Probabilistic Top-Down Parsing and Language Modeling

Brian Roark*
Brown University

This paper describes the functioning of a broad-coverage probabilistic top-down parser, and its application to the problem of language modeling for speech recognition. The paper first introduces key notions in language modeling and probabilistic parsing, and briefly reviews some previous approaches to using syntactic structure for language modeling. A lexicalized probabilistic top-down parser is then presented, which performs very well, in terms of both the accuracy of returned parses and the efficiency with which they are found, relative to the best broad-coverage statistical parsers. A new language model that utilizes probabilistic top-down parsing is then outlined, and empirical results show that it improves upon previous work in test corpus perplexity. Interpolation with a trigram model yields an exceptional improvement relative to the improvement observed by other models, demonstrating the degree to which the information captured by our parsing model is orthogonal to that captured by a trigram model. A small recognition experiment also demonstrates the utility of the model.

1. Introduction

With certain exceptions, computational linguists have in the past generally formed a separate research community from speech recognition researchers, despite some obvious overlap of interest. Perhaps one reason for this is that, until relatively recently, few methods have come out of the natural language processing community that were shown to improve upon the very simple language models still standardly in use in speech recognition systems. In the past few years, however, some improvements have been made over these language models through the use of statistical methods of natural language processing, and the development of innovative, linguistically well-motivated techniques for improving language models for speech recognition is generating more interest among computational linguists. While language models built around shallow local dependencies are still the standard in state-of-the-art speech recognition systems, there is reason to hope that better language models can and will be developed by computational linguists for this task.

This paper will examine language modeling for speech recognition from a natural language processing point of view. Some of the recent literature investigating approaches that use syntactic structure in an attempt to capture long-distance dependencies for language modeling will be reviewed. A new language model, based on probabilistic top-down parsing, will be outlined and compared with the previous literature, and extensive empirical results will be presented which demonstrate its utility.

Two features of our top-down parsing approach will emerge as key to its success. First, the top-down parsing algorithm builds a set of **rooted** candidate parse trees from left to right over the string, which allows it to calculate a generative probability for

* Department of Cognitive and Linguistic Sciences, Box 1978, Brown University, Providence, RI 02912

each prefix string from the probabilistic grammar, and hence a conditional probability for each word given the previous words and the probabilistic grammar. A left-to-right parser whose derivations are not rooted, i.e., with derivations that can consist of disconnected tree fragments, such as an LR or shift-reduce parser, cannot incrementally calculate the probability of each prefix string being generated by the probabilistic grammar, because their derivations include probability mass from unrooted structures. Only at the point when their derivations become rooted (at the end of the string) can generative string probabilities be calculated from the grammar. These parsers can calculate word probabilities based upon the parser state—as in Chelba and Jelinek (1998a)—but such a distribution is not generative from the probabilistic grammar.

A parser that is not left to right, but which has rooted derivations, e.g., a head-first parser, will be able to calculate generative joint probabilities for entire strings; however, it will not be able to calculate probabilities for each word conditioned on previously generated words, unless each derivation generates the words in the string in exactly the same order. For example, suppose that there are two possible verbs that could be the head of a sentence. For a head-first parser, some derivations will have the first verb as the head of the sentence, and the second verb will be generated after the first; hence the second verb's probability will be conditioned on the first verb. Other derivations will have the second verb as the head of the sentence, and the first verb's probability will be conditioned on the second verb. In such a scenario, there is no way to decompose the joint probability calculated from the set of derivations into the product of conditional probabilities using the chain rule. Of course, the joint probability can be used as a language model, but it cannot be interpolated on a word-by-word basis with, say, a trigram model, which we will demonstrate is a useful thing to do.

Thus, our top-down parser allows for the incremental calculation of generative conditional word probabilities, a property it shares with other left-to-right parsers with rooted derivations such as Earley parsers (Earley 1970) or left-corner parsers (Rosenkrantz and Lewis II 1970).

A second key feature of our approach is that top-down guidance improves the efficiency of the search as more and more conditioning events are extracted from the derivation for use in the probabilistic model. Because the rooted partial derivation is fully connected, all of the conditioning information that might be extracted from the top-down left context has already been specified, and a conditional probability model built on this information will not impose any additional burden on the search. In contrast, an Earley or left-corner parser will underspecify certain connections between constituents in the left context, and if some of the underspecified information is used in the conditional probability model, it will have to become specified. Of course, this can be done, but at the expense of search efficiency; the more that this is done, the less benefit there is from the underspecification. A top-down parser will, in contrast, derive an efficiency benefit from precisely the information that is underspecified in these other approaches.

Thus, our top-down parser makes it very easy to condition the probabilistic grammar on an arbitrary number of values extracted from the rooted, fully specified derivation. This has lead us to a formulation of the conditional probability model in terms of values returned from tree-walking functions that themselves are contextually sensitive. The top-down guidance that is provided makes this approach quite efficient in practice.

The following section will provide some background in probabilistic context-free grammars and language modeling for speech recognition. There will also be a brief review of previous work using syntactic information for language modeling, before we introduce our model in Section 4.

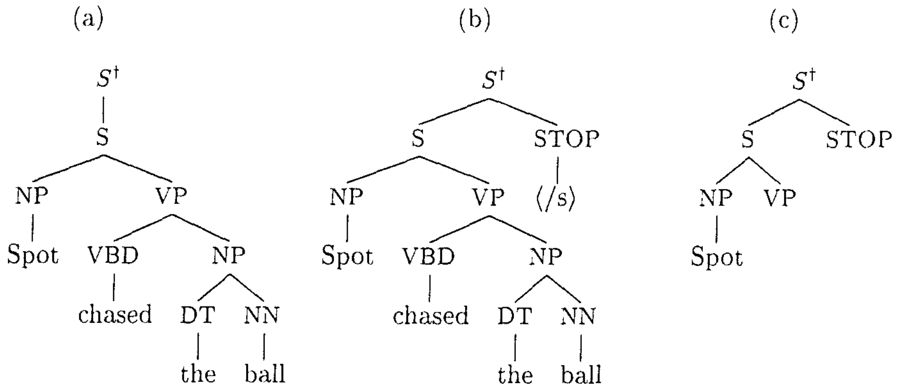

Figure 1
Three parse trees: (a) a complete parse tree; (b) a complete parse tree with an explicit stop symbol; and (c) a partial parse tree.

2. Background

2.1 Grammars and Trees

This section will introduce probabilistic (or stochastic) context-free grammars (PCFGs), as well as such notions as complete and partial parse trees, which will be important in defining our language model later in the paper.[1] In addition, we will explain some simple grammar transformations that will be used. Finally, we will explain the notion of c-command, which will be used extensively later as well.

PCFGs model the syntactic combinatorics of a language by extending conventional context-free grammars (CFGs). A CFG $G = (V, T, P, S^\dagger)$, consists of a set of nonterminal 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 $\alpha \in (V \cup T)^*$. These context-free rules can be interpreted as saying that a nonterminal symbol A expands into one or more either nonterminal or terminal symbols, $\alpha = X_0 \ldots X_k$.[2] A sequence of context-free rule expansions can be represented in a tree, with parents expanding into one or more children below them in the tree. Each of the individual local expansions in the tree is a rule in the CFG. Nodes in the tree with no children are called leaves. A tree whose leaves consist entirely of terminal symbols is complete. Consider, for example, the parse tree shown in (a) in Figure 1: the start symbol is $S^\dagger$, which expands into an S. The S node expands into an NP followed by a VP. These nonterminal nodes each in turn expand, and this process of expansion continues until the tree generates the terminal string, "Spot chased the ball", as leaves.

A CFG G defines a language L_G, which is a subset of the set of strings of terminal symbols, including only those that are leaves of complete trees rooted at $S^\dagger$, built with rules from the grammar G. We will denote strings either as w or as $w_0 w_1 \ldots w_n$, where w_n is understood to be the last terminal symbol in the string. For simplicity in displaying equations, from this point forward let w_i^j be the substring $w_i \ldots w_j$. Let $T_{w_0^n}$

[1] For a detailed introduction to PCFGs, see Manning and Schütze (1999), for example.
[2] For ease of exposition, we will ignore epsilon productions for now. An epsilon production has the empty string (ϵ) on the right-hand side, and can be written $A \rightarrow \epsilon$. Everything that is said here can be straightforwardly extended to include such productions.

be the set of all complete trees rooted at the start symbol, with the string of terminals w_0^n as leaves. We call $T_{w_0^n}$ the set of complete parses of w_0^n.

A PCFG is a CFG with a probability assigned to each rule; specifically, each right-hand side has a probability given the left-hand side of the rule. The probability of a parse tree is the product of the probabilities of each rule in the tree. Provided a PCFG is consistent (or tight), which it always will be in the approach we will be advocating, this defines a proper probability distribution over completed trees.[3]

A PCFG also defines a probability distribution over strings of words (terminals) in the following way:

$$\mathrm{P}(w_0^n) = \sum_{t \in T_{w_0^n}} \mathrm{P}(t) \tag{1}$$

The intuition behind Equation 1 is that, if a string is generated by the PCFG, then it will be produced if and only if one of the trees in the set $T_{w_0^n}$ generated it. Thus the probability of the string is the probability of the set $T_{w_0^n}$, i.e., the sum of its members' probabilities.

Up to this point, we have been discussing strings of words without specifying whether they are "complete" strings or not. We will adopt the convention that an explicit beginning of string symbol, ⟨s⟩, and an explicit end symbol, ⟨/s⟩, are part of the vocabulary, and a string w_0^n is a complete string if and only if w_0 is ⟨s⟩ and w_n is ⟨/s⟩. Since the beginning of string symbol is not predicted by language models, but rather is axiomatic in the same way that $S^\dagger$ is for a parser, we can safely omit it from the current discussion, and simply assume that it is there. See Figure 1(b) for the explicit representation.

While a complete string of words must contain the end symbol as its final word, a string prefix does not have this restriction. For example, "Spot chased the ball ⟨/s⟩" is a complete string, and the following is the set of prefix strings of this complete string: "Spot"; "Spot chased"; "Spot chased the"; "Spot chased the ball"; and "Spot chased the ball ⟨/s⟩". A PCFG also defines a probability distribution over string prefixes, and we will present this in terms of partial derivations. A partial derivation (or parse) d is defined with respect to a prefix string w_0^j as follows: it is the leftmost derivation of the string, with w_j on the right-hand side of the last expansion in the derivation.[4] Let $D_{w_0^j}$ be the set of all partial derivations for a prefix string w_0^j. Then

$$\mathrm{P}(w_0^j) = \sum_{d \in D_{w_0^j}} \mathrm{P}(d) \tag{2}$$

We left-factor the PCFG, so that all productions are binary, except those with a single terminal on the right-hand side and epsilon productions.[5] We do this because it delays predictions about what nonterminals we expect later in the string until we have seen more of the string. In effect, this is an underspecification of some of the predictions that our top-down parser is making about the rest of the string. The left-factorization transform that we use is identical to what is called right binarization in Roark and Johnson (1999). See that paper for more discussion of the benefits of

[3] A PCFG is consistent or tight if there is no probability mass reserved for infinite trees. Chi and Geman (1998) proved that any PCFG estimated from a treebank with the relative frequency estimator is tight. All of the PCFGs that are used in this paper are estimated using the relative frequency estimator.

[4] A leftmost derivation is a derivation in which the leftmost nonterminal is always expanded.

[5] The only ϵ-productions that we will use in this paper are those introduced by left-factorization.

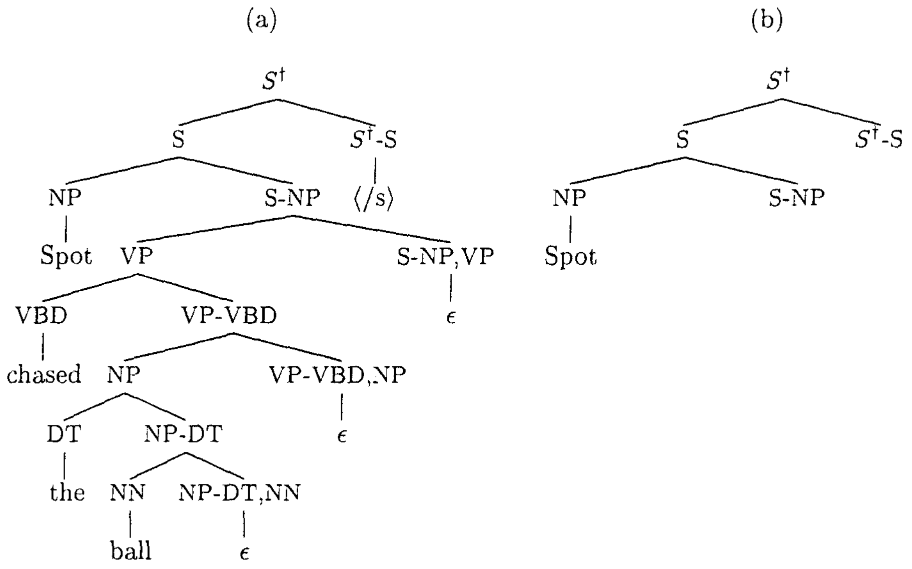

Figure 2
Two parse trees: (a) a complete left-factored parse tree with epsilon productions and an explicit stop symbol; and (b) a partial left-factored parse tree.

factorization for top-down and left-corner parsing. For a grammar G, we define a factored grammar G_f as follows:

i. $(A \rightarrow B \;\; A\text{-}B) \in G_f$ iff $(A \rightarrow B\beta) \in G$, s.t. $B \in V$ and $\beta \in V^*$
ii. $(A\text{-}\alpha \rightarrow B \;\; A\text{-}\alpha B) \in G_f$ iff $(A \rightarrow \alpha B\beta) \in G$, s.t. $B \in V$, $\alpha \in V^+$, and $\beta \in V^*$
iii. $(A\text{-}\alpha B \rightarrow \epsilon) \in G_f$ iff $(A \rightarrow \alpha B) \in G$, s.t. $B \in V$ and $\alpha \in V^*$
iv. $(A \rightarrow a) \in G_f$ iff $(A \rightarrow a) \in G$, s.t. $a \in T$

We can see the effect of this transform on our example parse trees in Figure 2. This underspecification of the nonterminal predictions (e.g., VP-VBD in the example in Figure 2, as opposed to NP), allows lexical items to become part of the left context, and so be used to condition production probabilities, even the production probabilities of constituents that dominate them in the unfactored tree. It also brings words further downstream into the look-ahead at the point of specification. Note that partial trees are defined in exactly the same way (Figure 2b), but that the nonterminal yields are made up exclusively of the composite nonterminals introduced by the grammar transform.

This transform has a couple of very nice properties. First, it is easily reversible, i.e., every parse tree built with G_f corresponds to a unique parse tree built with G. Second, if we use the relative frequency estimator for our production probabilities, the probability of a tree built with G_f is identical to the probability of the corresponding tree built with G.

Finally, let us introduce the term c-command. We will use this notion in our conditional probability model, and it is also useful for understanding some of the previous work in this area. The simple definition of c-command that we will be using in this paper is the following: a node A c-commands a node B if and only if (i) A does not dominate B; and (ii) the lowest branching node (i.e., non-unary node) that dominates

A also dominates B.[6] Thus in Figure 1(a), the subject NP and the VP each c-command the other, because neither dominates the other and the lowest branching node above both (the S) dominates the other. Notice that the subject NP c-commands the object NP, but not vice versa, since the lowest branching node that dominates the object NP is the VP, which does not dominate the subject NP.

2.2 Language Modeling for Speech Recognition

This section will briefly introduce language modeling for statistical speech recognition.[7]

In language modeling, we assign probabilities to strings of words. To assign a probability, the chain rule is generally invoked. The chain rule states, for a string of k+1 words:

$$\mathrm{P}(w_0^k) = \mathrm{P}(w_0) \prod_{i=1}^{k} \mathrm{P}(w_i \mid w_0^{i-1}) \tag{3}$$

A Markov language model of order n truncates the conditioning information in the chain rule to include only the previous n words.

$$\mathrm{P}(w_0^k) = \mathrm{P}(w_0)\mathrm{P}(w_1 \mid w_0) \ldots \mathrm{P}(w_{n-1} \mid w_0^{n-2}) \prod_{i=n}^{k} \mathrm{P}(w_i \mid w_{i-n}^{i-1}) \tag{4}$$

These models are commonly called n-gram models.[8] The standard language model used in many speech recognition systems is the trigram model, i.e., a Markov model of order 2, which can be characterized by the following equation:

$$\mathrm{P}(w_0^{n-1}) = \mathrm{P}(w_0)\mathrm{P}(w_1 \mid w_0) \prod_{i=2}^{n-1} \mathrm{P}(w_i \mid w_{i-2}^{i-1}) \tag{5}$$

To smooth the trigram models that are used in this paper, we interpolate the probability estimates of higher-order Markov models with lower-order Markov models (Jelinek and Mercer 1980). The idea behind interpolation is simple, and it has been shown to be very effective. For an interpolated $(n + 1)$-gram:

$$\mathrm{P}(w_i \mid w_{i-n}^{i-1}) = \lambda_n(w_{i-n}^{i-1})\widehat{\mathrm{P}}(w_i \mid w_{i-n}^{i-1}) + (1 - \lambda_n(w_{i-n}^{i-1}))\mathrm{P}(w_i \mid w_{i-n+1}^{i-1}) \tag{6}$$

Here $\widehat{\mathrm{P}}$ is the empirically observed relative frequency, and λ_n is a function from V^n to $[0, 1]$. This interpolation is recursively applied to the smaller-order n-grams until the bigram is finally interpolated with the unigram, i.e., $\lambda_0 = 1$.

3. Previous Work

There have been attempts to jump over adjacent words to words farther back in the left context, without the use of dependency links or syntactic structure, for example Saul and Pereira (1997) and Rosenfeld (1996, 1997). We will focus our very brief review, however, on those that use grammars or parsing for their language models. These can be divided into two rough groups: those that use the grammar as a language model,

[6] A node A dominates a node B in a tree if and only if either (i) A is the parent of B; or (ii) A is the parent of a node C that dominates B.

[7] For a detailed introduction to statistical speech recognition, see Jelinek (1997).

[8] The n in *n-gram* is one more than the order of the Markov model, since the n-gram includes the word being conditioned.

and those that use a parser to uncover phrasal heads standing in an important relation (c-command) to the current word. The approach that we will subsequently present uses the probabilistic grammar as its language model, but only includes probability mass from those parses that are found, that is, it uses the parser to find a subset of the total set of parses (hopefully most of the high-probability parses) and uses the sum of their probabilities as an estimate of the true probability given the grammar.

3.1 Grammar Models

As mentioned in Section 2.1, a PCFG defines a probability distribution over strings of words. One approach to syntactic language modeling is to use this distribution directly as a language model. There are efficient algorithms in the literature (Jelinek and Lafferty 1991; Stolcke 1995) for calculating exact string prefix probabilities given a PCFG. The algorithms both utilize a left-corner matrix, which can be calculated in closed form through matrix inversion. They are limited, therefore, to grammars where the nonterminal set is small enough to permit inversion. String prefix probabilities can be straightforwardly used to compute conditional word probabilities by definition:

$$\mathrm{P}(w_{j+1} \mid w_0^j) = \frac{\mathrm{P}(w_0^{j+1})}{\mathrm{P}(w_0^j)} \tag{7}$$

Stolcke and Segal (1994) and Jurafsky et al. (1995) used these basic ideas to estimate bigram probabilities from hand-written PCFGs, which were then used in language models. Interpolating the observed bigram probabilities with these calculated bigrams led, in both cases, to improvements in word error rate over using the observed bigrams alone, demonstrating that there is some benefit to using these syntactic language models to generalize beyond observed *n*-grams.

3.2 Finding Phrasal Heads

Another approach that uses syntactic structure for language modeling has been to use a shift-reduce parser to "surface" c-commanding phrasal headwords or part-of-speech (POS) tags from arbitrarily far back in the prefix string, for use in a trigram-like model.

A shift-reduce parser operates from left to right using a stack and a pointer to the next word in the input string.[9] Each stack entry consists minimally of a nonterminal label. The parser performs two basic operations: (i) **shifting**, which involves pushing the POS label of the next word onto the stack and moving the pointer to the following word in the input string; and (ii) **reducing**, which takes the top k stack entries and replaces them with a single new entry, the nonterminal label of which is the left-hand side of a rule in the grammar that has the k top stack entry labels on the right-hand side. For example, if there is a rule NP → DT NN, and the top two stack entries are NN and DT, then those two entries can be popped off of the stack and an entry with the label NP pushed onto the stack.

Goddeau (1992) used a robust deterministic shift-reduce parser to condition word probabilities by extracting a specified number of stack entries from the top of the current state, and conditioning on those entries in a way similar to an *n*-gram. In empirical trials, Goddeau used the top two stack entries to condition the word probability. He was able to reduce both sentence and word error rates on the ATIS corpus using this method.

[9] For details, see Hopcroft and Ullman (1979), for example.

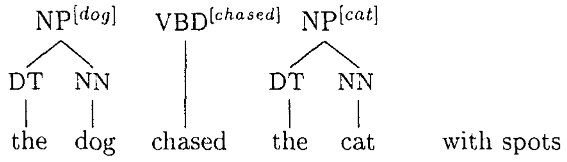

Figure 3
Tree representation of a derivation state.

The structured language model (SLM) used in Chelba and Jelinek (1998a, 1998b, 1999), Jelinek and Chelba (1999), and Chelba (2000) is similar to that of Goddeau, except that (i) their shift-reduce parser follows a nondeterministic beam search, and (ii) each stack entry contains, in addition to the nonterminal node label, the headword of the constituent. The SLM is like a trigram, except that the conditioning words are taken from the tops of the stacks of candidate parses in the beam, rather than from the linear order of the string.

Their parser functions in three stages. The first stage assigns a probability to the word given the left context (represented by the stack state). The second stage predicts the POS given the word and the left context. The last stage performs all possible parser operations (reducing stack entries and shifting the new word). When there is no more parser work to be done (or, in their case, when the beam is full), the following word is predicted. And so on until the end of the string.

Each different POS assignment or parser operation is a step in a derivation. Each distinct derivation path within the beam has a probability and a stack state associated with it. Every stack entry has a nonterminal node label and a designated headword of the constituent. When all of the parser operations have finished at a particular point in the string, the next word is predicted as follows: For each derivation in the beam, the headwords of the two topmost stack entries form a trigram with the conditioned word. This interpolated trigram probability is then multiplied by the normalized probability of the derivation, to provide that derivation's contribution to the probability of the word. More precisely, for a beam of derivations D_i

$$\mathrm{P}(w_{i+1} \mid w_0^i) = \frac{\sum_{d \in D_i} \mathrm{P}(w_{i+1} \mid h_{0d}, h_{1d}) \mathrm{P}(d)}{\sum_{d \in D_i} \mathrm{P}(d)} \tag{8}$$

where h_{0d} and h_{1d} are the lexical heads of the top two entries on the stack of d.

Figure 3 gives a partial tree representation of a potential derivation state for the string "`the dog chased the cat with spots`", at the point when the word "`with`" is to be predicted. The shift-reduce parser will have, perhaps, built the structure shown, and the stack state will have an NP entry with the head "`cat`" at the top of the stack, and a VBD entry with the head "`chased`" second on the stack. In the Chelba and Jelinek model, the probability of "`with`" is conditioned on these two headwords, for this derivation.

Since the specific results of the SLM will be compared in detail with our model when the empirical results are presented, at this point we will simply state that they have achieved a reduction in both perplexity and word error rate over a standard trigram using this model.

The rest of this paper will present our parsing model, its application to language modeling for speech recognition, and empirical results.

4. Top-Down Parsing and Language Modeling

Statistically based heuristic best-first or beam-search strategies (Caraballo and Charniak 1998; Charniak, Goldwater, and Johnson 1998; Goodman 1997) have yielded an enormous improvement in the quality and speed of parsers, even without any guarantee that the parse returned is, in fact, that with the maximum likelihood for the probability model. The parsers with the highest published broad-coverage parsing accuracy, which include Charniak (1997, 2000), Collins (1997, 1999), and Ratnaparkhi (1997), all utilize simple and straightforward statistically based search heuristics, pruning the search-space quite dramatically.[10] Such methods are nearly always used in conjunction with some form of dynamic programming (henceforth DP). That is, search efficiency for these parsers is improved by both statistical search heuristics and DP. Here we will present a parser that uses simple search heuristics of this sort without DP. Our approach is found to yield very accurate parses efficiently, and, in addition, to lend itself straightforwardly to estimating word probabilities on-line, that is, in a single pass from left to right. This on-line characteristic allows our language model to be interpolated on a word-by-word basis with other models, such as the trigram, yielding further improvements.

Next we will outline our conditional probability model over rules in the PCFG, followed by a presentation of the top-down parsing algorithm. We will then present empirical results in two domains: one to compare with previous work in the parsing literature, and the other to compare with previous work using parsing for language modeling for speech recognition, in particular with the Chelba and Jelinek results mentioned above.

4.1 Conditional Probability Model

A simple PCFG conditions rule probabilities on the left-hand side of the rule. It has been shown repeatedly—e.g., Briscoe and Carroll (1993), Charniak (1997), Collins (1997), Inui et al. (1997), Johnson (1998)—that conditioning the probabilities of structures on the context within which they appear, for example on the lexical head of a constituent (Charniak 1997; Collins 1997), on the label of its parent nonterminal (Johnson 1998), or, ideally, on both and many other things besides, leads to a much better parsing model and results in higher parsing accuracies.

One way of thinking about conditioning the probabilities of productions on contextual information (e.g., the label of the parent of a constituent or the lexical heads of constituents), is as annotating the extra conditioning information onto the labels in the context-free rules. Examples of this are bilexical grammars—such as Eisner and Satta (1999), Charniak (1997), Collins (1997)—where the lexical heads of each constituent are annotated on both the right- and left-hand sides of the context-free rules, under the constraint that every constituent inherits the lexical head from exactly one of its children, and the lexical head of a POS is its terminal item. Thus the rule S → NP VP becomes, for instance, S[*barks*] → NP[*dog*] VP[*barks*]. One way to estimate the probabilities of these rules is to annotate the heads onto the constituent labels in the training corpus and simply count the number of times particular productions occur (relative frequency estimation). This procedure yields conditional probability distributions of

[10] Johnson et al. (1999), Henderson and Brill (1999), and Collins (2000) demonstrate methods for choosing the best complete parse tree from among a set of complete parse trees, and the latter two show accuracy improvements over some of the parsers cited above, from which they generated their candidate sets. Here we will be comparing our work with parsing algorithms, i.e., algorithms that build parses for strings of words.

constituents on the right-hand side with their lexical heads, given the left-hand side constituent and its lexical head. The same procedure works if we annotate parent information onto constituents. This is how Johnson (1998) conditioned the probabilities of productions: the left-hand side is no longer, for example, S, but rather S$^\uparrow$SBAR, i.e., an S with SBAR as parent. Notice, however, that in this case the annotations on the right-hand side are predictable from the annotation on the left-hand side (unlike, for example, bilexical grammars), so that the relative frequency estimator yields conditional probability distributions of the original rules, given the parent of the left-hand side.

All of the conditioning information that we will be considering will be of this latter sort: the only novel predictions being made by rule expansions are the node labels of the constituents on the right-hand side. Everything else is already specified by the left context. We use the relative frequency estimator, and smooth our production probabilities by interpolating the relative frequency estimates with those obtained by "annotating" less contextual information.

This perspective on conditioning production probabilities makes it easy to see that, in essence, by conditioning these probabilities, we are growing the state space. That is, the number of distinct nonterminals grows to include the composite labels; so does the number of distinct productions in the grammar. In a top-down parser, each rule expansion is made for a particular candidate parse, which carries with it the entire rooted derivation to that point; in a sense, the left-hand side of the rule is annotated with the entire left context, and the rule probabilities can be conditioned on any aspect of this 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 $A \rightarrow \alpha$. 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 nonterminal 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.

Recall that we are working with a factored grammar, so some of the nodes in the factored tree have nonterminal labels that were created by the factorization, and may not be precisely what we want for conditioning purposes. In order to avoid any confusions in identifying the nonterminal label of a particular rule production in either its factored or nonfactored version, we introduce the function `constituent(`A`)` for every nonterminal in the factored grammar G_f, which is simply the label of the constituent whose factorization results in A. For example, in Figure 2, `constituent(`NP-DT-NN`)` is simply NP.

Note that a function can return different values depending upon the location in the tree of the nonterminal that is being expanded. For example, suppose that we have a function that returns the label of the closest sibling to the left of `constituent(`A`)` or NULL if no such node exists. Then a subsequent function could be defined as follows: return the parent of the parent (the grandparent) of `constituent(`A`)` *only if* `constituent(`A`)` has no sibling to the left—in other words, if the previous function returns NULL; *otherwise* return the second closest sibling to the left of `constituent(`A`)`, or, as always, NULL if no such node exists. If the function returns, for example, NP, this could either mean that the grandparent is NP or the second closest sibling is

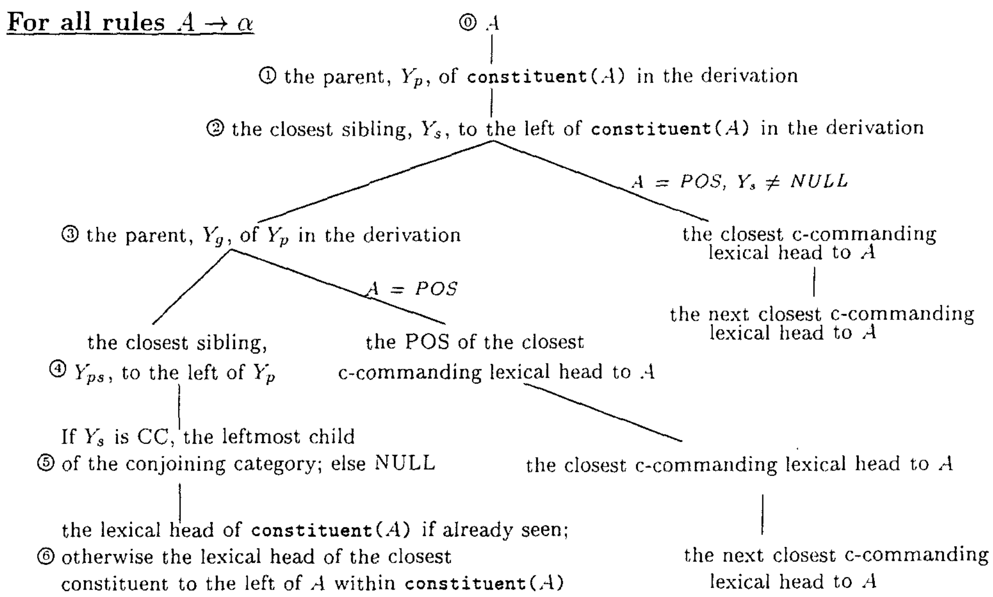
For all rules $A \rightarrow \alpha$
⓪ A
① the parent, Y_p, of constituent(A) in the derivation
② the closest sibling, Y_s, to the left of constituent(A) in the derivation
A = POS, $Y_s \neq$ NULL
③ the parent, Y_g, of Y_p in the derivation
the closest c-commanding lexical head to A
A = POS
the next closest c-commanding lexical head to A
④ the closest sibling, Y_{ps}, to the left of Y_p
the POS of the closest c-commanding lexical head to A
⑤ If Y_s is CC, the leftmost child of the conjoining category; else NULL
the closest c-commanding lexical head to A
⑥ the lexical head of constituent(A) if already seen; otherwise the lexical head of the closest constituent to the left of A within constituent(A)
the next closest c-commanding lexical head to A

Figure 4
Conditional probability model represented as a decision tree, identifying the location in the partial parse tree of the conditioning information.

NP; yet there is no ambiguity in the meaning of the function, since the result of the previous function disambiguates between the two possibilities.

The functions that were used for the present study to condition the probability of the rule, $A \rightarrow \alpha$, are presented in Figure 4, in a tree structure. This is a sort of decision tree for a tree-walking algorithm to decide what value to return, for a given partial tree and a given depth. For example, if the algorithm is asked for the value at level 0, it will return A, the left-hand side of the rule being expanded.[11] Suppose the algorithm is asked for the value at level 4. After level 2 there is a branch in the decision tree. If the left-hand side of the rule is a POS, and there is no sibling to the left of constituent(A) in the derivation, then the algorithm takes the right branch of the decision tree to decide what value to return; otherwise the left branch. Suppose it takes the left branch. Then after level 3, there is another branch in the decision tree. If the left-hand side of the production is a POS, then the algorithm takes the right branch of the decision tree, and returns (at level 4) the POS of the closest c-commanding lexical head to A, which it finds by walking the parse tree; if the left-hand side of the rule is not a POS, then the algorithm returns (at level 4) the closest sibling to the left of the parent of constituent(A).

The functions that we have chosen for this paper follow from the intuition (and experience) that what helps parsing is different depending on the constituent that is being expanded. POS nodes have lexical items on the right-hand side, and hence can bring into the model some of the head-head dependencies that have been shown to be so effective. If the POS is leftmost within its constituent, then very often the lexical

[11] Recall that A can be a composite nonterminal introduced by grammar factorization. When the function is defined in terms of constituent(A), the values returned are obtained by moving through the nonfactored tree.

Table 1
Levels of conditioning information, mnemonic labels, and a brief description of the information level for empirical results.

Conditioning	Mnemonic Label	Information Level
0,0,0	none	Simple PCFG
2,2,2	par+sib	Small amount of structural context
5,2,2	NT struct	All structural (non-lexical) context for non-POS
6,2,2	NT head	Everything for non-POS expansions
6,3,2	POS struct	More structural info for leftmost POS expansions
6,5,2	attach	All attachment info for leftmost POS expansions
6,6,4	all	Everything

item is sensitive to the governing category to which it is attaching. For example, if the POS is a preposition, then its probability of expanding to a particular word is very different if it is attaching to a noun phrase than if it is attaching to a verb phrase, and perhaps quite different depending on the head of the constituent to which it is attaching. Subsequent POSs within a constituent are likely to be open-class words, and less dependent on these sorts of attachment preferences.

Conditioning on parents and siblings of the left-hand side has proven to be very useful. To understand why this is the case, one need merely to think of VP expansions. If the parent of a VP is another VP (i.e., if an auxiliary or modal verb is used), then the distribution over productions is different than if the parent is an S. Conditioning on head information, both POS of the head and the lexical item itself, has proven useful as well, although given our parser's left-to-right orientation, in many cases the head has not been encountered within the particular constituent. In such a case, the head of the last child within the constituent is used as a proxy for the constituent head. All of our conditioning functions, with one exception, return either parent or sibling node labels at some specific distance from the left-hand side, or head information from c-commanding constituents. The exception is the function at level 5 along the left branch of the tree in Figure 4. Suppose that the node being expanded is being conjoined with another node, which we can tell by the presence or absence of a CC node. In that case, we want to condition the expansion on how the conjoining constituent expanded. In other words, this attempts to capture a certain amount of parallelism between the expansions of conjoined categories.

In presenting the parsing results, we will systematically vary the amount of conditioning information, so as to get an idea of the behavior of the parser. We will refer to the amount of conditioning by specifying the deepest level from which a value is returned for each branching path in the decision tree, from left to right in Figure 4: the first number is for left contexts where the left branch of the decision tree is always followed (non-POS nonterminals on the left-hand side); the second number is for a left branch followed by a right branch (POS nodes that are leftmost within their constituent); and the third number is for the contexts where the right branch is always followed (POS nodes that are not leftmost within their constituent). For example, (4,3,2) would represent a conditional probability model that (i) returns NULL for all functions below level 4 in all contexts; (ii) returns NULL for all functions below level 3 if the left-hand side is a POS; and (iii) returns NULL for all functions below level 2 for nonleftmost POS expansions.

Table 1 gives a breakdown of the different levels of conditioning information used in the empirical trials, with a mnemonic label that will be used when presenting results. These different levels were chosen as somewhat natural points at which to observe how much of an effect increasing the conditioning information has. We first include structural information from the context, namely, node labels from constituents in the left context. Then we add lexical information, first for non-POS expansions, then for leftmost POS expansions, then for all expansions.

All of the conditional probabilities are linearly interpolated. For example, the probability of a rule conditioned on six events is the linear interpolation of two probabilities: (i) the empirically observed relative frequency of the rule when the six events co-occur; and (ii) the probability of the rule conditioned on the first five events (which is in turn interpolated). The interpolation coefficients are a function of the frequency of the set of conditioning events, and are estimated by iteratively adjusting the coefficients so as to maximize the likelihood of a held-out corpus.

This was an outline of the conditional probability model that we used for the PCFG. The model allows us to assign probabilities to derivations, which can be used by the parsing algorithm to decide heuristically which candidates are promising and should be expanded, and which are less promising and should be pruned. We now outline the top-down parsing algorithm.

4.2 Top-Down Probabilistic Parsing

This parser is essentially a stochastic version of the top-down parser described in Aho, Sethi, and Ullman (1986). It uses a PCFG with a conditional probability model of the sort defined in the previous section. 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 parser takes an input string w_0^n, a PCFG G, and a priority queue of candidate analyses. A candidate analysis $C = (D, \mathcal{S}, P_D, F, w_i^n)$ consists of a derivation D, a stack $\mathcal{S}$, 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 $\mathcal{S}$ contains a sequence of nonterminal symbols, 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($\mathcal{S}$,w_i), which is a measure of the likelihood of the stack $\mathcal{S}$ rewriting with w_i at its left corner.

We can define a derives relation, denoted $\Rightarrow$, between two candidate analyses as follows. $(D, \mathcal{S}, P_D, F, w_i^n) \Rightarrow (D', \mathcal{S}', P_{D'}, F', w_j^n)$ if and only if[12]

i. $D' = D + A \rightarrow X_0 \ldots X_k$
ii. $\mathcal{S} = A\alpha\$$;
iii. either $\mathcal{S}' = X_0 \ldots X_k\alpha\$$ and $j = i$
or $k = 0$, $X_0 = w_i$, $j = i + 1$, and $\mathcal{S}' = \alpha\$$;
iv. $P_{D'} = P_D\mathrm{P}(A \rightarrow X_0 \ldots X_k)$; and
v. $F' = P_{D'}\mathrm{LAP}(\mathcal{S}', w_j)$

The parse begins with a single candidate analysis on the priority queue: $(\langle\rangle$, $S^\dagger\$$, 1, 1, $w_0^n)$. Next, the top ranked candidate analysis, $C = (D, \mathcal{S}, P_D, F, w_i^n)$, is popped from the priority queue. If $\mathcal{S} = \$$ 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.

[12] Again, for ease of exposition, we will ignore ϵ-productions. Everything presented here can be straightforwardly extended to include them. 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 $\mathcal{S}$ is used to denote stacks, while $S^\dagger$ is the start symbol.

We implement this as a beam search. For each word position i, we have a separate priority queue 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 H_{i+1}, all candidate analyses remaining on H_i are discarded. Since $w_n = \langle/\mathrm{s}\rangle$, all parses that are pushed onto H_{n+1} are complete. The parse on 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.

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 topmost nonterminal on the stack of the analysis or it might be the left corner of the nth nonterminal, after the top $n-1$ nonterminals have rewritten to ϵ. Of course, we cannot expect to have adequate statistics for each nonterminal/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.[13]

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

$$\mathrm{LAP}(S, w_i) = \sum_{\alpha \in (V \cup T)^*} \mathrm{P_G}(A_0^n \xrightarrow{\star} w_i\alpha) \tag{9}$$

We recursively estimate this with two empirically observed conditional probabilities for every nonterminal A_i: $\widehat{\mathrm{P}}(A_i \xrightarrow{\star} w_i\alpha)$ and $\widehat{\mathrm{P}}(A_i \xrightarrow{\star} \epsilon)$. The same empirical probability, $\widehat{\mathrm{P}}(A_i \xrightarrow{\star} X\alpha)$, is collected for every preterminal X as well. The LAP approximation for a given stack state and look-ahead terminal is:

$$\mathrm{P_G}(A_j^n \xrightarrow{\star} w_i\alpha) \approx \mathrm{P_G}(A_j \xrightarrow{\star} w_i\alpha) + \widehat{\mathrm{P}}(A_j \xrightarrow{\star} \epsilon)\mathrm{P_G}(A_{j+1}^n \xrightarrow{\star} w_i\alpha) \tag{10}$$

where

$$\mathrm{P_G}(A_j \xrightarrow{\star} w_i\alpha) \approx \lambda_{A_j}\widehat{\mathrm{P}}(A_j \xrightarrow{\star} w_i\alpha) + (1 - \lambda_{A_j}) \sum_{X \in V} \widehat{\mathrm{P}}(A_j \xrightarrow{\star} X\alpha)\widehat{\mathrm{P}}(X \rightarrow w_i) \tag{11}$$

The lambdas are a function of the frequency of the nonterminal A_j, in the standard way (Jelinek and Mercer 1980).

The beam threshold at word w_i is a function of the probability of the top-ranked candidate analysis on priority queue H_{i+1} and the number of candidates on H_{i+1}. The basic idea is that we want the beam to be very wide if there are few analyses that have been advanced, but relatively narrow if many analyses have been advanced. If $\tilde{p}$ is the probability of the highest-ranked analysis on H_{i+1}, then another analysis is discarded if its probability falls below $\tilde{p}f(\gamma, |H_{i+1}|)$, where γ is an initial parameter, which we call the **base beam factor**. For the current study, γ was 10^{-11}, unless otherwise noted, and $f(\gamma, |H_{i+1}|) = \gamma|H_{i+1}|^3$. Thus, if 100 analyses have already been pushed onto H_{i+1}, then a candidate analysis must have a probability above $10^{-5}\tilde{p}$ to avoid being pruned. After 1,000 candidates, the beam has narrowed to $10^{-2}\tilde{p}$. There is also a maximum number of allowed analyses on H_i, in case the parse fails to advance an analysis to H_{i+1}. This was typically 10,000.

As mentioned in Section 2.1, we left-factor the grammar, so that all productions are binary, except those with a single terminal on the right-hand side and epsilon productions. The only ϵ-productions are those introduced by left-factorization. Our factored

[13] Equivalently, we can split the analyses at this point, so that there is one POS per analysis.

grammar was produced by factoring the trees in the training corpus before grammar induction, which proceeded in the standard way, by counting rule frequencies.

5. Empirical Results

The empirical results will be presented in three stages: (i) trials to examine the accuracy and efficiency of the parser; (ii) trials to examine its effect on test corpus perplexity and recognition performance; and (iii) trials to examine the effect of beam variation on these performance measures. Before presenting the results, we will introduce the methods of evaluation.

5.1 Evaluation

Perplexity is a standard measure within the speech recognition community for comparing language models. In principle, if two models are tested on the same test corpus, the model that assigns the lower perplexity to the test corpus is the model closest to the true distribution of the language, and thus better as a prior model for speech recognition. Perplexity is the exponential of the cross entropy, which we will define next.

Given a random variable X with distribution p and a probability model q, the cross entropy, $H(p,q)$ is defined as follows:

$$H(p,q) = -\sum_{x \in X} p(x) \log q(x) \tag{12}$$

Let p be the true distribution of the language. Then, under certain assumptions, given a large enough sample, the sample mean of the negative log probability of a model will converge to its cross entropy with the true model.[14] That is

$$H(p,q) = -\lim_{n \rightarrow \infty} \frac{1}{n} \log q(w_0^n) \tag{13}$$

where w_0^n is a string of the language L. In practice, one takes a large sample of the language, and calculates the negative log probability of the sample, normalized by its size.[15] The lower the cross entropy (i.e., the higher the probability the model assigns to the sample), the better the model. Usually this is reported in terms of perplexity, which we will do as well.[16]

Some of the trials discussed below will report results in terms of word and/or sentence error rate, which are obtained when the language model is embedded in a speech recognition system. Word error rate is the number of deletion, insertion, or substitution errors per 100 words. Sentence error rate is the number of sentences with one or more errors per 100 sentences.

Statistical parsers are typically evaluated for accuracy at the constituent level, rather than simply whether or not the parse that the parser found is completely correct or not. A constituent for evaluation purposes consists of a label (e.g., NP) and a span (beginning and ending word positions). For example, in Figure 1(a), there is a VP that spans the words "`chased the ball`". Evaluation is carried out on a hand-parsed test corpus, and the manual parses are treated as correct. We will call the manual parse

[14] See Cover and Thomas (1991) for a discussion of the Shannon-McMillan-Breiman theorem, under the assumptions of which this convergence holds.

[15] It is important to remember to include the end marker in the strings of the sample.

[16] When assessing the magnitude of a perplexity improvement, it is often better to look at the reduction in cross entropy, by taking the log of the perplexity. It will be left to the reader to do so.

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 constituents, which are called labeled recall (LR) and labeled precision (LP). Sometimes in figures we will plot their average, and also what can be termed the parse error, which is one minus their average.

LR and LP are part of the standard set of PARSEVAL measures of parser quality (Black et al. 1991). From this set of measures, we will also include 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, we show the average number of rule expansions considered per word, that is, 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 nonempty 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, 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.

5.2 Parser Accuracy and Efficiency

The first set of results looks at the performance of the parser on the standard corpora for statistical parsing trials: Sections 2–21 (989,860 words, 39,832 sentences) of the Penn Treebank (Marcus, Santorini, and Marcinkiewicz 1993) served as the training data, Section 24 (34,199 words, 1,346 sentences) as the held-out data for parameter estimation, and Section 23 (59,100 words, 2,416 sentences) as the test data. Section 22 (41,817 words, 1,700 sentences) served as the development corpus, on which the parser was tested until stable versions were ready to run on the test data, to avoid developing the parser to fit the specific test data.

Table 2 shows trials with increasing amounts of conditioning information from the left context. There are a couple of things to notice from these results. First, and least surprising, is that the accuracy of the parses improved as we conditioned on more and more information. Like the nonlexicalized parser in Roark and Johnson (1999), we found that the search efficiency, in terms of number of rule expansions considered or number of analyses advanced, also improved as we increased the amount of conditioning. Unlike the Roark and Johnson parser, however, our coverage did not substantially drop as the amount of conditioning information increased, and in some cases, coverage improved slightly. They did not smooth their conditional probability estimates, and blamed sparse data for their decrease in coverage as they increased the conditioning information. These results appear to support this, since our smoothed model showed no such tendency.

Figure 5 shows the reduction in parser error, $1 - \frac{\text{LR+LP}}{2}$, and the reduction in rule expansions considered as the conditioning information increased. The bulk of

Table 2
Results conditioning on various contextual events, standard training and testing corpora.

Conditioning	LR	LP	CB	0 CB	≤ 2 CB	Percent Failed	Average Rule Expansions Considered[†]	Average Analyses Advanced[†]
				Section 23: 2245 sentences of length ≤ 40				
none	71.1	75.3	2.48	37.3	62.9	0.9	14,369	516.5
par+sib	82.8	83.6	1.55	54.3	76.2	1.1	9,615	324.4
NT struct	84.3	84.9	1.38	56.7	79.5	1.0	8,617	284.9
NT head	85.6	85.7	1.27	59.2	81.3	0.9	7,600	251.6
POS struct	86.1	86.2	1.23	60.9	82.0	1.0	7,327	237.9
attach	86.7	86.6	1.17	61.7	83.2	1.2	6,834	216.8
all	86.6	86.5	1.19	62.0	82.7	1.3	6,379	198.4
				Section 23: 2416 sentences of length ≤ 100				
attach	85.8	85.8	1.40	58.9	80.3	1.5	7,210	227.9
all	85.7	85.7	1.41	59.0	79.9	1.7	6,709	207.6

[†] per word

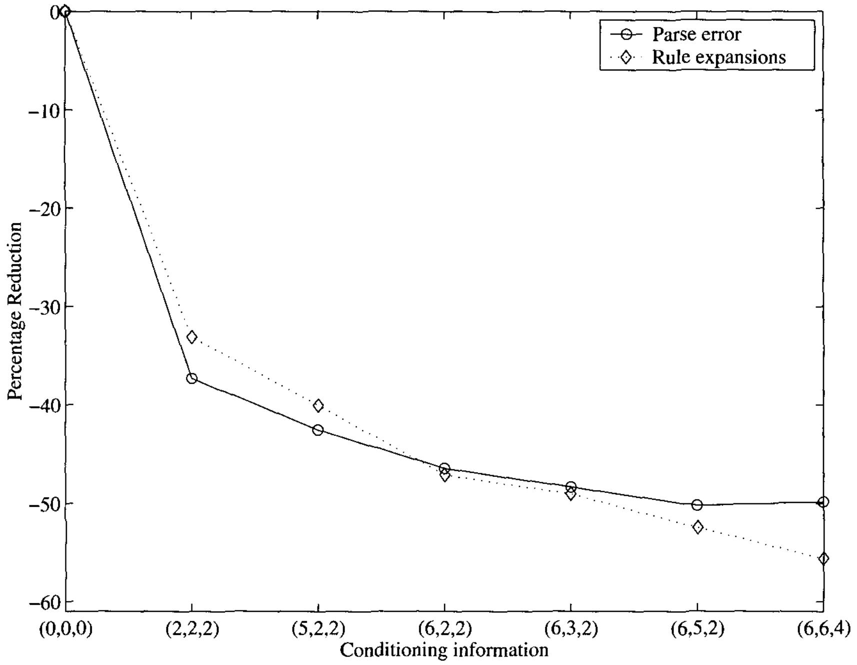

Figure 5
Reduction in average precision/recall error and in number of rule expansions per word as conditioning increases, for sentences of length ≤ 40.

the improvement comes from simply conditioning on the labels of the parent and the closest sibling to the node being expanded. Interestingly, conditioning all POS expansions on two c-commanding heads made no difference in accuracy compared to conditioning only leftmost POS expansions on a single c-commanding head; but it did improve the efficiency.

These results, achieved using very straightforward conditioning events and considering only the left context, are within one to four points of the best published

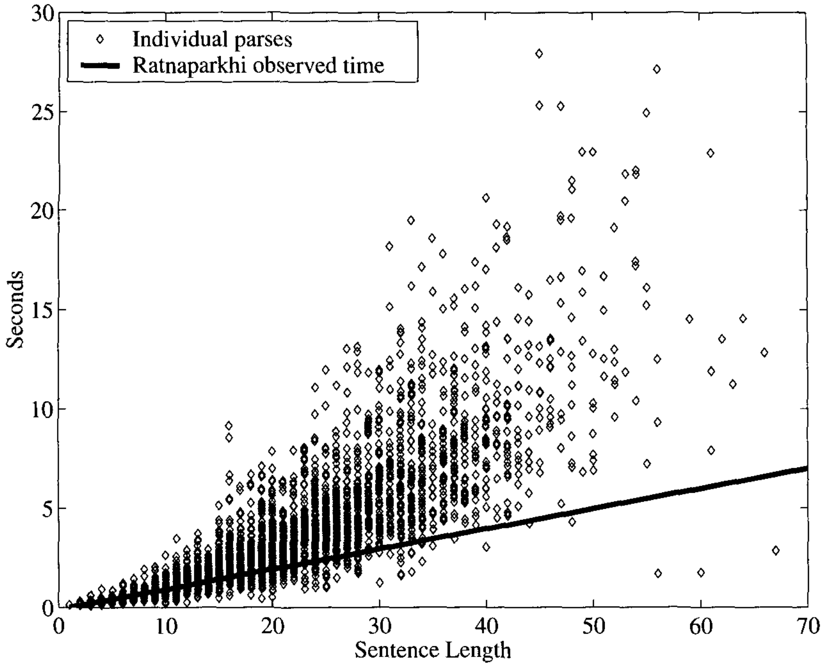

Figure 6
Observed running time on Section 23 of the Penn Treebank, with the full conditional probability model and beam of 10^{-11}, using one 300 MHz UltraSPARC processor and 256MB of RAM of a Sun Enterprise 450.

accuracies cited above.[17] Of the 2,416 sentences in the section, 728 had the totally correct parse, 30.1 percent tree accuracy. Also, the parser returns a set of candidate parses, from which we have been choosing the top ranked; if we use an oracle to choose the parse with the highest accuracy from among the candidates (which averaged 70.0 in number per sentence), we find an average labeled precision/recall of 94.1, for sentences of length $\leq$ 100. The parser, thus, could be used as a front end to some other model, with the hopes of selecting a more accurate parse from among the final candidates.

While we have shown that the conditioning information improves the efficiency in terms of rule expansions considered and analyses advanced, what does the efficiency of such a parser look like in practice? Figure 6 shows the observed time at our standard base beam of 10^{-11} with the full conditioning regimen, alongside an approximation of the reported observed (linear) time in Ratnaparkhi (1997). Our observed times look polynomial, which is to be expected given our pruning strategy: the denser the competitors within a narrow probability range of the best analysis, the more time will be spent working on these competitors; and the farther along in the sentence, the more chance for ambiguities that can lead to such a situation. While our observed times are not linear, and are clearly slower than his times (even with a faster machine), they are quite respectably fast. The differences between a *k*-best and a beam-search parser (not to mention the use of dynamic programming) make a running time difference unsur-

[17] Our score of 85.8 average labeled precision and recall for sentences less than or equal to 100 on Section 23 compares to: 86.7 in Charniak (1997), 86.9 in Ratnaparkhi (1997), 88.2 in Collins (1999), 89.6 in Charniak (2000), and 89.75 in Collins (2000).

prising. What is perhaps surprising is that the difference is not greater. Furthermore, this is quite a large beam (see discussion below), so that very large improvements in efficiency can be had at the expense of the number of analyses that are retained.

5.3 Perplexity Results

The next set of results will highlight what recommends this approach most: the ease with which one can estimate string probabilities in a single pass from left to right across the string. By definition, a PCFG's estimate of a string's probability is the sum of the probabilities of all trees that produce the string as terminal leaves (see Equation 1). In the beam search approach outlined above, we can estimate the string's probability in the same manner, by summing the probabilities of the parses that the algorithm finds. Since this is not an exhaustive search, the parses that are returned will be a subset of the total set of trees that would be used in the exact PCFG estimate; hence the estimate thus arrived at will be bounded above by the probability that would be generated from an exhaustive search. The hope is that a large amount of the probability mass will be accounted for by the parses in the beam. The method cannot overestimate the probability of the string.

Recall the discussion of the grammar models above, and our definition of the set of partial derivations $D_{w_0^j}$ with respect to a prefix string w_0^j (see Equations 2 and 7). By definition,

$$P(w_{j+1} \mid w_0^j) = \frac{P(w_0^{j+1})}{P(w_0^j)} = \frac{\sum_{d \in D_{w_0^{j+1}}} P(d)}{\sum_{d \in D_{w_0^j}} P(d)} \tag{14}$$

Note that the numerator at word w_j is the denominator at word w_{j+1}, so that the product of all of the word probabilities is the numerator at the final word, namely, the string prefix probability.

We can make a consistent estimate of the string probability by similarly summing over all of the trees within our beam. Let H_i^{init} be the priority queue H_i before any processing has begun with word w_i in the look-ahead. This is a subset of the possible leftmost partial derivations with respect to the prefix string w_0^{i-1}. Since H_{i+1}^{init} is produced by expanding only analyses on priority queue H_i^{init}, the set of complete trees consistent with the partial derivations on priority queue H_{i+1}^{init} is a subset of the set of complete trees consistent with the partial derivations on priority queue H_i^{init}, that is, the total probability mass represented by the priority queues is monotonically decreasing. Thus conditional word probabilities defined in a way consistent with Equation 14 will always be between zero and one. Our conditional word probabilities are calculated as follows:

$$P(w_i \mid w_0^{i-1}) = \frac{\sum_{d \in H_{i+1}^{init}} P(d)}{\sum_{d \in H_i^{init}} P(d)} \tag{15}$$

As mentioned above, the model cannot overestimate the probability of a string, because the string probability is simply the sum over the beam, which is a subset of the possible derivations. By utilizing a figure of merit to identify promising analyses, we are simply focusing our attention on those parses that are likely to have a high probability, and thus we are increasing the amount of probability mass that we do capture, of the total possible. It is not part of the probability model itself.

Since each word is (almost certainly, because of our pruning strategy) losing some probability mass, the probability model is not "proper"—the sum of the probabilities over the vocabulary is less than one. In order to have a proper probability distribution, we would need to renormalize by dividing by some factor. Note, however, that this renormalization factor is necessarily less than one, and thus would uniformly increase each word's probability under the model, that is, any perplexity results reported below will be higher than the "true" perplexity that would be assigned with a properly normalized distribution. In other words, renormalizing would make our perplexity measure lower still. The hope, however, is that the improved parsing model provided by our conditional probability model will cause the distribution over structures to be more peaked, thus enabling us to capture more of the total probability mass, and making this a fairly snug upper bound on the perplexity.

One final note on assigning probabilities to strings: because this parser does garden path on a small percentage of sentences, this must be interpolated with another estimate, to ensure that every word receives a probability estimate. In our trials, we used the unigram, with a very small mixing coefficient:

$$\mathrm{P}(w_i \mid w_0^{i-1}) = \lambda(w_0^{i-1}) \frac{\sum_{d \in H_{i+1}^{init}} \mathrm{P}(d)}{\sum_{d \in H_i^{init}} \mathrm{P}(d)} + (1 - \lambda(w_0^{i-1}))\widehat{\mathrm{P}}(w_i) \tag{16}$$

If $\sum_{d \in H_i^{init}} \mathrm{P}(d) = 0$ in our model, then our model provides no distribution over following words since the denominator is zero. Thus,

$$\lambda(w_0^{i-1}) = \begin{cases} 0 & \textit{if } \sum_{d \in H_i^{init}} \mathrm{P}(d) = 0 \\ .999 & \textit{otherwise} \end{cases} \tag{17}$$

Chelba and Jelinek (1998a, 1998b) also used a parser to help assign word probabilities, via the structured language model outlined in Section 3.2. They trained and tested the SLM on a modified, more "speech-like" version of the Penn Treebank. Their modifications included: (i) removing orthographic cues to structure (e.g., punctuation); (ii) replacing all numbers with the single token *N*; and (iii) closing the vocabulary at 10,000, replacing all other words with the UNK token. They used Sections 00–20 (929,564 words) as the development set, Sections 21–22 (73,760 words) as the check set (for interpolation coefficient estimation), and tested on Sections 23–24 (82,430 words). We obtained the training and testing corpora from them (which we will denote **C&J corpus**), and also created intermediate corpora, upon which only the first two modifications were carried out (which we will denote **no punct**). Differences in performance will give an indication of the impact on parser performance of the different modifications to the corpora. All trials in this section used Sections 00–20 for counts, held out 21–22, and tested on 23–24.

Table 3 shows several things. First, it shows relative performance for unmodified, no punct, and C&J corpora with the full set of conditioning information. We can see that removing the punctuation causes (unsurprisingly) a dramatic drop in the accuracy and efficiency of the parser. Interestingly, it also causes coverage to become nearly total, with failure on just two sentences per thousand on average.

We see the familiar pattern, in the C&J corpus results, of improving performance as the amount of conditioning information grows. In this case we have perplexity results as well, and Figure 7 shows the reduction in parser error, rule expansions, and perplexity as the amount of conditioning information grows. While all three seem to be similarly improved by the addition of structural context (e.g., parents and siblings), the addition of c-commanding heads has only a moderate effect on the parser accuracy, but a very large effect on the perplexity. The fact that the efficiency was improved more than the accuracy in this case (as was also seen in Figure 5), seems to indicate that this additional information is causing the distribution to become more peaked, so that fewer analyses are making it into the beam.

Table 3
Results conditioning on various contextual events, Sections 23–24, modifications following Chelba and Jelinek.

Corpora	Conditioning	LR	LP	Percent Failed	Perplexity	Average Rule Expansions Considered†	Average Analyses Advanced†
		Sections 23–24: 3761 sentences ≤ 120					
unmodified	all	85.2	85.1	1.7		7,206	213.5
no punct	all	82.4	82.9	0.2		9,717	251.8
C&J corpus	par+sib	75.2	77.4	0.1	310.04	17,418	457.2
C&J corpus	NT struct	77.3	79.2	0.1	290.29	15,948	408.8
C&J corpus	NT head	79.2	80.4	0.1	255.85	14,239	363.2
C&J corpus	POS struct	80.5	81.6	0.1	240.37	13,591	341.3
C&J corpus	all	81.7	82.1	0.2	152.26	11,667	279.7

†per word

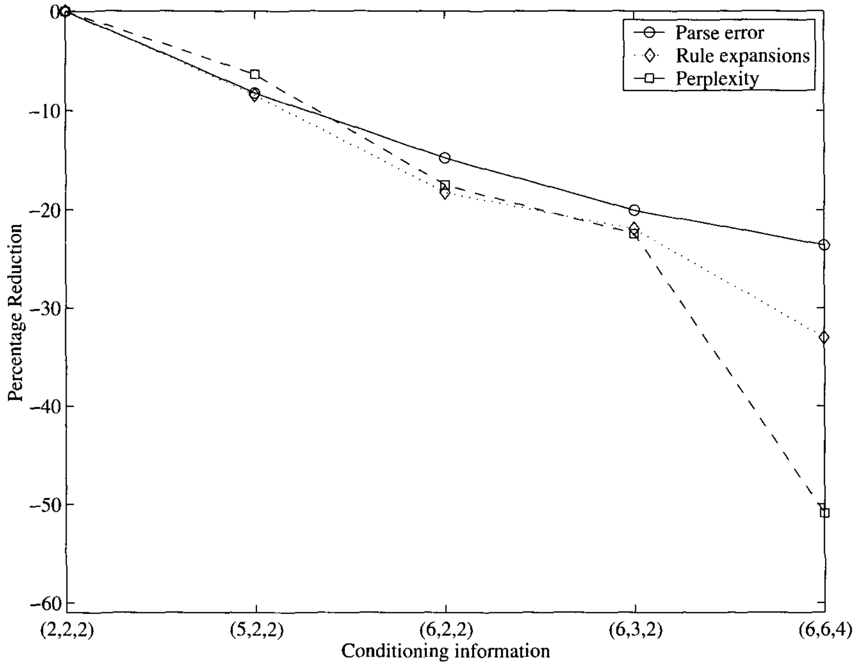

Figure 7
Reduction in average precision/recall error, number of rule expansions, and perplexity as conditioning increases.

Table 4 compares the perplexity of our model with Chelba and Jelinek (1998a, 1998b) on the same training and testing corpora. We built an interpolated trigram model to serve as a baseline (as they did), and also interpolated our model's perplexity with the trigram, using the same mixing coefficient as they did in their trials (taking 36 percent of the estimate from the trigram).[18] The trigram model was also trained on Sections 00–20 of the C&J corpus. Trigrams and bigrams were binned by the total

[18] Our optimal mixture level was closer to 40 percent, but the difference was negligible.

Table 4
Comparison with previous perplexity results.

	Perplexity		
Paper	Trigram Baseline	Model	Interpolation, $\lambda = .36$
Chelba and Jelinek (1998a)	167.14	158.28	148.90
Chelba and Jelinek (1998b)	167.14	153.76	147.70
Current results	167.02	152.26	137.26

count of the conditioning words in the training corpus, and maximum likelihood mixing coefficients were calculated for each bin, to mix the trigram with bigram and unigram estimates. Our trigram model performs at almost exactly the same level as theirs does, which is what we would expect. Our parsing model's perplexity improves upon their first result fairly substantially, but is only slightly better than their second result.[19] However, when we interpolate with the trigram, we see that the additional improvement is greater than the one they experienced. This is not surprising, since our conditioning information is in many ways orthogonal to that of the trigram, insofar as it includes the probability mass of the derivations; in contrast, their model in some instances is very close to the trigram, by conditioning on two words in the prefix string, which may happen to be the two adjacent words.

These results are particularly remarkable, given that we did not build our model as a language model per se, but rather as a parsing model. The perplexity improvement was achieved by simply taking the existing parsing model and applying it, with no extra training beyond that done for parsing.

The hope was expressed above that our reported perplexity would be fairly close to the "true" perplexity that we would achieve if the model were properly normalized, i.e., that the amount of probability mass that we lose by pruning is small. One way to test this is the following: at each point in the sentence, calculate the conditional probability of each word in the vocabulary given the previous words, and sum them.[20] If there is little loss of probability mass, the sum should be close to one. We did this for the first 10 sentences in the test corpus, a total of 213 words (including the end-of-sentence markers). One of the sentences was a failure, so that 12 of the word probabilities (all of the words after the point of the failure) were not estimated by our model. Of the remaining 201 words, the average sum of the probabilities over the 10,000-word vocabulary was 0.9821, with a minimum of 0.7960 and a maximum of 0.9997. Interestingly, at the word where the failure occurred, the sum of the probabilities was 0.9301.

5.4 Word Error Rate

In order to get a sense of whether these perplexity reduction results can translate to improvement in a speech recognition task, we performed a very small preliminary experiment on *n*-best lists. The DARPA '93 HUB1 test setup consists of 213 utterances read from the *Wall Street Journal*, a total of 3,446 words. The corpus comes with a baseline trigram model, using a 20,000-word open vocabulary, and trained on approximately 40 million words. We used Ciprian Chelba's A* decoder to find the 50 best hypotheses from each lattice, along with the acoustic and trigram scores.[21] Given

[19] Recall that our perplexity measure should, ideally, be even lower still.
[20] Thanks to Ciprian Chelba for this suggestion.
[21] See Chelba (2000) for details on the decoder.

Table 5
Word and sentence error rate results for various models, with differing training and vocabulary sizes, for the best language model factor for that particular model.

Model	Training Size	Vocabulary Size	LM Weight	Percentage Word Error Rate	Percentage Sentence Error Rate
Lattice trigram	40M	20K	16	13.7	69.0
Chelba (2000) ($\lambda = .4$)	20M	20K	16	13.0	
Current model	1M	10K	15	15.1	73.2
Treebank trigram	1M	10K	5	16.5	79.8
No language model			0	16.8	84.0

the idealized circumstances of the production (text read in a lab), the lattices are relatively sparse, and in many cases 50 distinct string hypotheses were not found in a lattice. We reranked an average of 22.9 hypotheses with our language model per utterance.

One complicating issue has to do with the tokenization in the Penn Treebank versus that in the HUB1 lattices. In particular, contractions (e.g., he's) are split in the Penn Treebank (he 's) but not in the HUB1 lattices. Splitting of the contractions is critical for parsing, since the two parts oftentimes (as in the previous example) fall in different constituents. We follow Chelba (2000) in dealing with this problem: for parsing purposes, we use the Penn Treebank tokenization; for interpolation with the provided trigram model, and for evaluation, the lattice tokenization is used. If we are to interpolate our model with the lattice trigram, we must wait until we have our model's estimate for the probability of both parts of the contraction; their product can then be interpolated with the trigram estimate. In fact, interpolation in these trials made no improvement over the better of the uninterpolated models, but simply resulted in performance somewhere between the better and the worse of the two models, so we will not present interpolated trials here.

Table 5 reports the word and sentence error rates for five different models: (i) the trigram model that comes with the lattices, trained on approximately 40M words, with a vocabulary of 20,000; (ii) the best-performing model from Chelba (2000), which was interpolated with the lattice trigram at $\lambda = 0.4$; (iii) our parsing model, with the same training and vocabulary as the perplexity trials above; (iv) a trigram model with the same training and vocabulary as the parsing model; and (v) no language model at all. This last model shows the performance from the acoustic model alone, without the influence of the language model. The log of the language model score is multiplied by the language model (LM) weight when summing the logs of the language and acoustic scores, as a way of increasing the relative contribution of the language model to the composite score. We followed Chelba (2000) in using an LM weight of 16 for the lattice trigram. For our model and the Treebank trigram model, the LM weight that resulted in the lowest error rates is given.

The small size of our training data, as well as the fact that we are rescoring n-best lists, rather than working directly on lattices, makes comparison with the other models not particularly informative. What is more informative is the difference between our model and the trigram trained on the same amount of data. We achieved an 8.5 percent relative improvement in word error rate, and an 8.3 percent relative improvement in sentence error rate over the Treebank trigram. Interestingly, as mentioned above, interpolating two models together gave no improvement over the better of the two, whether our model was interpolated with the lattice or the Treebank trigram. This

Table 6
Results with full conditioning on the C&J corpus at various base beam factors.

Base Beam Factor	LR	LP	Percentage Failed	Perplexity $\lambda = 0$	Perplexity $\lambda = .36$	Average Rule Expansions Considered†	Words Per Second
			Sections 23–24: 3761 sentences $\leq$ 120				
10^{-11}	81.7	82.1	0.2	152.26	137.26	11,667	3.1
10^{-10}	81.5	81.9	0.3	154.25	137.88	6,982	5.2
10^{-9}	80.9	81.3	0.4	156.83	138.69	4,154	8.9
10^{-8}	80.2	80.6	0.6	160.63	139.80	2,372	15.3
10^{-7}	78.8	79.2	1.2	166.91	141.30	1,468	25.5
10^{-6}	77.4	77.9	1.5	174.44	143.05	871	43.8
10^{-5}	75.8	76.3	2.6	187.11	145.76	517	71.6
10^{-4}	72.9	73.9	4.5	210.28	148.41	306	115.5
10^{-3}	68.4	70.6	8.0	253.77	152.33	182	179.6

†per word

contrasts with our perplexity results reported above, as well as with the recognition experiments in Chelba (2000), where the best results resulted from interpolated models.

The point of this small experiment was to see if our parsing model could provide useful information even in the case that recognition errors occur, as opposed to the (generally) fully grammatical strings upon which the perplexity results were obtained. As one reviewer pointed out, given that our model relies so heavily on context, it may have difficulty recovering from even one recognition error, perhaps more difficulty than a more locally oriented trigram. While the improvements over the trigram model in these trials are modest, they do indicate that our model is robust enough to provide good information even in the face of noisy input. Future work will include more substantial word recognition experiments.

5.5 Beam Variation

The last set of results that we will present addresses the question of how wide the beam must be for adequate results. The base beam factor that we have used to this point is 10^{-11}, which is quite wide. It was selected with the goal of high parser accuracy; but in this new domain, parser accuracy is a secondary measure of performance. To determine the effect on perplexity, we varied the base beam factor in trials on the Chelba and Jelinek corpora, keeping the level of conditioning information constant, and Table 6 shows the results across a variety of factors.

The parser error, parser coverage, and the uninterpolated model perplexity ($\lambda = 1$) all suffered substantially from a narrower search, but the interpolated perplexity remained quite good even at the extremes. Figure 8 plots the percentage increase in parser error, model perplexity, interpolated perplexity, and efficiency (i.e., decrease in rule expansions per word) as the base beam factor decreased. Note that the model perplexity and parser accuracy are quite similarly affected, but that the interpolated perplexity remained far below the trigram baseline, even with extremely narrow beams.

6. Conclusion and Future Directions

The empirical results presented above are quite encouraging, and the potential of this kind of approach both for parsing and language modeling seems very promising.

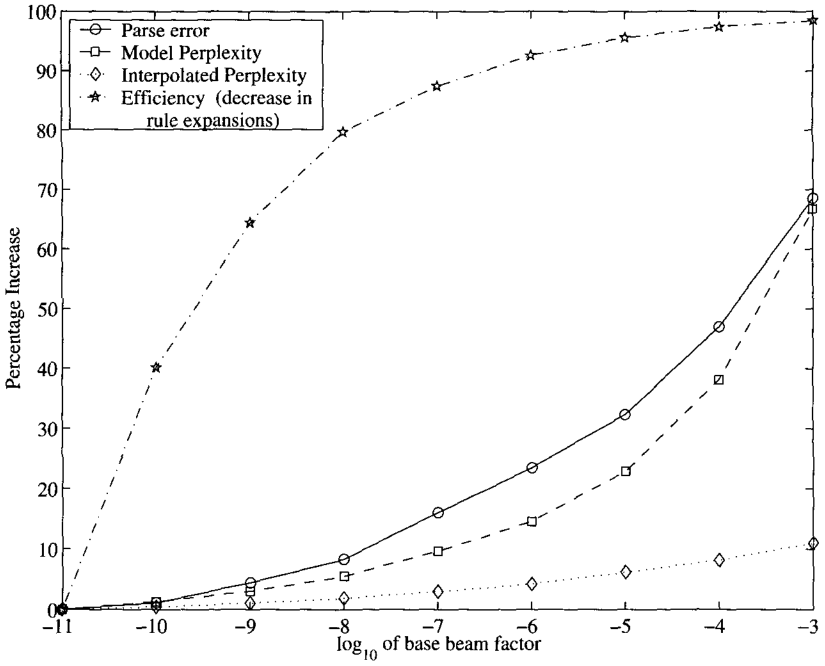

Figure 8
Increase in average precision/recall error, model perplexity, interpolated perplexity, and efficiency (i.e., decrease in rule expansions per word) as base beam factor decreases.

With a simple conditional probability model, and simple statistical search heuristics, we were able to find very accurate parses efficiently, and, as a side effect, were able to assign word probabilities that yield a perplexity improvement over previous results. These perplexity improvements are particularly promising, because the parser is providing information that is, in some sense, orthogonal to the information provided by a trigram model, as evidenced by the robust improvements to the baseline trigram when the two models are interpolated.

There are several important future directions that will be taken in this area. First, there is reason to believe that some of the conditioning information is not uniformly useful, and we would benefit from finer distinctions. For example, the probability of a preposition is presumably more dependent on a c-commanding head than the probability of a determiner is. Yet in the current model they are both conditioned on that head, as leftmost constituents of their respective phrases. Second, there are advantages to top-down parsing that have not been examined to date, e.g., empty categories. A top-down parser, in contrast to a standard bottom-up chart parser, has enough information to predict empty categories only where they are likely to occur. By including these nodes (which are in the original annotation of the Penn Treebank), we may be able to bring certain long-distance dependencies into a local focus. In addition, as mentioned above, we would like to further test our language model in speech recognition tasks, to see if the perplexity improvement that we have seen can lead to significant reductions in word error rate.

Other parsing approaches might also be used in the way that we have used a top-down parser. Earley and left-corner parsers, as mentioned in the introduction, also have rooted derivations that can be used to calculated generative string prefix probabilities incrementally. In fact, left-corner parsing can be simulated by a top-down parser by transforming the grammar, as was done in Roark and Johnson (1999), and so an approach very similar to the one outlined here could be used in that case. Perhaps some compromise between the fully connected structures and extreme underspecification will yield an efficiency improvement. Also, the advantages of head-driven parsers may outweigh their inability to interpolate with a trigram, and lead to better off-line language models than those that we have presented here.

Does a parsing model capture exactly what we need for informed language modeling? The answer to that is no. Some information is simply not structural in nature (e.g., topic), and we might expect other kinds of models to be able to better handle nonstructural dependencies. The improvement that we derived from interpolating the different models above indicates that using multiple models may be the most fruitful path in the future. In any case, a parsing model of the sort that we have presented here should be viewed as an important potential source of key information for speech recognition. Future research will show if this early promise can be fully realized.

Acknowledgments

The author wishes to thank Mark Johnson for invaluable discussion, guidance, and moral support over the course of this project. Many thanks also to Eugene Charniak for the use of certain grammar training routines, and for an enthusiastic interest in the project. Thanks also to four anonymous reviewers for valuable and insightful comments, and to Ciprian Chelba, Sanjeev Khudanpur, and Frederick Jelinek for comments and suggestions. Finally, the author would like to express his appreciation to the participants of discussions during meetings of the Brown Laboratory for Linguistic Information Processing (BLLIP); in addition to Mark and Eugene: Yasemin Altun, Don Blaheta, Sharon Caraballo, Massimiliano Ciaramita, Heidi Fox, Niyu Ge, and Keith Hall. This research was supported in part by NSF IGERT Grant #DGE-9870676.

References

Aho, Alfred V., Ravi Sethi, and Jeffrey D. Ullman. 1986. *Compilers, Principles, Techniques, and Tools*. Addison-Wesley, Reading, MA.

Black, Ezra, Steven Abney, Dan Flickenger, Claudia Gdaniec, Ralph Grishman, Philip Harrison, Donald Hindle, Robert Ingria, Frederick Jelinek, Judith Klavans, Mark Liberman, Mitchell P. Marcus, Salim Roukos, Beatrice Santorini, and Tomek Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of english grammars. In *DARPA Speech and Natural Language Workshop*, pages 306–311.

Briscoe, Ted and John Carroll. 1993. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–60.

Caraballo, Sharon and Eugene Charniak. 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24(2):275–298.

Charniak, Eugene. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 598–603, Menlo Park. AAAI Press/MIT Press.

Charniak, Eugene. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st Conference of the North American Chapter of the Association for Computational Linguistics*, pages 132–139.

Charniak, Eugene, Sharon Goldwater, and Mark Johnson. 1998. Edge-based best-first chart parsing. In *Proceedings of the Sixth Workshop on Very Large Corpora*, pages 127–133.

Chelba, Ciprian. 2000. *Exploiting Syntactic Structure for Natural Language Modeling*. Ph.D. thesis, The Johns Hopkins University.

Chelba, Ciprian and Frederick Jelinek. 1998a. Exploiting syntactic structure for language modeling. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, pages 225–231.

Chelba, Ciprian and Frederick Jelinek. 1998b. Refinement of a structured language model. In *International Conference on Advances in Pattern*

Recognition, pages 275–284.

Chelba, Ciprian and Frederick Jelinek. 1999. Recognition performance of a structured language model. In *Proceedings of the 6th European Conference on Speech Communication and Technology (Eurospeech)*, pages 1,567–1,570.

Chi, Zhiyi and Stuart Geman. 1998. Estimation of probabilistic context-free grammars. *Computational Linguistics*, 24(2):299–305.

Collins, Michael J. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting*, pages 16–23. Association for Computational Linguistics.

Collins, Michael J. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Collins, Michael J. 2000. Discriminative reranking for natural language parsing. In *The Proceedings of the 17th International Conference on Machine Learning*, pages 175–182.

Cover, Thomas M. and Joy A. Thomas. 1991. *Elements of Information Theory*. John Wiley and Sons, Inc., New York.

Earley, Jay. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 6(8):451–455.

Eisner, J. and G. Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th Annual Meeting*, pages 457–464. Association for Computational Linguistics.

Goddeau, David. 1992. Using probabilistic shift-reduce parsing in speech recognition systems. In *Proceedings of the 2nd International Conference on Spoken Language Processing*, pages 321–324.

Goodman, Joshua. 1997. Global thresholding and multiple-pass parsing. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (EMNLP-97)*, pages 11–25.

Henderson, John C. and Eric Brill. 1999. Exploiting diversity in natural language processing: Combining parsers. In *Proceedings of the Fourth Conference on Empirical Methods in Natural Language Processing (EMNLP-99)*, pages 187–194.

Hopcroft, John E. and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

Inui, Kentaro, Virach Sornlertlamvanich, Hozummi Tanaka, and Takenobu Tokunaga. 1997. A new formalization of probabilistic GLR parsing. In *Proceedings of the 5th International Workshop on Parsing Technologies*, pages 123–134.

Jelinek, Frederick. 1997. *Statistical Methods for Speech Recognition*. MIT Press, Cambridge, MA.

Jelinek, Frederick and Ciprian Chelba. 1999. Putting language into language modeling. In *Proceedings of the 6th European Conference on Speech Communication and Technology (Eurospeech)*, pages KN1–6.

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

Jelinek, Frederick and Robert L. Mercer. 1980. Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice*, pages 381–397.

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

Johnson, Mark, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting*, pages 535–541. Association for Computational Linguistics.

Jurafsky, Daniel, Chuck Wooters, Jonathan Segal, Andreas Stolcke, Eric Fosler, Gary Tajchman, and Nelson Morgan. 1995. Using a stochastic context-free grammar as a language model for speech recognition. In *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing*, pages 189–192.

Manning, Christopher D. and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA.

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

Ratnaparkhi, Adwait. 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 (EMNLP-97)*, pages 1–10.

Roark, Brian and Eugene Charniak. 2000. Measuring efficiency in high-accuracy, broad-coverage statistical parsing. In *Proceedings of the COLING-2000 Workshop on Efficiency in Large-scale Parsing Systems*, pages 29–36.

Roark, Brian and Mark Johnson. 1999. Efficient probabilistic top-down and left-corner parsing. In *Proceedings of the 37th Annual Meeting*, pages 421–428. Association for Computational Linguistics.

Rosenfeld, Ronald. 1996. A maximum entropy approach to adaptive statistical language modeling. *Computer, Speech and Language*, 10:187–228.

Rosenfeld, Ronald. 1997. A whole sentence maximum entropy language model. In *Proceedings of IEEE Workshop on Speech Recognition and Understanding*, pages 230–237.

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

Saul, Lawrence and Fernando C. N. Pereira. 1997. Aggregate and mixed-order Markov models for statistical language processing. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (EMNLP-97)*, pages 81–89.

Stolcke, Andreas. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–202.

Stolcke, Andreas and Jonathan Segal. 1994. Precise *n*-gram probabilities from stochastic context-free grammars. In *Proceedings of the 32nd Annual Meeting*, pages 74–79. Association for Computational Linguistics.