed, with none of the intermediate constituents that would be observed in more detailed hierarchical structures. For example, noun phrases can consist of a determiner followed by a sequence of nouns, adjectives and other 'nouny' things, in nearly every permutation. Given limited training data, the probability of observing

[1] See the bracketing guidelines in Bies, Ferguson, Katz, MacIntyre, Tredinnick, Kim, Marcinkiewicz and Schasberger (1995) for details on how the Penn Treebank was annotated.

all possible permutations is zero. We will discuss a method of smoothing to allow for unseen rules to be assigned a probability.

To give an idea of how much of a problem this might be, of the 15 thousand or so PCFG rules that can be induced from the Penn Wall St. Journal Treebank in the standard way, over 3600 of these are NP rules with only pre-terminals on the right-hand side. The pre-terminal categories on the right-hand side of these base NPs include determiners, adjectives, common and proper nouns, gerunds, and punctuation, among other things. While there are some ordering constraints (e.g. determiners occur typically first and only once), productive noun compounding and variations in punctuation result in many possible combinations. Some of these are observed, but many are not. For example, the following rule occurs in the training data with a probability of approximately 0.00000285:

(1) $$\text{NP} \rightarrow \text{DT JJ JJ NN NN NNS}$$

The POS tag DT is for determiners, JJ for adjectives, NN for singular common nouns, and NNS for plural common nouns, so this rule would cover something like *'the delicious black duck beak soups'*. Unfortunately, the following two rules are not observed in the training corpus, and thus have a probability of zero with the current grammar estimation technique:

(2) $$\text{NP} \rightarrow \text{DT JJ JJ JJ NN NN NNS}$$

(3) $$\text{NP} \rightarrow \text{DT `` JJ '' JJ NN NN NNS}$$

Hence, there is no way to cover, in a flat NP rule, something like *'the so-called delicious black duck beak soups'*, nor something like *'the* "*delicious*" *black duck beak soups'*. This may be argued to be a short-coming of the grammar formalism, and that may be true; more hierarchical structure would help with some of this, to the extent that there would be more exemplars of shorter rules.[2] However, this is the grammar that has been provided by the annotators via the treebank, and the grammar estimation techniques that have been used up to now do not provide sufficient probability to unseen rules of the sort we have given in the example.

A method that has been adopted for treebank parsing in the past (Collins 1997; Charniak 2000) is what Charniak has termed a Markov grammar. The basic idea is to make a Markov assumption about the dependencies between the children, i.e. that the probabilities of children are independent of their siblings when the distance between them is beyond some fixed *n*. This method has been noted to improve the accuracy of the above mentioned statistical parsers, but it also serves to increase the coverage.

In the next section, we will briefly outline the parser and the model from Roark (2001a). We then motivate and describe the changes to the model, followed by empirical results.

[2] Johnson (1998) showed, however, that the perhaps more linguistically well-motivated structures may not be as effective as flat structures for modeling the dependencies, since the additional level in the hierarchy carries an independence assumption that appears to be false in general.

```
TOP-DOWN-PARSER(S†, M_G, w = w_0 ... w_n ⟨/s⟩)
 1  i ← 0
 2  𝒮 = S†$        ▷ Let 𝒮 be the stack, and $ the end-of-stack marker
 3  repeat
 4        ▷ let X be the top stack symbol on 𝒮
 5        POP X from 𝒮
 6        if X ∈ T
 7          then if X = w_i
 8                 then i ← i + 1
 9                 else ERROR
10          else if M_G[X, w_i] = X → Y_1 ... Y_k
11                 then PUSH Y_1 ... Y_k onto 𝒮
12                      OUTPUT(X → Y_1 ... Y_k)
13                 else ERROR
14  until X = $
15  if w_i ≠ ⟨/s⟩      ▷ if look-ahead is not the end-of-string
16    then ERROR
```

Fig. 1. A deterministic top-down parsing algorithm, modified from Aho, Sethi, and Ullman (Algorithm 4.3), taking a start symbol $S^\dagger$, a parsing table M_G, and an input string w as arguments. The symbol ▷ precedes comments.

2 Top-down probabilistic parsing

2.1 Background

This parser is essentially a stochastic version of the top-down parser described in Aho, Sethi and Ullman (1986). To present the parser, we first present their deterministic algorithm, then discuss how to handle the non-determinism. The parser will be presented as taking strings of words as input, but it can also be applied to strings of POS tags, with the obvious changes to look-ahead calculations.

A CFG $G = (V, T, P, S^\dagger)$, consists of a set of non-terminal symbols V, a set of terminal symbols T, a start symbol $S^\dagger \in V$, and a set of rule productions P of the form: $A \rightarrow \alpha$, where $A \in V$ and $\alpha \in (V \cup T)^*$. Deterministic top-down parsing (see the algorithm[3] in Figure 1) is effected via a parsing table M_G, which takes a non-terminal category X and the look-ahead word (the next word in the string) w_i, and returns either a rule expanding the non-terminal or a fail symbol. Consider the set of productions from a very simple context-free grammar[4]:

(4) $\text{S} \rightarrow \text{NP VP}$

(5) $\text{NP} \rightarrow \text{DT NN}$

(6) $\text{VP} \rightarrow \text{V NP}$

(7) $\text{DT} \rightarrow \textit{the}$

(8) $\text{NN} \rightarrow \textit{moon}$

[3] Algorithms in this paper are formatted according to the style in Corman, Leiserson and Rivest (1990).

[4] We consider rules expanding pre-terminals to terminals as rules in the grammar, rather than a lexicon separate from the productions.

(9) NN → *sun*

(10) V → *is*

This grammar is very limited, but can handle strings like *'the moon is the moon'*. The parsing table will have an entry for the pair M_G(S,*the*) = S → NP VP, since rule 4 provides the only possible path, given the above grammar, from S to *the*. Given a grammar G, a parsing table M_G can be built, by finding, for every non-terminal/terminal pair, all rules that can be the first step in a path from the non-terminal to the terminal. If a parsing table can be built where each entry in the table is unique, i.e. in which there is no ambiguity about which rule to apply with any pair (such as the above grammar), then the grammar is said to be LL(1), where LL stands for left-to-right and leftmost, and the 1 means that there is one terminal item in look-ahead. With such a parsing table, one can deterministically parse the input top-down, with the algorithm in Figure 1.

Suppose that we were to enrich our small toy grammar with a couple of rules to handle NP modification with prepositional phrases, to be able to handle strings like *'the moon is the sun of the night'*. The rules might look something like:

(11) NP → NP PP

(12) PP → IN NP

(13) IN → *of*

(14) NN → *night*

With the introduction of these rules, the grammar is no longer LL(1), because for certain non-terminal/terminal pairs, there is more than one rule in the table. For example, in this case M_G(NP,*the*) has two entries: rules 5 and 11. Grammars sufficient to cover freely occurring strings of English are typically massively ambiguous, so any top-down approach must be able to handle such non-determinism.

Our basic approach will be to keep many separate derivations, each of which follows a search path akin to the deterministic parser just outlined. This will involve assigning each partial derivation a *figure-of-merit*, or a score of how good the derivation is. The goal is to work on just the promising ones, and discard the rest. Finding an appropriate figure-of-merit can be difficult, because of issues of comparability. Two competing analyses may be at different points in their derivation, so this figure-of-merit must be able to, in a sense, normalize the scores with respect to the extent of the derivation. This can be done by including in the score the probability of the derivation to that point, as well as some estimate of how much probability the analysis is going to spend to extend the derivation. We do not compare derivations with different terminal yields, but rather extend the set of competing derivations to the current word before moving on to the next word; hence the derivations are more comparable than they might otherwise be. Each derivation probability is monotonically decreasing, i.e. every rule added to the derivation decreases its probability; yet each rule also brings the existing derivation closer to the look-ahead word, so that the amount of probability that will have to be spent, for promising analyses, to reach the look-ahead word will offset the drop in probability. Thus attention is appropriately focused on these promising derivations.

2.2 Algorithm

To introduce our parsing algorithm, we will first define *candidate analysis* (i.e. a partial parse), and then a *derives* relation between candidate analyses. We will then present the algorithm in terms of this relation.

The input to the parser is a string w_0^n and a PCFG G[5]. The parser's basic data structure is a priority queue of candidate analyses. A candidate analysis $C = (D, \mathscr{S}, P_D, F, w_i^n)$ consists of a partial derivation D, a stack $\mathscr{S}$, a derivation probability P_D, a figure-of-merit F, and a string w_i^n remaining to be parsed. The first word in the string remaining to be parsed, w_i, we will call the *look-ahead* word. The derivation D consists of a sequence of rules used from G. The stack $\mathscr{S}$ contains a sequence of non-terminal symbols that need to be accounted for, and an end-of-stack marker \$ at the bottom. The probability P_D is the product of the probabilities of all rules in the derivation D. F is the product of P_D and a look-ahead probability, LAP($\mathscr{S}$, w_i), which is a measure of the likelihood of the stack $\mathscr{S}$ rewriting with w_i at its left corner. Exactly how the LAP is calculated is described below.

We can define a *derives* relation, denoted $\Rightarrow$, between two candidate analyses as follows. $(D, \mathscr{S}, P_D, F, w_i^n) \Rightarrow (D', \mathscr{S}', P_{D'}, F', w_j^n)$ if and only if[6]

(i) $D' = D + A \rightarrow \beta$
(ii) $\mathscr{S} = A\alpha\$$;
(iii) either $\mathscr{S}' = \beta\alpha\$$ and $j = i$
or $\beta = w_i$, $j = i + 1$, and $\mathscr{S}' = \alpha\$$;
(iv) $P_{D'} = P_D \mathrm{P}(A \rightarrow \beta)$; and
(v) $F' = P_{D'} \mathrm{LAP}(\mathscr{S}', w_j)$.

The parse then begins with a single candidate analysis on the priority queue: $(\langle\rangle, S^\dagger\$, 1, 1, w_0^n)$. It then proceeds as follows. The top ranked candidate analysis, $C = (D, \mathscr{S}, P_D, F, w_i^n)$, is popped from the priority queue. If $\mathscr{S} = \$$ and $w_i = \langle/\mathrm{s}\rangle$, then the analysis is complete. Otherwise, all C' such that $C \Rightarrow C'$ are pushed onto the priority queue.

We implement this as a beam search. For each word position i, we have a separate priority queue, $\mathscr{H}_i$, of analyses with look-ahead w_i. When there are 'enough' analyses by some criteria (which we will discuss below) on priority queue $\mathscr{H}_{i+1}$, all candidate analyses remaining on $\mathscr{H}_i$ are discarded. Since $w_n = \langle/\mathrm{s}\rangle$, all parses that are pushed onto $\mathscr{H}_{n+1}$ are complete. The parse on $\mathscr{H}_{n+1}$ with the highest probability is returned for evaluation. In the case that no complete parse is found, a partial parse is returned and evaluated. Figure 2 presents the algorithm formally.

The LAP is the probability of a particular terminal being the next left-corner of a particular analysis. The terminal may be the left-corner of the top-most non-terminal

[5] A PCFG is a CFG with a probability assigned to each rule, such that the probabilities of all rules expanding a given non-terminal sum to one; specifically, each right-hand side has a probability given the left-hand side of the rule. An additional condition for well-formedness is that the PCFG is consistent or tight, i.e. there is no probability mass lost to infinitely large trees.

[6] The + in (i) denotes concatenation. To avoid confusion between sets and sequences, $\emptyset$ will not be used for empty strings or sequences, rather the symbol $\langle\rangle$ will be used. Note that the script $\mathscr{S}$ is used to denote stacks, while $S^\dagger$ is the start symbol.

```
ABOVE-THRESHOLD(C = (D, 𝒮, P_D, F, w_i^n⟨/s⟩), ℋ_{i+1}, γ, f)
 1  ℋ_{i+1}[0] = (D', 𝒮', P_{D'}, F', w_{i+1}^n⟨/s⟩)      ▷ Heap provides the best scoring entry
 2  if F > P_{D'} * f(γ, |ℋ_{i+1}|)
 3    then return TRUE
 4    else return FALSE

ND-TOP-DOWN-PARSER(G = (V, T, P, S†), ⇒, w = w_0 ... w_n⟨/s⟩, γ, f)
 1  i ← 0
 2  ℋ_i[0] ← (⟨⟩, S†$, 1, 1, w_0^n⟨/s⟩)      ▷ Let ℋ_i be the priority queue for w_i
 3  for i ← 0 to n
 4  do while ABOVE-THRESHOLD(ℋ_i[0], ℋ_{i+1}, γ, f)
 5     do C ← ℋ_i[0] = (D, 𝒮, P_D, F, w_i^n⟨/s⟩)
 6        POP C from ℋ_i
 7        ▷ let X be the top stack symbol on 𝒮
 8        if X ∈ T
 9          then ∀C' such that C ⇒ C' : PUSH C' onto ℋ_{i+1}
10          else ∀C' such that C ⇒ C' : PUSH C' onto ℋ_i
11  ▷ At the end of the string, we must empty the stack to complete the derivation
12  while ABOVE-THRESHOLD(ℋ_{n+1}[0], ℋ_{n+2}, γ, f)
13  do C ← ℋ_{n+1}[0] = (D, 𝒮, P_D, F, ⟨/s⟩)
14     POP C from ℋ_{n+1}
15     ▷ let X be the top stack symbol on 𝒮
16     if X = $
17       then PUSH C onto ℋ_{n+2}
18       else ∀C' such that C ⇒ C' : PUSH C' onto ℋ_{n+1}
19  if EMPTY(ℋ_{n+2}[0])      ▷ if no analysis made the final heap
20    then ERROR
```

Fig. 2. A non-deterministic top-down parsing algorithm, taking a context-free grammar G, a derives relation $\Rightarrow$, an input string w, a base beam-factor γ, and a threshold function f as arguments. The symbol $\Rightarrow$ denotes our derives relation defined above. The symbol ▷ precedes comments.

on the stack of the analysis or it might be the left-corner of the nth non-terminal, after the top $n-1$ non-terminals have rewritten to ϵ. Of course, we cannot expect to have adequate statistics for each non-terminal/word pair that we encounter, so we smooth to the POS. Since we do not know the POS for the word, we must sum the LAP for all POS labels[7].

For a PCFG G, a stack $\mathscr{S} = A_0 \dots A_n\$$ (which we will write $A_0^n\$$) and a look-ahead terminal item w_i, we define the look-ahead probability as follows:

$$\text{LAP}(\mathscr{S}, w_i) = \sum_{\alpha \in (V \cup T)^*} \text{P}_\text{G}(A_0^n \stackrel{*}{\Rightarrow} w_i\alpha) \tag{15}$$

We recursively estimate this with two empirically observed conditional probabilities for every non-terminal A_i: $\widehat{\text{P}}(A_i \stackrel{*}{\Rightarrow} w_i\alpha)$ and $\widehat{\text{P}}(A_i \stackrel{*}{\Rightarrow} \epsilon)$. The same empirical probability, $\widehat{\text{P}}(A_i \stackrel{*}{\Rightarrow} X\alpha)$, is collected for every pre-terminal X as well. The LAP

[7] Equivalently, we can split the analyses at this point, so that there is one POS per analysis. If the POS label is given by the input string, then, obviously, this does not need to occur.

approximation for a given stack state and look-ahead terminal is:

$$\text{(16)} \qquad \mathrm{P_G}(A_j^n \overset{*}{\Rightarrow} w_i\alpha) \approx \mathrm{P_G}(A_j \overset{*}{\Rightarrow} w_i\alpha) + \widehat{\mathrm{P}}(A_j \overset{*}{\Rightarrow} \epsilon)\mathrm{P_G}(A_{j+1}^n \overset{*}{\Rightarrow} w_i\alpha)$$

where

$$\text{(17)} \qquad \mathrm{P_G}(A_j \overset{*}{\Rightarrow} w_i\alpha) \approx \lambda_{A_j}\widehat{\mathrm{P}}(A_j \overset{*}{\Rightarrow} w_i\alpha) + (1-\lambda_{A_j})\sum_{X\in V}\widehat{\mathrm{P}}(A_j \overset{*}{\Rightarrow} X\alpha)\widehat{\mathrm{P}}(X \to w_i)$$

The λ_{A_j} mixing coefficients for interpolation are a function of the frequency of the non-terminal A_j, and are estimated in the standard way using held-out training data (Jelinek and Mercer 1980).

Unknown words are dealt with as follows. Each word in the held-out corpus is assigned an unknown word category, based on certain features of the word. For example, if the word ends in *'s'* it goes into one category; if it ends in *'ing'* it goes in another, etc. A certain amount of probability mass is reserved for the occurrence of these unknown categories as children of the particular part-of-speech tags with which they were observed. Then, when an unknown word is observed in the test set, it is assigned an unknown word category, and the parser treats the word as that category.

The beam threshold at word w_i is a function of the probability of the top ranked candidate analysis, $\mathscr{H}_{i+1}[0]$, on priority queue $\mathscr{H}_{i+1}$ and the number, $|\mathscr{H}_{i+1}|$, of candidate analyses on $\mathscr{H}_{i+1}$. The basic idea is that we want the beam to be very wide if there are few analyses that have been added to $\mathscr{H}_{i+1}$, but relatively narrow if many analyses have been advanced. If $\tilde{p}$ is the probability of the highest ranked analysis on $\mathscr{H}_{i+1}$, then all other analyses are discarded if their probability falls below $\tilde{p}f(\gamma, |\mathscr{H}_{i+1}|)$, where γ is an initial parameter, which we call the *base beam factor*. For the results that will be presented in this paper, γ was 10^{-11}, unless otherwise noted, and $f(\gamma, |\mathscr{H}_{i+1}|) = \gamma|\mathscr{H}_{i+1}|^3$. This function has the effect of having a very wide beam early, but closing fast. If 100 analyses have already been pushed onto $\mathscr{H}_{i+1}$, then a candidate analysis must have a probability above $10^{-5}\tilde{p}$ to avoid being pruned. After 1000 candidate analyses, the beam has narrowed to $10^{-2}\tilde{p}$. There is also a maximum number of allowed analyses on $\mathscr{H}_i$, in case the parse fails to advance an analysis to $\mathscr{H}_{i+1}$. For this study, the maximum was 10,000.

Each derivation step in our top-down parser carries with it the derivation to that point, which can be used to provide relevant conditioning information for future steps in the derivation. We do not use the entire left-context to condition the rule probabilities, but rather 'pick-and-choose' which events in the left-context we would like to condition on. One can think of the conditioning events as functions, which take the partial tree structure as an argument and return a value, upon which the rule probability can be conditioned. Each of these functions is an algorithm for walking the provided tree and returning a value. For example, suppose that we want to condition the probability of the rule $A \to \alpha$. We might write a function that takes the partial tree, finds the parent of the left-hand side of the rule and returns its node label. If the left-hand side has no parent, i.e. it is at the root of the tree, the function returns the null value (NULL). We might write another function that returns the non-terminal label of the closest sibling to the left of A, and NULL if no such node

exists. We can then condition the probability of the production on the values that were returned by the set of functions.

The parser and model that we have presented in this section is that from Roark (2001a). For more detail about the specific probabilistic model, we refer the reader to that paper. For the purposes of this paper, we merely point out that the base probability for this parser is the simple PCFG probability, i.e. the probability of the right-hand side given the left-hand side. With the maximum likelihood (relative frequency) estimator for this PCFG given the training corpus, the probability of an unseen production is zero. We would like to smooth the probabilities in such a way that unseen productions are given some probability mass. The next section will detail one way to do this.

3 Smoothed grammars

3.1 Markov grammars

The basic idea behind smoothing the production probabilities is to forget some of the previous children of a constituent when estimating the probability of the next child. Perhaps the easiest way to see how this would work in practice is through the chain rule. To simplify the notation, let us assume that all rules in the grammar are of the form $A \rightarrow B_0 \ldots B_k$, and that B_0 is always some start symbol, and B_k is always some stop symbol. Since every rule begins with B_0, its probability is always one. A PCFG assigns probability to a rule as follows:

$$P(A \rightarrow B_0 \ldots B_k) = \prod_{i=1}^{k} P(B_i|A, B_0, \ldots, B_{i-1}) \tag{18}$$

A Markov assumption of order n would change each component of the previous equation as follows:

$$P(B_i|A, B_0, \ldots, B_{i-1}) = P(B_i|A, B_{i-n}, \ldots, B_{i-1}) \tag{19}$$

This results in a new decomposition of the probability of a PCFG rule. For example, suppose we chose a Markov grammar of order 1:

$$P(A \rightarrow B_0 \ldots B_k) = \prod_{i=1}^{k} P(B_i|A, B_{i-1}) \tag{20}$$

To get back to our unobserved rules above, the probabilistic grammar estimation for a Markov grammar of order 1 no longer asks how frequently the entire sequence of children has been observed with that parent, but rather how frequently each child has been observed with that previous sibling and that parent.

Another way to see how this might work is via a tree or grammar transformation. Figure 3 shows an NP constituent as it undergoes two transformations. The first is left-factorization, which is a binarization transform that preserves the probability distribution over productions[8]. A smoothed Markov grammar of order n can be

[8] See Roark (2001a) for more detail on this left-factorization transform.

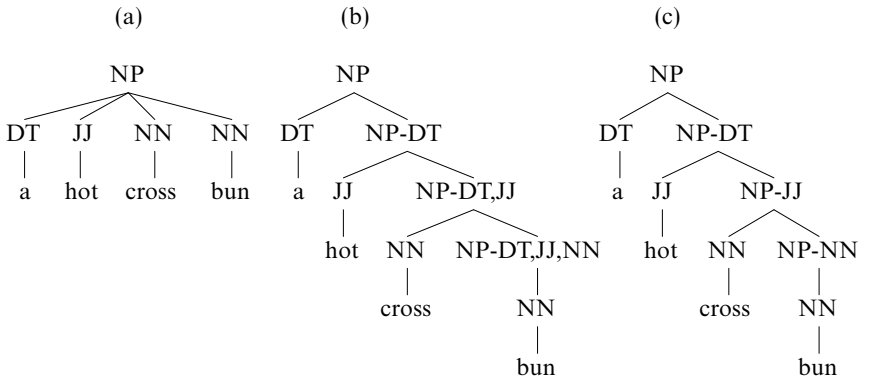


Fig. 3. Noun phrase structures: (a) original flat representation; (b) left-factored tree; and (c) Markov grammar, order 1.

thought of as one in which the non-terminals created by left-factorization are modified to contain only the last n children of the constituent.

The move to estimating PCFG rule probabilities in this manner simplifies some things and complicates others. We will be able to do away with keeping track of specific rules, and simply evaluate the probability of subsequent children as we grow the trees, conditioned on events in the left-context as before, now including some number of previous children in the production. Complicating matters slightly is the fact that constituent head identification is now no longer tied to specific rules, and this will force us to include head identification in our probabilistic model.

We have adopted, following Charniak (2000), a smoothed third-order Markov grammar. This means that a certain amount of probability mass (depending on the smoothing parameters) is reserved for arbitrary permutations of children that have been observed under a particular parent. In estimating the grammar in this way, we have moved from a PCFG with some 15,000 possible productions to one with an infinite number of possible flat productions, i.e. after de-transformation. At each point in the rule expansion, some probability mass is reserved for producing any child that has been observed with that parent. This process can continue indefinitely.

Note, however, that the model as it is here does not absolutely prevent garden pathing with a predictive parser. Suppose that no category on the stack of an analysis has been observed with a child that can have the next input item at its left-corner. Then that analysis will fail to incorporate the next word. We can allow for any non-terminal to be emitted as the child of any other (non-POS) non-terminal by simply adding some ϵ to the count of all possible parent-child pairs. We refer to models that do not add this ϵ as *Smoothed (std)*, and those that do add an ϵ (which cannot garden path) as *Smoothed (any)*.

3.2 Probability model

Equation 20 shows a markov grammar of order 1. In this case, the probability of each child is conditioned on the parent category (i.e. the left-hand side of the CFG rule) and the previous child of that parent. Because of the top-down orientation

of our parsing algorithm, each rule expansion will occur within a fully connected partial parse tree, which will contain all ancestors and siblings to the left of the current category. Since this information is already explicit for the given parse, there is nothing to prevent us conditioning the probability of each child on other features of this left-context. For example, we could choose to condition the probability of the child on not just the left-hand side and previous children of the left-hand side, but also on, say, the parent category of the left-hand side.

In Roark (2001a), the conditional probability was based on a PCFG rule identifier, and other features from the context were added to this. Our conditional probability model is simplified compared with that model, given the uniformity of the conditioning events. Instead of beginning with a PCFG rule identifier, which encodes a composite of the parent and the previous children (recall the non-terminals introduced by left-factorization), then continuing with values returned from tree-walking functions, now all of the conditioning events can be encoded as values returned from tree-walking functions. In addition, our look-ahead probability will also be defined in terms of these tree-walking functions, and our new head probability will also be defined in this way. Hence, our grammar estimation routine now involves simply taking a given set of functions and performing maximum likelihood estimation of the conditioned variable given the values returned from these conditioning functions. This is the same for all three components of our model.

To make this explicit, let us briefly define a small set of treewalking functions, and give the conditional probability models used in the trials that will follow. These functions take a pointer to a node in the tree as an argument. Each node in the tree contains, as a part of its structure, pointers to: (i) its label; (ii) its parent node (*parent*); (iii) its first child node (*child*); (iv) its sibling to the left (*leftsib*); and (v) its designated head child node (*head*). The function then moves the node pointer to other locations in the tree, and returns a value from the final position of the pointer. We use these functions as follows: hypothesize a new arc and node in the tree, and pass the function a pointer to the new node. Hence all values returned from the functions are relative to the newly hypothesized node, the label of which is the conditioned variable. Most of the functions are given additional parameters, so each individual function will be identified by a function name and up to two parameter values. Figures 4 and 5 give the algorithms for the tree-walking functions.

Note that each function returns a single value, but these values are combined as sets of conditioning variables, so that together they can define more complex conditioning contexts. In fact, we can extract values from arbitrary positions in the left-context, and hence create sets of conditioning variables that encode arbitrarily complex tree fragments. The question arises as to where the particular tree-walking functions come from, and how their optimal combination is determined. This is a very interesting issue, and one deserving of further research. For this paper, we used the same features as were used in Roark (2001a) – excepting the difference in simple PCFG versus Markov grammar encoding – which were chosen though a combination of intuition and empirical trials on a development corpus.

```
PAR-SIB(node, m, n)
 1  for i ← 1 to m                          ▷ Move up m nodes
 2  do if node ≠ NULL
 3        then node ← node.parent
 4  for i ← 1 to n                          ▷ Move left n nodes
 5  do if node ≠ NULL
 6        then node ← node.leftsib
 7  if node ≠ NULL
 8     then return node.label
 9     else return NULL

LEFTMOST-PS(node, m, n)
 1  if node.leftsib ≠ NULL                  ▷ Only for leftmost children
 2     then return NULL
 3     else return PAR-SIB(node, m, n)

LEX-HEAD(node, m)
 1  if node ≠ NULL
 2     then node ← node.head                          ▷ Go to a node's head child
 3  while node ≠ NULL and node.child ≠ NULL           ▷ Until node is a leaf
 4  do node ← node.head
 5  for i ← 1 to m                          ▷ Move up m nodes
 6  do if node ≠ NULL
 7        then node ← node.parent
 8  if node ≠ NULL
 9     then return node
10     else return NULL

CURR-HEAD(node, m)
 1  if node = NULL
 2     then return NULL
 3  headnode ← LEX-HEAD(node.parent, m)
 4  if headnode ≠ NULL                      ▷ If parent's head has been found, return it
 5     then return headnode.label
 6     else headnode ← LEX-HEAD(node.leftsib, m)     ▷ Else, left-sibling head
 7          if headnode ≠ NULL
 8             then return headnode.label
 9             else return NULL
```

Fig. 4. Tree-walking functions to return conditioning values for the probability model. *node* is a pointer to a node in the tree, which is a data structure with five fields: *label* which is a pointer to a character string; and *parent*, *child*, *leftsib*, and *head*, which are pointers to other nodes in the tree. The symbol ▷ precedes comments.

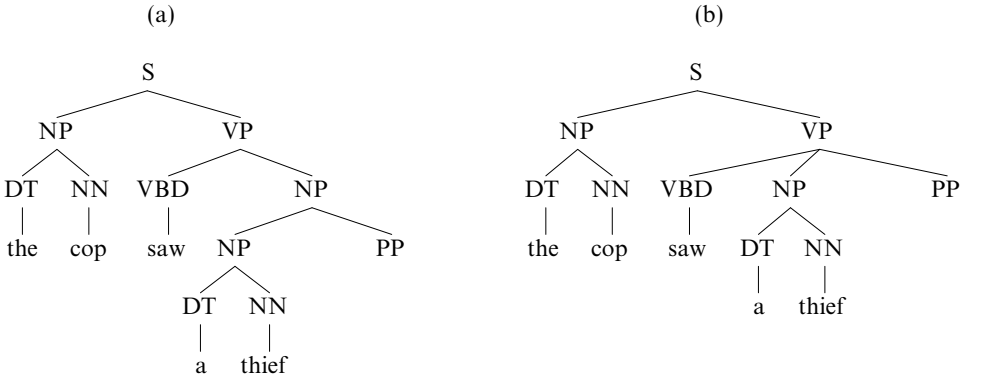
Figures 6 and 7 give the conditional probability models for non-POS expansions and POS expansions, respectively. Each model is a linear order of functions, and the probability of the conditioned event is conditioned on the values returned by these functions. The figures also provide two example trees each, with a newly hypothesized node (the conditioned variable), and the values that would be returned from each of the tree-walking functions. The conditional probability estimate with n features is the linear interpolation of the MLP relative frequency estimate for n

```
LEFT-CCOMMAND(node)
 1  while node ≠ NULL and node.leftsib = NULL        ▷ node is leftmost child
 2  do node ← node.parent
 3  if node = NULL
 4     then return NULL
 5  parenthead ← node.parent.head
 6  if parenthead ≠ NULL        ▷ Go to head of constituent, if found
 7     then node ← parenthead
 8     else node ← node.leftsib        ▷ Else, left-sibling
 9  return node

CC-HEAD(node, m, n)
 1  for i ← 1 to m
 2  do if node ≠ NULL
 3        then node ← LEFT-CCOMMAND(node)
 4  return CURR-HEAD(node, n)

LEFTMOST-CCH(node, m, n)
 1  if node.leftsib ≠ NULL                    ▷ Only for leftmost children
 2     then return NULL
 3     else return CC-HEAD(node, m, n)

CONJ-PARALLEL(node)
 1  if node ≠ NULL and node.leftsib = NULL
 2     then node ← node.parent
 3  if node = NULL
 4     then return NULL
 5  thislabel ← node.label
 6  siblabel ← PAR-SIB(node, 0, 1)
 7  if siblabel = 'CC'        ▷ If parent is being conjoined
 8     then node ← node.leftsib
 9          while node ≠ NULL node.label ≠ thislabel
10          do node ← node.leftsib        ▷ Find first category with same label
11          if node ≠ NULL
12             then node ← node.child
13                  return node.label        ▷ Return label of first child of conjoined node
14  return NULL
```

Fig. 5. More tree-walking functions to return conditioning values for the probability model. *node* is a pointer to a node in the tree, which is a data structure with five fields: *label* which is a pointer to a character string; and *parent*, *child*, *leftsib*, and *head*, which are pointers to other nodes in the tree. The symbol ▷ precedes comments.

features and the conditional probability estimate with $n-1$ features. The order in which these models is presented is the order of interpolation.

The second part of the probability model is the probability that the previous child of the constituent is the head of the constituent. There are three possible cases: (i) the head has already been found to the left of the previous child; (ii) the previous child is the head; or (iii) none of the previous children is the head of the constituent. For example, when the new node is built in the Figure 6a, the probability for (i) above is zero, since the previous child of the NP is the first child; the probabilities

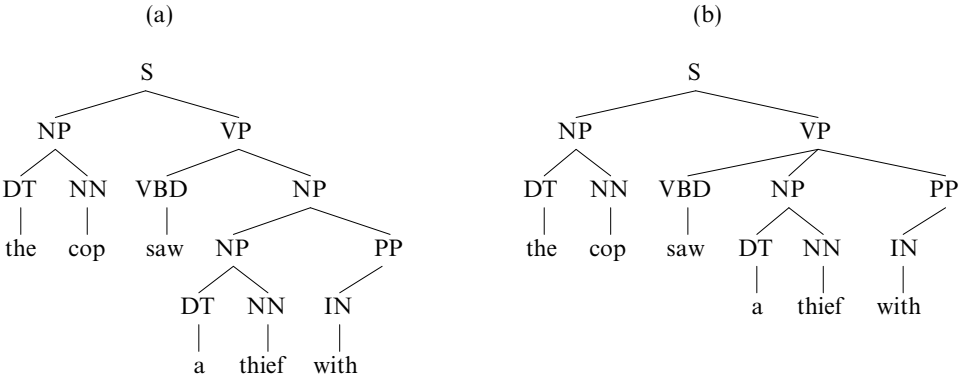


	Conditioning Function	Description	Value Returned Tree (a)	Tree (b)
0	PAR-SIB(*node,1,0*)	Left-hand side (LHS), i.e. parent	NP	VP
1	PAR-SIB(*node,0,1*)	Last child of LHS	NP	NP
2	PAR-SIB(*node,0,2*)	2nd last child of LHS	NULL	VBD
3	PAR-SIB(*node,0,3*)	3rd last child of LHS	NULL	NULL
4	PAR-SIB(*node,2,0*)	Parent of LHS (PAR)	VP	S
5	PAR-SIB(*node,1,1*)	Last child of PAR	VBD	NP
6	PAR-SIB(*node,3,0*)	Parent of PAR (GPAR)	S	NULL
7	PAR-SIB(*node,2,1*)	Last child of GPAR	NP	NULL
8	CONJ-PARALLEL(*node*)	First child of conjoined category	NULL	NULL
9	CURR-HEAD(*node,0*)	Lexical head of current constituent	*thief*	*saw*

Fig. 6. Two trees, to illustrate the tree-walking functions for non-POS expansions. The newly hypothesized node in trees (a) and (b) is the 'PP'. The labels of these new nodes are the conditioned variables.

for (ii) and (iii) must be estimated. The conditioned variable is one of the three above alternatives. The head probability model that we used in these trials consisted entirely of values returned by the PAR-SIB function with the following parameters: (0,1), (1,0), (0,0), (0,2), and (0,3). In words, we are conditioning the head location on: (0) the label of the previous child; (1) the left-hand side (i.e. parent label of the newly hypothesized node); (2) the label of the newly hypothesized node; (3) the label of the second child to the left; and (4) the label of the thirrd child to the left. Once the head is identified as the previous child, that selection is fixed for that candidate analysis from that point forward. For every rule expansion, more than one analysis must be considered, depending on the range of possibilities for head assignment.

One possible concern would be the use of the new node label to condition the head probability, and also the lexical head of the constituent to condition the probability of the new node label. The way that the CURR-HEAD function is defined however,



Conditioning Function		Description	Value Returned Tree (a)	Tree (b)
0	PAR-SIB(*node,1,0*)	Left-hand side (LHS), i.e. parent	IN	IN
1	PAR-SIB(*node,2,0*)	Parent of LHS (PAR)	PP	PP
2	PAR-SIB(*node,1,1*)	Last child of PAR	NULL	NULL
3	LEFTMOST-PS(*node,3,0*)	Parent of PAR (GPAR)	NP	VP
4	LEFTMOST-CCH(*node,1,1*)	POS of C-Commanding head	NN	VBD
5	CC-HEAD(*node,1,0*)	C-Commanding head	*thief*	*saw*
6	CC-HEAD(*node,2,0*)	Next C-Commanding head	*saw*	*cop*

Fig. 7. Two trees, to illustrate the tree-walking functions for POS expansions. The newly hypothesized node in both trees (a) and (b) is the word 'with'. The labels of these new nodes are the conditioned variables.

is that, if the head of the constituent has not been assigned yet, it selects the head of the previous child. Hence the head probability can be evaluated after the rule expansion probability.

The Look-Ahead Probability (LAP) is defined in exactly the way it was in the previous section, except that instead of being conditioned on the composite category created through factorization, it is conditioned on the label of the current category and the three previously emitted children.

4 Empirical results

Evaluation is carried out on a hand-parsed test corpus, which was created using the Penn Treebank II annotation guidelines (Bies et al. 1995), and the manual parses are treated as correct. We will call the manual parse GOLD and the parse that the parser returns TEST. Precision is the number of common constituents in GOLD and TEST divided by the number of constituents in TEST. Recall is the number of common constituents in GOLD and TEST divided by the number of constituents in GOLD. Following standard practice, we will be reporting scores only for non-part-of-speech

Table 1. *Parsing results using the conditional probability model from Roark (2001a), with (i) a PCFG backbone, from Roark (2001a); (ii) a standard smoothed Markov grammar of order 3, and (iii) a Markov grammar of order 3 with probability mass reserved for any non-terminal being emitted. Results are trained on sections 2-21 and tested on section 23 of the Penn* Wall St. Journal *Treebank*

Grammar	LR	LP	CB	0 CB	≤ 2 CB	Pct. failed	Avg. rule expansions considered†	Average analyses advanced†
section 23: 2416 sentences of length ≤ 100								
PCFG	85.7	85.7	1.41	59.0	79.9	1.7	6,709	207.6
Smoothed (std)	86.4	86.8	1.31	59.5	81.6	0	9,008	198.9
Smoothed (any)	86.4	86.8	1.32	59.8	81.5	0	16,378	199.0

†per word

constituents, which are called Labeled Recall (LR) and Labeled Precision (LP). Also following standard practice, we will ignore punctuation altogether, and treat ADVP and PRN as equivalent.

LR and LP are part of the standard set of PARSEVAL measures of parser quality (Black, Abney, Flickinger *et al*., 1991). We will also include, from this set of measures, the crossing bracket scores: average Crossing Brackets (CB), percentage of sentences with no crossing brackets (0 CB), and the percentage of sentences with two crossing brackets or fewer ($\leq$ 2 CB). In addition, to measure efficiency, we will show the average number of rule expansions considered per word, i.e. the number of rule expansions for which a probability was calculated (see Roark and Charniak, 2000), and the average number of analyses advanced to the next priority queue per word.

This is an incremental parser with a pruning strategy and no backtracking. In such a model, it is possible to commit to a set of partial analyses at a particular point that cannot be completed given the rest of the input string (i.e. the parser can *garden path*). In such a case, the parser fails to return a complete parse. In the event that no complete parse is found, the highest initially ranked parse on the last non-empty priority queue is returned. All unattached words are then attached at the highest level in the tree. In such a way we predict no new constituents and all incomplete constituents are closed. This structure is evaluated for precision and recall, which is entirely appropriate for these incomplete as well as complete parses. If we fail to identify nodes later in the parse, the recall will suffer, and if our early predictions were bad, both precision and recall will suffer. Of course, the percentage of these failures are reported as well.

The grammars were induced from sections 2-21 of the Penn *Wall St. Journal* Treebank (Marcus, Santorini and Marcinkiewicz 1993), and tested on section 23. Table 1 gives results using two variants of the smoothed grammar model, along with the results with these same conditioning features from Roark (2001a)[9].

[9] The conditional probability models that are presented here are, with the exception of the Markov grammar smoothing, identical to the models used in Roark (2001a).

Table 2. *Parsing results using the new conditional probability model (Smoothed (std), with a variety of base beam factors. Results are trained on sections 2-21 and tested on section 23 of the Penn* Wall St. Journal *Treebank (†per word)*

Base Beam Factor	LR	LP	CB	0 CB	≤ 2 CB	Pct. failed	Avg. rule expansions considered†	Average analyses advanced†
section 23: 2416 sentences of length ≤ 100								
10^{-11}	86.4	86.8	1.31	59.5	81.6	0	9,008	198.9
10^{-10}	86.2	86.5	1.34	59.2	81.4	0	5,528	120.0
10^{-9}	86.1	86.4	1.36	59.0	81.1	0	3,439	72.6
10^{-8}	85.6	85.9	1.41	58.3	80.3	0	2,159	43.9
10^{-7}	85.3	85.0	1.49	56.8	79.3	0	1,374	26.6
10^{-6}	84.2	84.5	1.59	55.3	77.9	0	898	16.2

Even though the conditioning events are the same, our accuracy improves by nearly one percentage point, while our coverage goes to 100%. The rule expansions considered for the same beam definition increases by a third, but this is hardly surprising. The number of productions in the original form (i.e. after being de-transformed) that now have probability mass is infinite, as opposed to the previous, unsmoothed grammar of about 15,000 rules. Note that adding an ϵ to the counts for all parent-children pairs (the *Smoothed (any)* model) considers many more candidates, yet the distribution is still peaked enough to give the same level of accuracy. For this corpus, it appears that only observed children need to be considered as children of a particular parent.

Table 2 gives results with a variety of base beam factors. Recall that the beam threshold is defined as a variable probability range. For a given base beam factor γ, we define the beam as $\gamma|\mathscr{H}_{i+1}|^3$, i.e. the range narrows with the cube of the number of analyses advanced. The results in Table 2 indicate that, with the new model, the beam can be greatly narrowed without losing much accuracy, and maintaining complete coverage. At a $\gamma = 10^{-9}$, the parser loses less than half a point of either precision or recall, while considering fewer than forty percent of the rule expansions that were considered at the widest beam. Recall that this measure correlates nearly perfectly with time (Roark and Charniak 2000), so there is an equivalent speedup[10].

5 Parsing transcribed speech

Up to this point, we have been parsing edited newspaper text. One of the benefits of our parsing approach has been the application of our probabilistic parser as a language model for statistical speech recognition (Roark 2001a). The ultimate applicability of the methods that we will describe depends on whether or not

[10] At the widest beam, on a 700 Mhz Pentium III processor running linux, the parser processes the *Wall St. Journal* test set in on average 9.6 words per second.

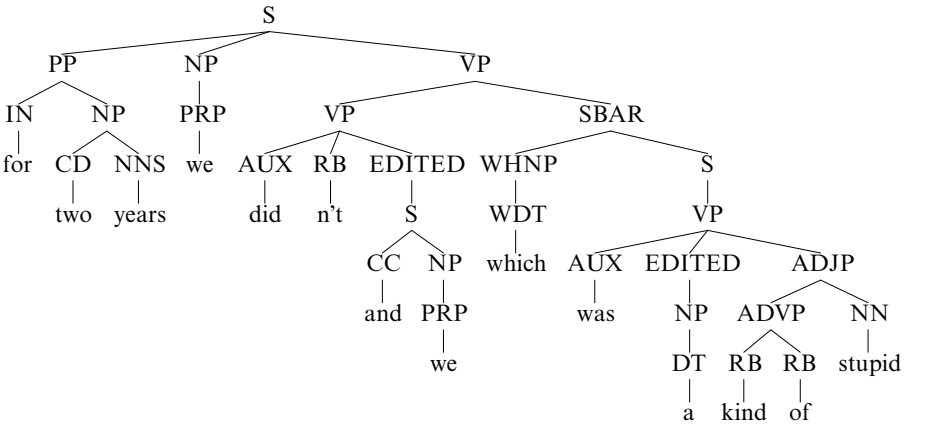


Fig. 8. A sample parse tree from the Penn Switchboard Treebank.

a parser such as this can effectively parse spontaneous speech. This section will examine parsing spontaneous telephone conversation transcripts.

As we have seen, treebank parsers can be amazingly effective on edited newspaper text. Parsing spontaneous speech, however, is a different matter. False starts, sentence and word fragments, and ungrammaticality are quite common, all of which, needless to say, pose a problem for any parser, but particularly for a statistical parser trained on written, edited text. The release of a new Penn Treebank version, including a large treebank of Switchboard telephone speech, is thus a great opportunity for examining how well treebank techniques can be made to handle these kinds of phenomena. It was viewing this treebank that spurred us to investigate the smoothed Markov grammar approach presented in the previous section.

Figure 8 gives an example parse tree from the new treebank. There is a new non-terminal, 'EDITED', which is used for false starts. For example, in the tree in Figure 8, a conjoined clause was begun (*'and we'*), but the VP is then continued with a subordinate clause. The words in the falsely started clause are placed under an EDITED constituent, with as much internal structure as is evident from the input. A second false start occurs further along in the string.

These EDITED nodes provide a way to fit disfluencies into a parse structure, and hence we can apply a parser trained on a treebank of such trees directly to strings of spontaneous speech, without pre-processing. In other words, we will model the joint probability of the word string and the disfluency labels directly by including the disfluency labels in our PCFG. There have been other approaches to modeling the joint probability of disfluency labels and words, including Stolcke, Shriberg, Hakkani-Tür and Tür (1999) and Heeman and Allen (1999). These approaches, however, differ from the current model in that they used finite-state part-of-speech taggers to model the hidden structure, rather than a PCFG. They also differ with respect to the kinds of features that were used in their model. For example, Stolcke *et al.* (1999) used prosodic features in their HMM to help predict their hidden events. It is likely that the same kinds of prosodic features that proved useful in that model could be profitably included in the current model to improve disambiguation.

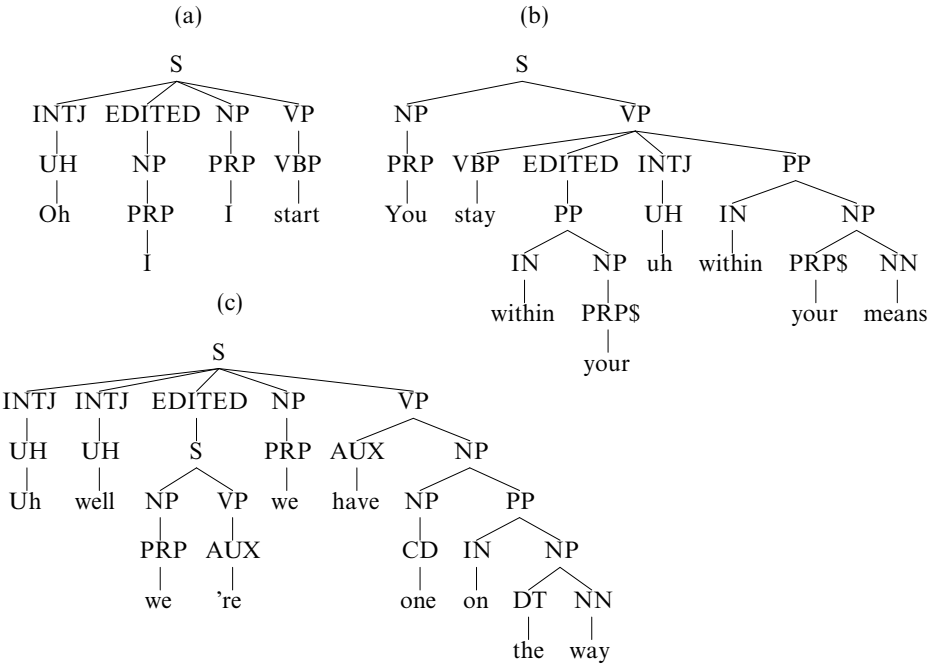


Fig. 9. Typical disfluencies from the Switchboard treebank.

Following Charniak and Johnson (2001), we designated all of sections 2 and 3 (92,536 sentences, 945,294 words) as the training corpus; files sw4004 through sw4153 (6,051 sentences, 67,050 words) as the test corpus; files sw4154 through sw4483 (6,021 sentences, 68,543 words) as the held out corpus; and files sw4519 through sw4936 (5895 sentences, 69,597 words) as the development corpus. These transcriptions have had punctuation inserted by annotators, to delimit interjections and false starts. Hence, a sentence such as

(21) *'Uh well we 're we have one on the way'*

was transcribed

(22) *'Uh , well we 're , we have one on the way.*

The word counts above include punctuation.

The first thing we tried was to simply leave the model as it was for the *Wall St. Journal* parsing trials, train on the new treebank in exactly the same way, parse and evaluate[11]. It is clear, however, by inspecting some of the disfluencies found in the corpus, that there is some relationship between the false start and what replaces it. Figure 9 shows two common kinds of false starts, in which the EDITED node is either followed by the identical constituent, or by a very similar constituent. In the same way that we were able to effectively model parallelism in conjoined

[11] These trials will be reported a bit later in the section.

```
EDIT-SKIP(node)
 1  if PUNCTUATION(node)      ▷ If punctuation
 2     then return TRUE
 3  if node.label = 'PRN' or node.label = 'INTJ'    ▷ If a parenthetical or interjection
 4     then return TRUE
 5  return FALSE

EDIT-CHILD(node)
 1  if node ≠ NULL and node.leftsib = NULL and node.parent ≠ NULL
 2     then node ← node.parent
 3     else return NULL
 4  thislabel ← node.label                    ▷ Save left-hand side label
 5  node ← node.leftsib
 6  while node ≠ NULL and EDIT-SKIP(node)
 7  do node ← node.leftsib                    ▷ Move left past skip categories
 8  if node = NULL or node.label ≠ 'EDITED'
 9     then return NULL
10  parentlabel ← node.parent.label                ▷ Save parent label
11  node ← node.child                                 ▷ Move to first child
12  if node.label = parentlabel                ▷ If same label as parent, keep going
13     then node ← node.child
14  if node.label ≠ thislabel                  ▷ If category is not same as left-hand side
15     then return NULL
16  node ← node.child                                 ▷ Go to first child
17  if node = NULL
18     then return NULL
19     else return node.label
```

Fig. 10. Functions for conditioning probabilities so as to capture EDITED node parallelism. *node* is a pointer to a node in the tree, which is a data structure with five fields: *label* which is a pointer to a character string; and *parent*, *child*, *leftsib*, and *head*, which are pointers to other nodes in the tree. The symbol ▷ precedes comments.

constituents, we could condition the probability of structures on values returned from a function looking at these EDITED constituents.

There are two observations that we could easily exploit to model the likelihood of an EDITED constituent: (i) the first child of the first constituent *after* the EDITED constituent tends to be the same as the first child of the category *inside* of the EDITED constituent; and (ii) the first word after the EDITED constituent tends to be the same as the first word of the EDITED constituent. There are a couple of provisos that need to made to these observations. First, interjections (e.g. *'uh'*) and parentheticals (e.g. *'you know'* or *'I mean'*) as well as punctuation frequently stand between the disfluency and the continuation. Hence, any algorithm that wants to link a disfluency and its continuation should skip these categories. Second, as evidenced by the disfluency in Figure 9c, the category directly under the EDITED node may not be the next produced category, but rather the category under which the disfluency occurs. In this case, to get the parallelism, one must look at the category under the EDITED node, and, if it is the same as the parent of the EDITED node, go to its first child to find the parallel constituent.

```
EDIT-LEX(node, m)
 1  while node ≠ NULL and node.leftsib = NULL        ▷ Move up left-child chain
 2  do node ← node.parent
 3  if node = NULL
 4     then return NULL
 5  node ← node.leftsib
 6  while node ≠ NULL and EDIT-SKIP(node)
 7  do node ← node.leftsib                  ▷ Move left past skip categories
 8  if node = NULL or node.label ≠ 'EDITED'
 9     then return NULL
10  while node.child ≠ NULL
11  do node ← node.child                ▷ Move down to the left-corner
12  for i ← 1 to m                                  ▷ Move up m nodes
13  do if node ≠ NULL
14        then node ← node.parent
15  if node ≠ NULL
16     then return node
17     else return NULL
```

Fig. 11. Another function for conditioning probabilities so as to capture EDITED node parallelism. *node* is a pointer to a node in the tree, which is a data structure with five fields: *label* which is a pointer to a character string; and *parent*, *child*, *leftsib*, and *head*, which are pointers to other nodes in the tree. The symbol ▷ precedes comments.

Table 3. *Parsing results using the model from previous sections, and the new model with functions to condition probabilities on EDITED node parallelism. The parser was trained on sections 2 and 3 of the switchboard treebank, and tested on files sw4004 through sw4153 (†per word)*

Model	LR	LP	CB	0 CB	≤ 2 CB	Pct. failed	Avg. rule expansions considered†	Average analyses advanced†
sw4004-sw4153: 6051 sentences of length ≤ 100								
WSJ Model	84.6	85.2	0.64	83.6	92.1	0	9,987	176.6
w/ EDITED fnct	85.2	85.6	0.61	83.8	92.4	0	9,562	170.1
and no punct	84.0	84.6	0.68	82.1	91.5	0.08	12,051	182.2

Two new functions were written, one to match the first constituent following the EDITED node with a constituent under the EDITED node, and condition its expansion on the first child of this edited category (EDIT-CHILD). This function is presented in Figure 10. The next is to condition the probability of the first lexical item after an EDITED node with the left-corner lexical item in the EDITED node (EDIT-LEX). This function is presented in Figure 11. The revised conditional probability models are presented in Figure 12.

Table 3 gives parsing results both with the conditional probability models from the previous sections, and with the new EDITED node functions. Since punctuation is added by annotators, and not in the speech stream, we also ran a trial with the new EDITED node functions, with punctuation removed, to see the effect on

Conditional Probability Model for non-POS expansions

Conditioning Function		Description
0	PAR-SIB(*node,1,0*)	Left-hand side (LHS)
1	PAR-SIB(*node,0,1*)	Last child of LHS
2	PAR-SIB(*node,0,2*)	2nd last child of LHS
3	PAR-SIB(*node,0,3*)	3rd last child of LHS
4	PAR-SIB(*node,2,0*)	Parent of LHS (PAR)
5	EDIT-CHILD(*node*)	First Child of category under EDITED node
6	PAR-SIB(*node,1,1*)	Last child of PAR
7	PAR-SIB(*node,3,0*)	Parent of PAR (GPAR)
8	PAR-SIB(*node,2,1*)	Last child of GPAR
9	CONJ-PARALLEL(*node*)	First child of conjoined category
10	CURR-HEAD(*node,0*)	Lexical head of current constituent

Conditional Probability Model for POS expansions

Conditioning Function		Description
0	PAR-SIB(*node,1,0*)	Left-hand side (LHS)
1	PAR-SIB(*node,2,0*)	Parent of LHS (PAR)
2	PAR-SIB(*node,1,1*)	Last child of PAR
3	EDIT-LEX(*node,1*)	POS of EDITED left-corner lexical item
4	EDIT-LEX(*node,0*)	EDITED left-corner lexical item
5	LEFTMOST-PS(*node,3,0*)	Parent of PAR (GPAR)
6	LEFTMOST-CCH(*node,1,1*)	POS of C-Commanding head
7	CC-HEAD(*node,1,0*)	C-Commanding head
8	CC-HEAD(*node,2,0*)	Next C-Commanding head

Fig. 12. The modified conditional probability models used for Switchboard parsing. These are identical to those in Figures 6 and 7, except for the EDIT-CHILD and EDIT-LEX functions.

accuracy. The look-ahead and head probability models remained the same as in the previous section. Overall, all models do pretty well. The new functions provide a half a percentage point improvement in accuracy, and about a four percent decrease in expansions considered. As far as correctly finding EDITED nodes, the old model, inherited from the *Wall St. Journal* parser, gets 56.5% recall and 67.0% precision for these nodes; the new model gets 63.5% recall and 71.0% precision. Thus our new functions do seem to be buying us some improvement in detecting disfluencies, which translates to overall accuracy improvements. Removing punctuation hurt much less than has been reported for the *Wall St. Journal* Treebank (Roark 2001a), which bodes well for parsing the output of a speech recognizer.

Table 4 gives the performance with our Switchboard conditional probability model at a variety of base beam factors. It would be surprising if the same parameterization worked equally well both for Wall St. Journal text and spoken language. Indeed,

Table 4. *Parsing results using the new model with functions to condition probabilities on EDITED node parallelism, at various base beam factors. The parser was trained on sections 2 and 3 of the switchboard treebank, and tested on files sw4004 through sw4153 (†per word)*

Base Beam Factor	LR	LP	CB	0 CB	≤ 2 CB	Pct. failed	Avg. rule expansions considered†	Average analyses advanced†
sw4004-sw4153: 6051 sentences of length ≤ 100								
10^{-11}	85.2	85.6	0.61	83.8	92.4	0	9,562	170.1
10^{-10}	85.1	85.6	0.61	83.9	92.5	0	5,799	101.7
10^{-9}	85.0	85.5	0.61	83.8	92.4	0	3,574	61.1
10^{-8}	84.8	85.4	0.62	83.7	92.3	0	2,219	36.8
10^{-7}	84.4	85.0	0.64	83.5	92.1	0	1,404	22.4
10^{-6}	83.5	84.1	0.68	82.7	91.9	0.02	915	13.6

between 10^{-9} and 10^{-11} there is virtually no difference in accuracy, yet there is a 60% reduction in the number of rule expansions considered[12].

Note that at the narrowest beam tested, 10^{-6}, one sentence fails to find a parse, i.e. the parser garden paths. For these models we are using the *Smoothed (std)* model, where only categories observed as children of a particular parent have non-zero probability of being children of that parent. If failing one sentence out of 6,000 is enough of a problem for a particular application, the more permissive model, which does not garden path, can be used, perhaps in response to failure.

Since the Switchboard treebank is relatively new, there is only one other parsing result that we are aware of, to which we could compare these results. This is the two-stage architecture presented in Charniak and Johnson (2001), which first runs a high-precision classifier to decide whether lexical items are EDITED. If they are, then they are removed for input into a statistical parser. The labeled precision and recall percentages presented in that paper were measured according to a special definition, which has three modifications to the definition that we have been using until now. First, all internal structure of EDITED nodes is removed, creating flat constituents. Secondly, two EDITED nodes with no intermediate non-EDITED material are merged. Thirdly, the beginning and ending positions of the EDITED constituents are treated as equivalent for scoring purposes (like punctuation).

Table 5 presents their results and ours with this modified precision and recall metric. We present precision and recall, as well as the F-measure, since the precision and recall can be rather far apart. We also measured the performance of our parser just with respect to these modified EDITED nodes. These results indicate that we do pretty well on the internal structure of edited nodes, so that our performance drops somewhat when that structure is omitted. With the pre-processing, the Charniak

[12] The expansions considered metric is directly proportional to time, and provides a machine-independent metric, as argued in Roark and Charniak (2000).

Table 5. *Results using the Charniak and Johnson modified labeled precision and recall metric, of their parser, our parser, and EDITED nodes from our parser*

Parser	LR	LP	F-measure
Charniak and Johnson	86.5	85.3	85.9
Our parser	84.7	84.9	84.8
Our EDITED nodes	63.9	67.5	65.6

parser outperforms ours by a point and a half. This difference is unsurprising, since that model, of course, is not incremental, and hence can capture certain dependencies that elude ours.

To summarize this section, we have taken our existing parser and applied it unmodified to transcribed speech with quite good results. With the additional conditioning information, we eke out an additional half a point of accuracy, and it even performs fairly well in the absence of punctuation.

6 Conclusion

In this paper, we have presented a relatively simple change to a PCFG model, which results in complete coverage when used by a top-down parser. Reserving an appropriate amount of probability mass for unobserved events is essential for predictive parsing of this sort. The model that we have presented uses standard smoothing techniques to reserve just enough probability mass for unobserved events – not so much that the original peaked distribution fails to favor the better parses, but enough so that in situations that were previously failures (either catastrophic garden paths or recoverable but highly unlikely derivations) better derivations can receive appropriate probability mass. We have shown that this can be done without sacrificing efficiency or accuracy, and without resorting to pre-defined recovery heuristics.

Most of the results presented in this paper were produced for the author's Ph.D. thesis. That thesis contains a host of experiments with these models, including some which demonstrate perplexity and word error rate reduction over other language models. Due to space constraints, the reader is referred to Roark (2001b) for those results.

Acknowledgements

Much of the work that is presented here was done while a PhD student in the Department of Cognitive and Linguistic Sciences at Brown University. The author would like to thank Mark Johnson for support and discussion during the course of this work. Thanks also to Eugene Charniak, Fred Jelinek and Julie Sedivy for reading and commenting on the work. This research was supported in part by NSF IGERT grant #DGE-9870676.

References

Aho, A. V., Sethi, R. and Ullman, J. D. (1986) *Compilers, Principles, Techniques, and Tools.* Addison-Wesley.

Bies, A., Ferguson, M., Katz, K., MacIntyre, R., Tredinnick, V., Kim, G., Marcinkiewicz, M. A. and Schasberger, B. (1995) Bracketing guidelines for treebank ii style penn tree bank project. Technical Report, CIS, University of Pennsylvania.

Black, E., Abney, S., Flickenger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M. P., Roukos, S., Santorini, B. and Strzalkowski, T. (1991) A procedure for quantitatively comparing the syntactic coverage of english grammars. *DARPA Speech and Natural Language Workshop*, pp. 306–311.

Charniak, E. (2000) A maximum-entropy-inspired parser. *Proceedings 1st Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 132–139.

Charniak, E. and Johnson, M. (2001) Edit detection and parsing for transcribed speech. *Proceedings 2nd Conference of the North American Chapter of the Association for Computational Linguistics.*

Collins, M. J. (1997) Three generative, lexicalised models for statistical parsing. *Proceedings 35th Annual Meeting of the Association for Computational Linguistics*, pp. 16–23.

Corman, T. H., Leiserson, C. E. and Rivest, R. L. (1990) *Introduction to Algorithms.* MIT Press.

Heeman, P. A. and Allen, J. F. (1999) Speech repairs, intonational phrases, and discourse markers: Modeling speakers' utterances in spoken dialogue. *Computational Linguistics* **25**(4): 527–571.

Jelinek, F. and Mercer, R. L. (1980) Interpolated estimation of markov source parameters from sparse data. *Proceedings Workshop on Pattern Recognition in Practice*, pp. 381–397.

Johnson, M. (1998) PCFG models of linguistic tree representations. *Computational Linguistics* **24**(4): 617–636.

Marcus, M. P., Santorini, B. and Marcinkiewicz, M. A. (1993) Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics* **19**(2): 313–330.

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

Roark, B. (2001b) *Robust Probabilistic Predictive Syntactic Processing*. PhD thesis, Brown University. (`http://arXiv.org/abs/cs/0105019`.)

Roark, B. and Charniak, E. (2000) Measuring efficiency in high-accuracy, broad-coverage statistical parsing. *Proceedings COLING-2000 Workshop on Efficiency in Large-scale Parsing Systems*, pp. 29–36.

Stolcke, A., Shriberg, E., Hakkani-Tür, D. and Tür, G. (1999) Modeling the prosody of hidden events for improved word recognition. *Proceedings of Eurospeech*, pp. 307–310.