Probabilistic Feature Grammars [*]

Joshua Goodman
Harvard University
40 Oxford St.
Cambridge, MA 02138

Email: goodman@eecs.harvard.edu

Abstract

We present a new formalism, *probabilistic feature grammar* (PFG). PFGs combine most of the best properties of several other formalisms, including those of Collins, Magerman, and Charniak, and in experiments have comparable or better performance. PFGs generate features one at a time, probabilistically, conditioning the probabilities of each feature on other features in a local context. Because the conditioning is local, efficient polynomial time parsing algorithms exist for computing inside, outside, and Viterbi parses. PFGs can produce probabilities of strings, making them potentially useful for language modeling. Precision and recall results are comparable to the state of the art with words, and the best reported without words.

1 Introduction

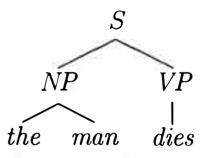
Recently, many researchers have worked on statistical parsing techniques which try to capture additional context beyond that of simple probabilistic context-free grammars (PCFGs), including Magerman (1995), Charniak (1996), Collins (1996; 1997), Black, Lafferty, and Roukos (1992), Eisele (1994) and Brew (1995). Each has tried to capture the hierarchical nature of language, as typified by context-free grammars, and to then augment this with additional context sensitivity based on various *features* of the input. Unfortunately, none of these works combines the most important benefits of all the others, and most lack a certain elegance. We have therefore tried to synthesize these works into a new formalism, *probabilistic feature grammar* (PFG). PFGs have several important properties. First, PFGs can condition on features beyond the nonterminal of each node, including features such as the head word or grammatical number of a constituent. Also, PFGs can be parsed using efficient polynomial-time dynamic programming algorithms, and learned quickly from a treebank. Finally, unlike most other formalisms, PFGs are potentially useful for language modeling or as one part of an integrated statistical system (e.g. Miller et al., 1996) or for use with algorithms requiring outside probabilities. Empirical results are encouraging: our best parser is comparable to those of Magerman (1995) and Collins (1996) when run on the same data. When we run using part-of-speech (POS) tags alone as input, we perform significantly better than comparable parsers.

2 Motivation

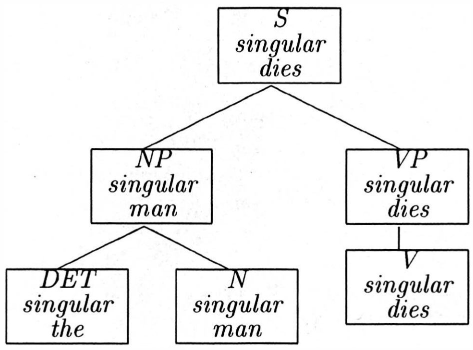
PFG can be regarded in several different ways: as a way to make history-based grammars (Magerman, 1995) more context free, and thus amenable to dynamic programming; as a way to generalize the work of Black et al. (1992); as a way to turn Collins' parser (Collins, 1996) into a generative probabilistic language model; or as an extension of language-modeling techniques to stochastic grammars. The resulting formalism, which is relatively simple and elegant, has most of the advantages of each of the systems from which it is derived.

Consider the following simple parse tree for the sentence "The man dies":

[*] This material is based upon work supported by the National Science Foundation under Grant No. IRI-9350192 and a National Science Foundation Graduate Student Fellowship. I would like to thank Stanley Chen, Lillian Lee, David Magerman, Wheeler Ruml, Stuart Shieber, and Fernando Pereira for helpful comments and discussions, as well as the anonymous reviewers for their useful comments and suggestions.



While this tree captures the simple fact that sentences are composed of noun phrases and verb phrases, it fails to capture other important restrictions. For instance, the NP and VP must have the same number, both singular, or both plural. Also, a man is far more likely to die than spaghetti, and this constrains the head words of the corresponding phrases. This additional information can be captured in a parse tree that has been augmented with features, such as the category, number, and head word of each constituent, as is traditionally done in many feature-based formalisms, such as HPSG, LFG, etc.



While a normal PCFG has productions such as

$$S \rightarrow NP\ VP$$

we will write these augmented productions as, for instance,

$$(S, singular, dies) \rightarrow (NP, singular, man)(VP, singular, dies)$$

In a traditional probabilistic context-free grammar, we could augment the first tree with probabilities in a simple fashion. We estimate the probability of $S \rightarrow NP\ VP$ using a tree bank to determine $\frac{C(S \rightarrow NP\ VP)}{C(S)}$, the number of occurrences of $S \rightarrow NP\ VP$ divided by the number of occurrences of S. For a reasonably large treebank, probabilities estimated in this way would be reliable enough to be useful (Charniak, 1996). On the other hand, it is not unlikely that we would never have seen any counts at all of

$$\frac{C((S, singular, dies) \rightarrow (NP, singular, man)(VP, singular, dies))}{C((S, singular, dies))}$$

which is the estimated probability of the corresponding production in our grammar augmented with features.

The introduction of features for number and head word has created a data sparsity problem. Fortunately, the data-sparsity problem is well known in the language-modeling community, and we can use their techniques, n-gram models and smoothing, to help us. Consider the probability of a five word sentence, $w_1...w_5$:

$$P(w_1w_2w_3w_4w_5) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_1w_2) \times P(w_4|w_1w_2w_3) \times P(w_5|w_1w_2w_3w_4)$$

While exactly computing $P(w_5|w_1w_2w_3w_4)$ is difficult, a good approximation is $P(w_5|w_1w_2w_3w_4) \approx P(w_5|w_3w_4)$.

Let $C(w_3w_4w_5)$ represent the number of occurrences of the sequence $w_3w_4w_5$ in a corpus. We can then empirically approximate $P(w_5|w_3w_4)$ with $\frac{C(w_3w_4w_5)}{C(w_4w_5)}$. Unfortunately, this approximation alone is not enough; there may still be many three word combinations that do not occur in the corpus, but that should not be assigned zero probabilities. So we *smooth* this approximation, for instance by using

$$P(w_5|w_3w_4) \approx \lambda_1 \frac{C(w_3w_4w_5)}{C(w_3w_4)} + (1-\lambda_1)\left(\lambda_2 \frac{C(w_4w_5)}{C(w_4)} + (1-\lambda_2)\frac{C(w_5)}{\sum_w C(w)}\right)$$

Now, we can use these same approximations in PFGs. Let us assume that our PFG is binary branching and has g features, numbered $1...g$; we will call the parent features a_i, the left child features b_i, and the right child features c_i. In our earlier example, a_1 represented the parent nonterminal category; a_2 represented the parent number (singular or plural); a_3 represented the parent head word; b_1 represented the left child category; etc. We can write a PFG production as $(a_1, a_2, ..., a_g) \rightarrow (b_1, b_2, ..., b_g)(c_1, c_2, ..., c_g)$. If we think of the set of features for a constituent A as being the random variables $A_1, ..., A_g$, then the probability of a production is the conditional probability

$$P(B_1 = b_1, ..., B_g = b_g, C_1 = c_1, ..., C_g = c_g | A_1 = a_1, ..., A_g = a_g)$$

We write a_1^k to represent $A_1 = a_1, ..., A_k = a_k$, and sometimes write a_i as shorthand for $A_i = a_i$. We can then write this conditional probability as

$$P(b_1^g, c_1^g | a_1^g)$$

This joint probability can be factored as the product of a set of conditional probabilities in many ways. One simple way is to arbitrarily order the features as $b_1, ..., b_g, c_1, ..., c_g$. We then condition each feature on the parent features and all features earlier in the sequence.

$$P(b_1^g, c_1^g | a_1^g) = P(b_1 | a_1^g) \times P(b_2 | a_1^g, b_1^1) \times P(b_3 | a_1^g, b_1^2) \times \cdots \times P(c_g | a_1^g, b_1^g, c_1^{g-1})$$

We can now approximate the various terms in the factorization by making independence assumptions. For instance, returning to the concrete example above, consider feature c_1, the right child nonterminal or terminal category. The following approximation should work fairly well in practice:

$$P(c_1 | a_1^g b_1^g) \approx P(c_1 | a_1, b_1)$$

That is, the category of the right child is well determined by the category of the parent and the category of the left child. Just as n-gram models approximate conditional lexical probabilities by assuming independence of words that are sufficiently distant, here we approximate conditional feature probabilities by assuming independence of features that are sufficiently unrelated. Furthermore, we can use the same kinds of backing-off techniques that are used in smoothing traditional language models to allow us to condition on relatively large contexts. In practice, a grammarian determines which features should be considered independent, and the optimal order of backoff, possibly using experiments on development test data for feedback. It might be possible to determine the optimal order of backoff automatically, a subject of future research.

Intuitively, in a PFG, features are produced one at a time. The probability of a feature being produced depends on a subset of the features in a local context of that feature. Figure 1 shows an example of this feature-at-a-time generation for the noun phrase "the man." To the right of the figure, the independence assumptions made by the grammarian are shown.

3 Formalism

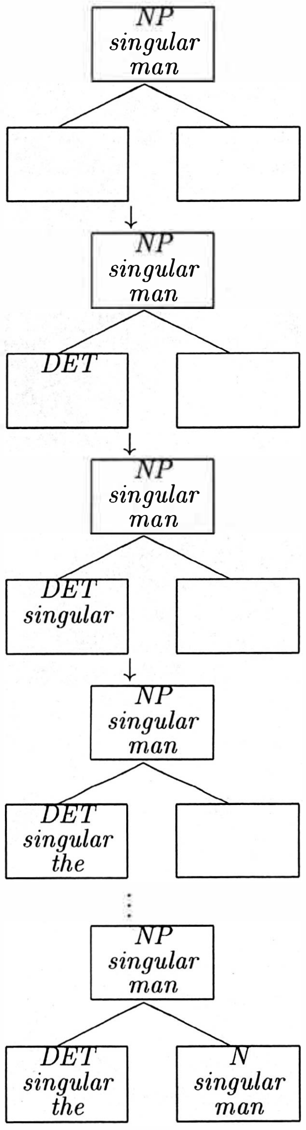
In a PCFG, the important concepts are the terminals and nonterminals, the productions involving these, and the corresponding probabilities. In a PFG, a vector of features corresponds to the terminals and nonterminals. PCFG productions correspond to PFG *events* of the form $(a_1, ..., a_g) \rightarrow (b_1, ..., b_g)(c_1, ..., c_g)$, and our PFG rule probabilities correspond to products of conditional probabilities, one for each feature that needs to be generated.

3.1 Events and EventProbs

There are two kinds of PFG events of immediate interest. The first is a *binary event*, in which a feature set a_1^g (the parent features) generates features $b_1^g c_1^g$ (the child features). Figure 1 is an example of a single such event. Binary events generate the whole tree except the start node, which is generated by a *start event*.

The probability of an event is given by an *EventProb*, which generates each new feature in turn, assigning it a conditional probability given all the known features. For instance, in a binary event, the EventProb assigns probabilities to each of the child features, given the parent features and any child features that have already been generated.

Formally, an EventProb $\mathcal{E}$ is a 3-tuple $\langle \mathcal{K}, \overline{N}, \overline{F} \rangle$, where $\mathcal{K}$ is the set of conditioning features (the Known features), $\overline{N} = N_1, N_2, ..., N_n$ is an ordered list of conditioned features (the New features), and $\overline{F} = f_1, f_2, ..., f_n$



$$P(B_1{=}DET|A_1 = NP,\ A_2{=}singular,\ A_3{=}man) \approx P(B_1{=}DET|A_1 = NP,\ A_2{=}singular)$$

$$P(B_2{=}singular|A_1 = NP,\ A_2{=}singular,\ A_3{=}man,\ B_1{=}DET) \approx P(B_2{=}singular|A_2{=}singular,\ B_1{=}DET)$$

$$P(B_3{=}the|A_1 = NP,\ A_2{=}singular, \ldots,\ B_2{=}singular) \approx P(B_3{=}the|B_1 = DET,\ A_3{=}man)$$

$$P(C_3{=}man|A_1 = NP,\ A_2{=}singular, \ldots,\ C_2{=}singular) \approx P(C_3{=}man|A_3 = man,\ A_1{=}NP)$$

Figure 1: Producing *the man*, one feature at a time

is a parallel list of functions. Each function $f_i(n_i, k_1, ..., k_k, n_1, n_2, ..., n_{i-1})$ returns $P(N_i = n_i | K_1 = k_1, ... K_k = k_k, N_1 = n_1, N_2 = n_2, ..., N_{i-1} = n_{i-1})$, the probability that feature $N_i = n_i$ given all the known features and all the lower indexed new features.

For a binary event, we may have $\mathcal{E}_B = \langle \{a_1, a_2, ..., a_g\}, \langle b_1, ..., b_g, c_1, ..., c_g \rangle, \overline{F}_B \rangle$; that is, the child features are conditioned on the parent features and earlier child features. For a start event we have $\mathcal{E}_S = \langle \{\}, \langle a_1, a_2, ..., a_g \rangle, \overline{F}_S \rangle$; i.e. the parent features are conditioned only on each other.

3.2 Terminal Function, Binary PFG

We need one last element: a function T from a set of g features to $\langle T, N \rangle$ which tells us whether a part of an event is terminal or nonterminal: the *terminal function.* A Binary PFG is then a quadruple $\langle g, \mathcal{E}_B, \mathcal{E}_S, T \rangle$: a number of features, a binary EventProb, a start EventProb, and a terminal function.

Of course, using binary events allows us to model n-ary branching grammars for any fixed n: we simply add additional features for terminals to be generated in the future, as well as a feature for whether or not this intermediate node is a "dummy" node (the continuation feature).[1]

4 Comparison to Previous Work

PFG bears much in common with previous work, but in each case has at least some advantages over previous formalisms.

Some other models (Charniak, 1996; Brew, 1995; Collins, 1996; Black, Lafferty, and Roukos, 1992) use probability approximations that do not sum to 1, meaning that they should not be used either for language modeling, e.g. in a speech recognition system, or as part of an integrated model such as that of Miller et al. (1996). Some models (Magerman, 1995; Collins, 1996) assign probabilities to parse trees conditioned on the strings, so that an unlikely sentence with a single parse might get probability 1, making these systems unusable for language modeling. PFGs use joint probabilites, so can be used both for language modeling and as part of an integrated model.

Furthermore, unlike all but one of the comparable systems, PFGs can compute outside probabilities, which are useful for grammar induction, some parsing algorithms (Goodman, 1996), and, as we will show, pruning (Goodman, 1997).

4.1 Bigram Lexical Dependency Parsing

Collins (1996) introduced a parser with extremely good performance. From this parser, we take many of the particular conditioning features that we will use in PFGs. As noted, this model cannot be used for language modeling. There are also some inelegancies in the need for a separate model for Base-NPs, and the treatment of punctuation as inherently different from words. The model also contains a non-statistical rule about the placement of commas. Finally, Collins' model uses memory proportional to the sum of the squares of each training sentence's length. PFGs in general use memory which is only linear.

4.2 Generative Lexicalized Parsing

Collins (1997) worked independently from us to construct a model similar to ours. In particular, Collins wished to adapt his previous parser (Collins, 1996) to a generative model. In this he succeeded. However, while we present a fairly simple and elegant formalism, which captures all information as features, Collins uses a variety of different techniques: variables (analogous to our features); a special stop category; modifications of nonterminal categories; and information computed as a function of child nodes. This lack of homogeneity fails to show the underlying structure of the model, and the ways it could be expanded.

Furthermore, our model of generation is very general. While our implementation captures head words through the particular choice of features, Collins' model explicitly generates first the head phrase, then the right children, and finally the left children. Thus, our model can be used to capture a wider variety of grammatical theories, simply by changing the choice of features.

[1] Unary branching events are in general difficult to deal with (Stolcke, 1993). We introduce an additional EventProb for unary events, and do not allow more than one unary event in a row.

Finally, there are some subtle interesting differences with respect to the distance metric. While both bigram lexical dependency parsing and generative lexicalized parsing use a distance metric, generative lexicalized parsing does not include symbols in the child node on the parent head side as part of the distance, because it is difficult to do so in that model. Our use of features allows this information to be captured.

4.3 Simple PCFGs

Charniak (1996) showed that a simple PCFG formalism in which the rules are simply "read off" of a treebank can perform very competitively. Furthermore, he showed that a simple modification, in which productions at the right side of the sentence have their probability boosted to encourage right branching structures, can improve performance even further. PFGs are a superset of PCFGs, so we can easily model the basic PCFG grammar used by Charniak, although the boosting cannot be exactly duplicated. However, we can use more principled techniques, such as a feature that captures whether a particular constituent is at the end of the sentence, and a feature for the length of the constituent. Charniak's boosting strategy means that the scores of constituents are no longer probabilities, meaning that they cannot be used with the inside-outside algorithm. Furthermore, the PFG feature-based technique is not extra-grammatical, meaning that no additional machinery needs to be added for parsing or grammar induction.

4.4 Stochastic HPSG

Brew (1995) introduced a stochastic version of HPSG. In his formalism, in some cases even if two features have been constrained to the same value by unification, the probabilities of their productions are assumed independent. The resulting probability distribution is then normalized so that probabilities sum to one. This leads to problems with grammar induction pointed out by Abney (1996). Our formalism, in contrast, explicitly models dependencies to the extent possible given data sparsity constraints.

4.5 IBM Language Modeling Group

Researchers in the IBM Language Modeling Group developed a series of successively more complicated models to integrate statistics with features.

The first model (Black, Garside, and Leech, 1993; Black, Lafferty, and Roukos, 1992) essentially tries to convert a unification grammar to a PCFG, by instantiating the values of the features. Due to data sparsity, however, not all features can be instantiated. Instead, they create a grammar where many features have been instantiated, and many have not; they call these partially instantiated features sets *mnemonics*. They then create a PCFG using the mnemonics as terminals and nonterminals. Features instantiated in a particular mnemonic are generated probabilistically, while the rest are generated through unification. Because no smoothing is done, and because features are grouped, data sparsity limits the number of features that can be generated probabilistically, whereas because we generate features one at a time and smooth, we are far less limited in the number of features we can use. Their technique of generating some features probabilistically, and the rest by unification, is somewhat inelegant; also, for the probabilities to sum to one, it requires an additional step of normalization, which they appear not to have implemented.

In their next model (Black et al., 1992; Magerman, 1994, pp. 46–56), which strongly influenced our model, five attributes are associated with each nonterminal: a syntactic category, a semantic category, a rule, and two lexical heads. The rules in this grammar are the same as the mnemonic rules used in the previous work, developed by a grammarian. These five attributes are generated one at a time, with backoff smoothing, conditioned on the parent attributes and earlier attributes. Our generation model is essentially the same as this. Notice that in this model, unlike ours, there are two kinds of features: those features captured in the mnemonics, and the five categories; the categories and mnemonic features are modeled very differently. Also, notice that a great deal of work is required by a grammarian, to develop the rules and mnemonics.

The third model Magerman:94a, extends the second model to capture more dependencies, and to remove the use of a grammarian. Each decision in this model can in principal depend on any previous decision and on any word in the sentence. Because of these potentially unbounded dependencies, there is no dynamic programming algorithm: without pruning, the time complexity of the model is exponential. One motivation for PFG was to capture similar information to this third model, while allowing dynamic programming. This third model uses

```
for each length $l$, shortest to longest
    for each start $s$
        for each split length $t$
            for each $b_1^g$ s.t. $chart[s, s+t, b_1^g] \neq 0$
                for each $c_1^g$ s.t. $chart[s+t, s+l, c_1^g] \neq 0$
                    for each $a_1$ consistent with $b_1^g c_1^g$
                        ⋮
                        for each $a_g$ consistent with $b_1^g c_1^g a_1^{g-1}$
                            $chart[s, s+l, a_1^g] += \mathcal{E}_B(a_1^g \rightarrow b_1^g c_1^g)$
return $\sum_{a_1^g} \mathcal{E}_S(a_1^g) \times chart[1, n+1, a_1^g])$
```

Figure 2: PFG Inside Algorithm

a more complicated probability model: all probabilities are determined using decision trees; it is an area for future research to determine whether we can improve our performance by using decision trees.

4.6 Probabilistic LR Parsing with Unification Grammars

Briscoe and Carroll describe a formalism (Briscoe and Carroll, 1993; Carroll and Briscoe, 1992) similar in many ways to the first IBM model. In particular, a context-free covering grammar of a unification grammar is constructed. Some features are captured by the covering grammar, while others are modeled only through unifications. Only simple plus-one-style smoothing is done, so data sparsity is still significant. The most important difference between the work of Briscoe and Carroll (1993) and that of Black, Garside, and Leech (1993) is that Briscoe et. al. associate probabilities with the (augmented) transition matrix of an LR Parse table; this gives them more context sensitivity than Black et. al. However, the basic problems of the two approaches are the same: data sparsity; difficulty normalizing probabilities; and lack of elegance due to the union of two very different approaches.

5 Parsing

The parsing algorithm we use is a simple variation on probabilistic versions of the CKY algorithm for PCFGs, using feature vectors instead of nonterminals (Baker, 1979; Lari and Young, 1990). The parser computes inside probabilities (the sum of probabilities of all parses, i.e. the probability of the sentence) and Viterbi probabilities (the probability of the best parse), and, optionally, outside probabilities. In Figure 2 we give the inside algorithm for PFGs. Notice that the algorithm requires time $O(n^3)$ in sentence length, but is potentially exponential in the number of children, since there is one loop for each parent feature, a_1 through a_g.

When parsing a PCFG, it is a simple matter to find for every right and left child what the possible parents are. On the other hand, for a PFG, there are some subtleties. We must loop over every possible value for each feature. At first, this sounds overwhelming, since it requires guessing a huge number of feature sets, leading to a run time exponential in the number of features; in practice, most values of most features will have zero probabilities, and we can avoid considering these. For instance, features such as the length of a constituent take a single value per cell. Many other features take on very few values, given the children. For example, we arrange our parse trees so that the head word of each constituent is dominated by one of its two children. This means that we need consider only two values for this feature for each pair of children. The single most time consuming feature is the Name feature, which corresponds to the terminals and non-terminals of a PCFG. For efficiency, we keep a list of the parent/left-child/right-child name triples which have non-zero probabilities, allowing us to hypothesize only the possible values for this feature given the children. Careful choice of features helps keep parse times reasonable.

5.1 Pruning

We use two pruning methods to speed parsing. The first is a simple beam-search method, inspired by techniques used by Collins (1996) and Charniak (1996), and described in detail by Goodman (1997). Within each cell in the parse chart, we multiply each entry's inside probability by the prior probability of the parent features of that entry, using a special EventProb, $\mathcal{E}_P$. We then remove those entries whose combined probability is too much lower than the best entry of the cell.

In speech recognition, multiple-pass recognizers (Zavaliagkos et al., 1994) have been very successful. We can use an analogous technique, multiple-pass parsing (Goodman, 1997) with either PCFGs or PFGs. We use a simple, fast grammar for the first pass, which approximates the later pass. We then remove any events whose combined inside-outside product is too low: essentially those events that are unlikely given the complete sentence. The first pass is fast, and does not slow things down much, but allows us to speed up the second pass significantly. The technique is particularly natural for PFGs, since for the first pass, we can simply use a grammar with a superset of the features from the previous pass. Actually, the features we used in our first pass were Name, Continuation, and two new features especially suitable for a fast first pass, the length of the constituent and the terminal symbol following the constituent. Since these two features are uniquely determined by the chart cell of the constituent, they are especially suitable for use in a first pass, since they provide useful information without increasing the number of elements in the chart. However, when used in our second pass, these features did not help performance, presumably because they captured information similar to that captured by other features. Multiple-pass techniques have dramatically sped up PFG parsing.

6 Experimental Results

The PFG formalism is an extremely general one that has the capability to model a wide variety of phenomena. Still, it is useful to apply the formalism to actual data, as a proof of concept.

We used two PFGs, one which used the head word feature, and one otherwise identical grammar with no word based features, only POS tags. The grammars had the features shown in Figure 4. A sample parse tree with these features is given in Figure 3.

The feature D_L is a 3-tuple, indicating the number of punctuation characters, verbs, and words to the left of a constituent's head word. To avoid data sparsity, we do not count higher than 2 punctuation characters, 1 verb, or 4 words. So, a value of $\mathrm{D}_L = 014$ indicates that a constituent has no punctuation, at least one verb, and 4 or more words to the left of its head word. D_R is a similar feature for the right side. Finally, D_B gives the numbers between the constituent's head word, and the head word of its other child. Words, verbs, and punctuation are counted as being to the left of themselves: that is, a terminal verb has one verb and one word on its left.

Notice that the continuation of the lower NP in Figure 3 is R1, indicating that it inherits its child from the right, with the 1 indicating that it is a "dummy" node to be left out of the final tree.

The probability functions we used were similar to those of Section 2. There were three important differences. The first is that in some cases, we can compute a probability exactly. For instance, if we know that the head word of a parent is "man" and that the parent got its head word from its right child, then we know that with probability 1, the head word of the right child is "man." In cases where we can compute a probability exactly, we do so. Second, we smoothed slightly differently. In particular, when smoothing a probability estimate of the form

$$p(a|bc) \approx \lambda \frac{C(abc)}{C(bc)} + (1-\lambda)p(a|b)$$

we set $\lambda = \frac{C(bc)}{k+C(bc)}$, using a separate k for each probability distribution. Finally, we did additional smoothing for words, adding counts for the unknown word.

It would take quite a bit of space to give the table that shows for each feature the order of backoff for that feature. We instead discuss a single example, the order of backoff for the 2_R feature, the category of the second child of the right feature set. The least relevant features are first: $W_R, P_R, C_L, N_L, C_P, N_P, 1_R, H_R, C_R, N_R$. We back off from the head word feature first, because, although relevant, this feature creates significant data sparsity. Notice that in general, features from the same feature set, in this case the right features, are kept longest; parent features are next most relevant; and sibling features are considered least relevant. We leave out

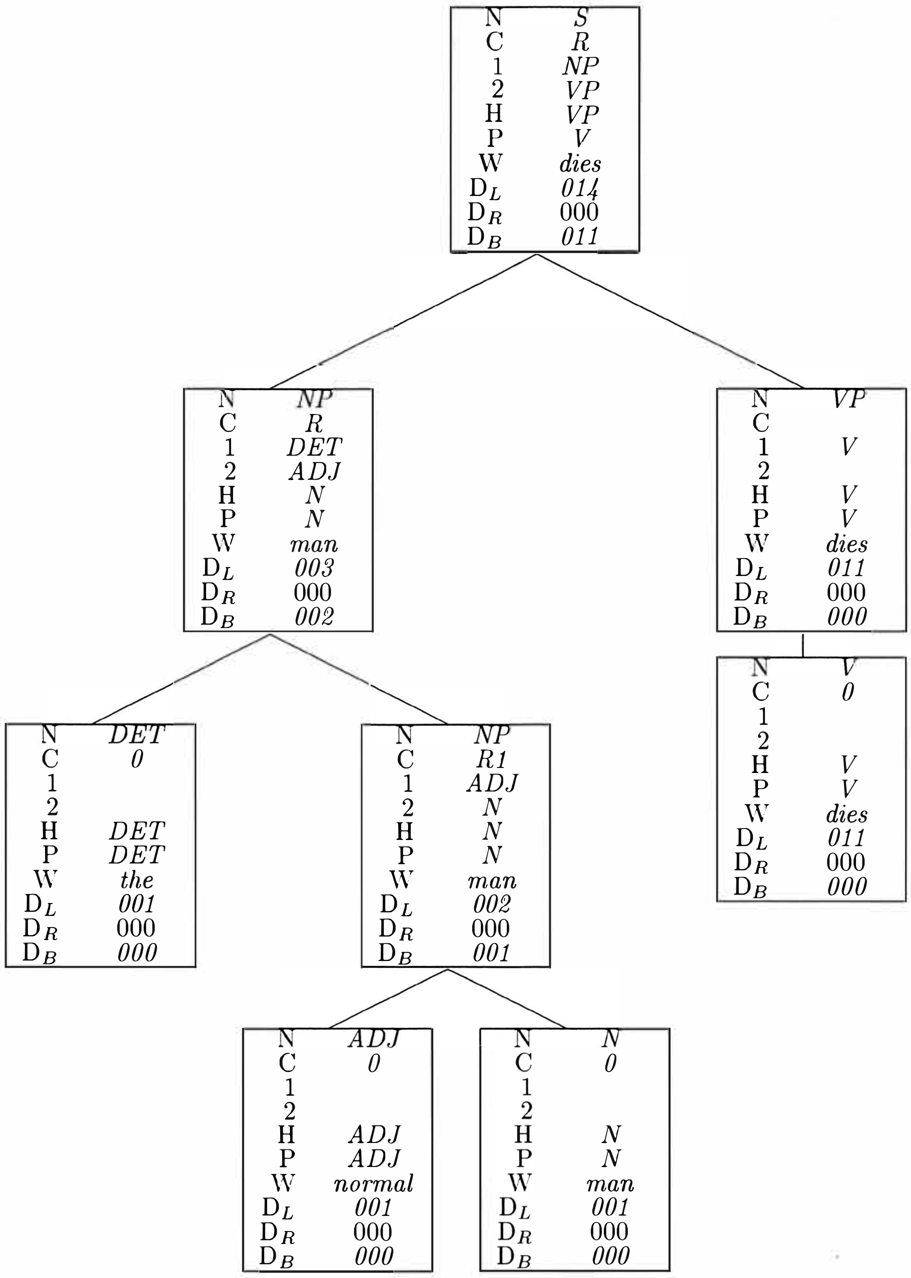


Figure 3: Example tree with features: The normal man dies

N	**Name** Corresponds to the terminals and nonterminals of a PCFG.
C	**Continuation** Tells whether we are generating modifiers to the right or the left, and whether it is time to generate the head node.
1	**Child1** Name of first child to be generated.
2	**Child2** Name of second child to be generated. In combination with Child1, this allows us to simulate a second order Markov process on nonterminal sequences.
H	**Head name** Name of the head category.
P	**Head pos** Part of speech of head word.
W	**Head word** Actual head word. Not used in POS only model.
$\mathbf{D}_L$	Δ **left** Count of punctuation, verbs, words to left of head.
$\mathbf{D}_R$	Δ **right** Counts to right of head.
$\mathbf{D}_B$	Δ **between** Counts between parent's and child's heads.

Figure 4: Features Used in Experiments

entirely features that are unlikely to be relevant and that would cause data sparsity, such as W_L, the head word of the left sibling.

6.1 Results

We used the same machine-labeled data as Collins (1996; 1997): TreeBank II sections 2-21 for training, section 23 for test, section 00 for development, using all sentences of 40 words or less.[2] We also used the same scoring method (replicating even a minor bug for the sake of comparison).

Our results are the best we know of from POS tags alone, and, with the head word feature, fall between the results of Collins and Magerman, as given by Collins (1997).

Model	Labeled Recall	Labeled Precision	Crossing Brackets	0 Crossing Brackets	≤ 2 Crossing Brackets
PFG Words	84.8%	85.3%	1.21	57.6%	81.4%
PFG POS only	81.0%	82.2%	1.47	49.8%	77.7%
Collins 97 best	88.1%	88.6%	0.91	66.5%	86.9%
Collins 96 best	85.8%	86.3%	1.14	59.9%	83.6%
Collins 96 POS only	76.1%	76.6%	2.26		
Magerman	84.6%	84.9%	1.26	56.6%	81.4%

7 Conclusions and Future Work

While the empirical performance of probabilistic feature grammars is very encouraging, we think there is far more potential. First, for grammarians wishing to integrate statistics into more conventional models, the features of PFG are a very useful tool, corresponding to the features of DCG, LFG, HPSG, and similar formalisms. TreeBank II is annotated with many semantic features, currently unused in all but the simplest way by all systems; it should be easy to integrate these features into a PFG.

PFG has other benefits that we would like to explore, including the possibility of its use as a language model, for applications such as speech recognition. Furthermore, the dynamic programming used in the model is amenable to efficient rescoring of lattices output by speech recognizers.

[2]We are grateful to Michael Collins and Adwait Ratnaparkhi for supplying us with the part-of-speech tags.

Another benefit of PFG is that both inside and outside probabilities can be computed, making it possible to reestimate PFG parameters. We would like to try experiments using PFGs and the inside/outside algorithm to estimate parameters from unannotated text.

While the generality and elegance of the PFG model makes these and many other experiments possible, we are also encouraged by the very good experimental results. Wordless model performance is excellent, and the more recent models with words are comparable to the state of the art.

References

Abney, Steve. 1996. Stochastic attribute-value grammars. Available as cmp-lg/9610003, October.

Baker, J.K. 1979. Trainable grammars for speech recognition. In *Proceedings of the Spring Conference of the Acoustical Society of America*, pages 547–550, Boston, MA, June.

Black, Ezra, George Garside, and Geoffrey Leech. 1993. *Statistically-Driven Computer Grammars of English: the IBM/Lancaster Approach*, volume 8 of *Language and Computers: Studies in Practical Linguistics.* Rodopi, Amsterdam.

Black, Ezra, Frederick Jelinek, John Lafferty, David M. Magerman, Robert Mercer, and Salim Roukos. 1992. Towards history-based grammars: Using printer models for probabilistic parsing. In *Proceedings of the February 1992 DARPA Speech and Natural Language Workshop.*

Black, Ezra, John Lafferty, and Salim Roukos. 1992. Development and evaluation of a broad-coverage probabilistic grammar of english-language computer manuals. In *Proceedings of the 30th Annual Meeting of the ACL*, pages 185–192.

Brew, Chris. 1995. Stochastic HPSG. In *Proceedings of the Seventh Conference of the European Chapter of the ACL*, pages 83–89, Dublin, Ireland, March.

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

Carroll, John and Ted Briscoe. 1992. Probabilistic normalisation and unpacking of packed parse forests for unification-based grammars. In *Proceedings of the AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, pages 33–38, Cambridge, MA.

Charniak, Eugene. 1996. Tree-bank grammars. Technical Report CS-96-02, Department of Computer Science, Brown University. Available from `ftp://ftp.cs.brown.edu/pub/techreports/96/cs96-02.ps.Z` .

Collins, Michael. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 184–191, Santa Cruz, CA, June.

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

Eisele, Andreas. 1994. Towards probabilistic extensions of constraint-based grammars. DYANA-2 Deliverable R1.2.B, September. Available from `ftp://ftp.ims.uni-stuttgart.de/pub/papers/DYANA2/R1.2.B`.

Goodman, Joshua. 1996. Parsing algorithms and metrics. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 177–183, Santa Cruz, CA, June.

Goodman, Joshua. 1997. Global thresholding and multiple-pass parsing. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing.*

Lari, K. and S.J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56.

Magerman, David. 1994. *Natural Language Parsing as Statistical Pattern Recognition.* Ph.D. thesis, Stanford University University, February.

Magerman, David. 1995. Statistical decision–models for parsing. In *Proceedings of the 33rd Annual Meeting of the ACL*, pages 276–283, Cambridge, MA.

Miller, Scott, David Stallard, Robert Bobrow, and Richard Schwartz. 1996. A fully statistical approach to natural language interfaces. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 55–61, Santa Cruz, CA, June.

Stolcke, Andreas. 1993. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. Technical Report TR-93-065, International Computer Science Institute, Berkeley, CA.

Zavaliagkos, G., T. Anastasakos, G. Chou, C. Lapre, F. Kubala, J. Makhoul, L. Nguyen, R. Schwartz, and Y. Zhao. 1994. Improved search, acoustic and language modeling in the BBN Byblos large vocabulary CSR system. In *Proceedings of the ARPA Workshop on Spoken Language Technology*, pages 81–88, Plainsboro, New Jersey.