A Maximum-Entropy-Inspired Parser [*]

Eugene Charniak
Department of Computer Science
Brown University, Box 1910, Providence RI
ec@cs.brown.edu

August 4, 1999

Abstract

We present a new parser for parsing down to Penn tree-bank style parse trees that achieves 90.1% average precision/recall for sentences of length 40 and less, and 89.5% for sentences of length 100 and less when trained and tested on the previously established [5,9,10,15,17] "standard" sections of the Wall Street Journal tree-bank. This represents a 15% decrease in error rate over the best single-parser results on this corpus [10]. The major technical innovation in this parser is the use of a "maximum-entropy-inspired" model for conditioning and smoothing that allowed us successfully to test and combine many different conditioning events. We also present some partial results showing the effects of different conditioning information, including a surprising 2% improvement due to guessing the lexical head's pre-terminal before guessing the lexical head.

1 Introduction

We present a new parser for parsing down to Penn tree-bank style parse trees [16] that achieves 90.1% average precision/recall for sentences of length ≤ 40, and 89.5% for sentences of length ≤ 100, when trained and tested on the previously established [5,9,10,15,17] "standard" sections of the Wall Street Journal tree-bank. This represents a 15% decrease in error rate over the best single-parser results on this corpus [10].

Following [5,10], our parser is based upon a probabilistic generative model. That is, for all sentences s and all parses π, the parser assigns a

[*]This research was supported in part by NSF grant LIS SBR 9720368. The author would like to thank Mark Johnson and all the rest of the Brown NLP group.

probability $p(s, \pi) = p(\pi)$, the equality holding when we restrict consideration to π whose yield is s. Then for any sentence s the parser returns the parse π that maximizes this probability. That is, the parser implements the function

$$\begin{aligned} \arg max_{\pi} p(\pi \mid s) &= \arg max_{\pi} p(\pi, s) & (1)\\ &= \arg max_{\pi} p(\pi). & (2) \end{aligned}$$

What fundamentally distinguishes one probabilistic generative parser from another is how it computes $p(\pi)$, and it is to that topic we turn next.

2 The Generative Model

The model assigns a probability to a parse by a top-down process of considering each constituent c in π and for each c first guessing the pre-terminal of c, $t(c)$ (t for "tag"), then the lexical head of c, $h(c)$, and then the expansion of c into further constituents $e(c)$. Thus the probability of a parse is given by the equation

$$\begin{aligned} p(\pi) = \prod_{c \in \pi} & p(t(c) \mid l(c), H(c)) \cdot p(h(c) \mid t(c), l(c), H(c)) \cdot \\ & p(e(c) \mid l(c), t(c), h(c), H(c)) \qquad (3) \end{aligned}$$

where $l(c)$ is the label of c (e.g., is it a noun phrase (np), verb-phrase, etc) and $H(c)$ is the relevant history of c — information outside c that our probability model deems important in determining the probability in question. Much of the interesting work is determining what goes into $H(c)$. Since $H(c)$, the extra information, differs for each of the three probabilities on the right-hand side of Equation 3, strictly speaking we should subscript H, but as it is just a placeholder until we achieve the requisite level of detail, we omit the subscripts for simplicity. Also, whenever it is clear to which constituent we are referring we omit the (c) in, e.g., $h(c)$. In this notation the above equation takes the following form:

$$p(\pi) = \prod_{c \in \pi} p(t \mid l, H) \cdot p(h \mid t, l, H) \cdot p(e \mid l, t, h, H). \qquad (4)$$

Next we describe how we assign a probability to the expansion e of a constituent. In Section 5 we present some results in which the possible expansions of a constituent are fixed in advanced by extracting a tree-bank grammar [3] from the training corpus. The method that gives the best results, however, uses a Markov grammar — a method for assigning probabilities to any possible expansion using statistics gathered from the training corpus [6,10,15]. The method we use follows that of [10]. In this scheme a traditional probabilistic context-free grammar (PCFG) rule can be thought of as consisting of a left-hand side with a label $l(c)$ drawn from the non-terminal symbols of our grammar, and a right-hand side that is a sequence of one or more such symbols. (We assume that all terminal symbols are generated by rules of the form "*preterm* $\rightarrow$ *word*" and we treat these as a special case.) For us the non-terminal symbols are those of the tree-bank, augmented by the symbols **aux** and **auxg** which have been assigned deterministically to certain auxiliary verbs such as "have" or "having". For each expansion we distinguish one of the right-hand side labels as the "middle" or "head" symbol $M(c)$. $M(c)$ is the constituent from which the head lexical item h is obtained according to deterministic rules that pick the head of a constituent from among the heads of its children. To the left of M is a sequence of one or more left labels $L_i(c)$ including the special termination symbol $\triangle$, which indicates that there are no more symbols to the left, and similarly for the labels to the right, $R_i(c)$. Thus an expansion $e(c)$ looks like:

$$l \rightarrow \triangle L_m ... L_1 M R_1 ... R_n \triangle \tag{5}$$

The expansion is generated by guessing first M, then in order L_1 through L_{m+1} ($= \triangle$), and similarly for R_1 through R_{n+1}.

In a pure Markov PCFG we are given the left-hand side label l and then probabilistically generate the right-hand side conditioning on no information other than l and (possibly) previously generated pieces of the right-hand side itself. In the simplest of such models, a zero-order Markov grammar, each label on the right-hand side is generated conditioned only on l — that is, according to the distributions $p(L_i \mid l)$, $p(M \mid l)$, and $p(R_i \mid l)$.

More generally, one can condition on the m previously generated labels, thereby obtaining an mth-order Markov grammar. So, for example, in a second-order Markov PCFG, L_2 would be conditioned on L_1 and M. In our complete model, of course, the probability of each label in the expansions is also conditioned on other material as specified in Equation 4, e.g., $p(e \mid l, t, h, H)$. Thus we would use $p(L_2 \mid L_1, M, l, t, h, H)$. Note that L_1 is conditioned on M but not on R_1, because when we guess L_1 we have seen only M. The second conditioning label in this case is the ubiquitous $\triangle$ symbol. On the other hand, the label R_1 would be conditioned on M and L_1. Also note that the $\triangle$s on both ends of the expansion in Expression 5 are conditioned just like any other label in the expansion.

3 Maximum-Entropy-Inspired Parsing

The major problem confronting the author of a generative parser is what information to use to condition the probabilities required in the model, and how to smooth the empirically obtained probabilities to take the sting out of the sparse data problems that are inevitable with even the most modest conditioning. For example, in a second-order Markov grammar we conditioned the L_2 label according to the distribution $p(L_2 \mid L_1, M, l, t, h, H)$. Also, remember that H is a placeholder for any other information beyond the constituent c that may be useful in assigning c a probability.

In the past few years the maximum entropy, or log-linear, approach has recommended itself to probabilistic model builders for its flexibility and its novel approach to smoothing [1,17]. A complete review of log-linear models is beyond the scope of this paper. Rather, we concentrate on the aspects of these models which most directly influenced the model presented here.

To compute a probability in a log-linear model one first defines a set of "features", functions from the space of configurations over which one is trying to compute probabilities to integers that denote the number of times some pattern occurs in the input. In our work we assume that any feature can occur at most once, so features are boolean-valued: 0 if the pattern does not occur, 1 if it does.

In the parser we further assume that features are chosen from certain feature schema and that every feature is a boolean conjunction of sub-features. For example, in computing the probability of the head's pre-terminal t we might want a feature schema $f(t, l)$ that returns 1 if the observed pre-terminal of $c = t$ and the label of $c = l$, and zero otherwise. This feature is obviously composed of two sub-features, one recognizing t, the other l. If both return 1, then the feature returns 1.

Now consider computing a conditional probability $p(a \mid H)$ with a set of features $f_1 \ldots f_j$ that connect a to the history H. In a log-linear model the probability function takes the following form:

$$p(a \mid H) = \frac{1}{Z(H)} e^{\lambda_1(a,H) f_1(a,H) + \ldots + \lambda_m(a,H) f_m(a,H)}. \qquad (6)$$

Here the λ_i are weights between negative and positive infinity that indicate the relative importance of a feature: the more relevant the feature to the value of the probability, the higher the absolute value of the associated λ.

The function $Z(H)$, called the partition function, is a normalizing constant (for fixed H), so the probabilities over all a sum to one.

Now for our purposes it is useful to rewrite this as a sequence of multiplicative functions $g_i(a, H)$ for $0 \leq i \leq j$:

$$p(a \mid H) = g_0(a, H) g_1(a, H) \ldots g_j(a, H). \tag{7}$$

Here $g_0(a, H) = (1)/(Z(H))$ and $g_i(a, H) = e^{\lambda_i(a,H) f_i(a,H)}$. The intuitive idea is that each factor g_i is larger than one if the feature in question makes the probability more likely, one if the feature has no effect, and smaller than one if it makes the probability less likely.

Maximum-entropy models have two benefits for a parser builder. First, as already implicit in our discussion, factoring the probability computation into a sequence of values corresponding to various "features" suggests that the probability model should be easily changeable — just change the set of features used. This point is emphasized by Ratnaparkhi in discussing of his parser [17]. Second, and this is a point we have not yet mentioned, the features used in these models need have no particular independence of one another. This is a very useful fact if one wants to use a log-linear model for smoothing. That is, suppose we want to compute a conditional probability $p(a \mid b, c)$, but we are not sure that we has enough examples of the conditioning event b, c in the training corpus to ensure that the empirically obtained probability $\hat{p}(a \mid b, c)$ is accurate. The traditional way to handle this is also to compute $\hat{p}(a \mid b)$, and perhaps $\hat{p}(a \mid c)$ as well, and take some combination of these values as one's best estimate for $p(a \mid b, c)$. This method is known as "deleted interpolation" smoothing. In max-entropy models one can simply include features for all three events $f_1(a, b, c)$, $f_2(a, b)$, and $f_3(a, c)$ and combine them in the model according to Equation 6, or equivalently, Equation 7. The fact that the features are very far from independent is not a concern.

Now let us note that we can get an equation of exactly the same form as Equation 7 in the following fashion:

$$p(a \mid b, c, d) = p(a \mid b) \frac{p(a \mid b, c)}{p(a \mid b)} \frac{p(a \mid b, c, d)}{p(a \mid b, c)}. \tag{8}$$

Note that the first term of the equation gives a probability based upon little conditioning information and that each subsequent term is a number from zero to positive infinity that is greater or smaller than one if the new information being considered makes the probability greater or smaller than the previous estimate.

As it stands, this last equation is pretty much content free. But let us look at how it works for a particular case in our parsing scheme. Consider the probability distribution for choosing the pre-terminal for the head of a constituent. In Equation 4 we wrote this as $p(t \mid l, H)$. As we discuss in more detail in Section 5 several different features in the context surrounding c are useful to include in H: the label, head pre-terminal and head of the parent of c (denoted as l_p, t_p, h_p), the label of c's left sibling (l_b for "before"), and the label of the grandparent of c (l_g). That is we wish to compute $p(t \mid l, l_p, t_p, l_b, l_g, h_p)$. We can now rewrite this in the form of Equation 8 as follows:

$$\begin{aligned} p(t \mid l, l_p, t_p, l_b, l_g, h_p) &= p(t \mid l)\frac{p(t \mid l, l_p)}{p(t \mid l)}\frac{p(t \mid l, l_p, t_p)}{p(t \mid l, l_p)}\frac{p(t \mid l, l_p, t_p, l_b)}{p(t \mid l, l_p, t_p)} \\ &\quad \frac{p(t \mid l, l_p, t_p, l_b, l_g)}{p(t \mid l, l_p, t_p, l_b)}\frac{p(t \mid l, l_p, t_p, l, l_g, h_p)}{p(t \mid l, l_p, t_p, l_b, l_g)}. \end{aligned} \tag{9}$$

Here we have sequentially conditioned on steadily increasing portions of c's history. In many cases this is clearly warranted. For example, it does not seem to make much sense to condition on, say, h_p without first conditioning on t_p. In other cases, however, we seem to be conditioning on apples and oranges, so to speak. For example, one can well imagine that one might want to condition on the parent's lexical head without conditioning on the left sibling, or the grandparent label. One way to do this is to modify the simple version shown in Equation 9 to allow this:

$$\begin{aligned} p(t \mid l, l_p, t_p, b, l_g, h_p) &= p(t \mid l)\frac{p(t \mid l, l_p)}{p(t \mid l)}\frac{p(t \mid l, l_p, t_p)}{p(t \mid l, l_p)}\frac{p(t \mid l, l_p, t_p, b)}{p(t \mid l, l_p, t_p)} \\ &\quad \frac{p(t \mid l, l_p, t_p, l_g)}{p(t \mid l, l_p, t_p)}\frac{p(t \mid l, l_p, t_p, h_p)}{p(t \mid l, l_p, t_p)}. \end{aligned} \tag{10}$$

Note the changes to the last three terms in Equation 10. Rather than conditioning each term on the previous ones, they are now conditioned only on those aspects of the history that seem most relevant. The hope is that by doing this we will have less problem with the splitting of conditioning events, and thus a somewhat lesser problem with sparse data.

We make one more point on the connection of Equation 10 to a maximum entropy formulation. Suppose we were, in fact, going to compute a true maximum entropy model based upon the features used in Equation 10, $f_1(t, l), f_2(t, l, l_p), f_3(t, l, l_p) \ldots$. This requires finding the appropriate λ_is for Equation 6, which is accomplished using an algorithm such as iterative scaling [11] in which values for the λ_i are initially "guessed" and then modified until they converge on stable values. With no prior knowledge of values for the λ_i one traditionally starts with $\lambda_i = 0$, this being a neutral assumption that the feature has neither a positive or negative impact on the probability in question. With some prior knowledge, non-zero values can greatly speed up this process because fewer iterations are required for convergence. We comment on this because in our example we can substantially speed the process by choosing values picked so that, when the maximum-entropy equation is expressed in the form of Equation 7, the g_i have as their initial values the values of the corresponding terms in Equation 10. (Our experience is that rather than requiring 50 or so iterations, three or four suffice.) Now we observe that if we were to use a maximum-entropy approach with but run iterative scaling zero times, we would, in fact, just have Equation 10.

The major advantage of using Equation 10 is that one can generally get away without computing the partition function $Z(H)$. In the simple (content-free) form (Equation 9), it is clear that $Z(H) = 1$. In the more interesting version, Equation 10, this is not true in general, but one would not expect it to differ much from one, and we assume that as long as we are not publishing the raw probabilities (as we would be doing, for example, in publishing perplexity results) the difference from one should be unimportant. As partition-function calculation is typically the major on-line computational problem for maximum-entropy models, this simplifies the model significantly.

Naturally, the distributions required by Equation 10 cannot be used without smoothing. In a pure maximum entropy model this is done by feature selection. That is, only a small subset of all possible features is used, since to use them all would set the probability distribution to those observed in the training training corpus for all features, and this would be no smoothing at all. So, for example, in Ratnaparkhi's maximum-entropy parser [17] only features that have occurred five or more times in the training corpus are allowed.

While we can smoothed in the same fashion, we choose instead to use standard deleted interpolation. (Actually, we use a minor variation described in [4].) As we use it to get a distribution with n conditioning events, we interpolate the observed distribution with the probability given $n-1$ conditioning events, using $\lambda_n(b_{1,n})$ as our interpolation factor, where $b_{1,n} = b_1, \ldots b_n$, the n conditioning events. We write this more formally as:

$$p(a \mid b_{1,n}) = \lambda_n(b_{1,n})\hat{p}(a \mid b_{1,n}) + (1 - \lambda_n(b_{1,n}))p(a \mid b_{1,n-1}). \qquad (11)$$

Naturally $p(a \mid b_{1,n-1})$ is recursively smoothed in the same fashion.

While at first glance computing smoothed distributions for all the quantities in Equation 10 according to Equation 11 might seem computationly intensive, in practice it is not too bad because all the smoothed probabilities in Equation 11 are themselves terms in Equation 10.

4 The Experiment

We created a parser based upon the maximum-entropy-inspired model of the last section, smoothed using standard deleted interpolation as in Equation 11. As the generative model is top-down and we use a standard bottom-up best-first probabilistic chart parser [2,7], we use the chart parser as a first pass to generate candidate possible parses to be evaluated in the second pass by our probabilistic model. For runs with the generative model based upon Markov grammar statistics, the first pass uses the same statistics, but conditioned only on standard PCFG information. This allows the second pass to see expansions not present in the training corpus.

We use the gathered statistics for all observed words, even those with very low counts, thought obviously our deleted interpolation smoothing gives less emphasis to observed probabilities for rare words. For words that are not observed in the training data we guess their pre-terminals using statistics on capitalization, hyphenation, word-endings (the last two letters), and the probability that a given pre-terminal is realized using a previously unobserved word.

As noted above, the probability model uses five smoothed probability distributions, one each for L_i, M, R_i, t, and h. The equation for the (unsmoothed) conditional probability distribution for t is given in Equation 10. The other four equations are as follows, with the equations for right labels R_i identical to those for L_i once R is uniformly substituted for L:

$$\begin{aligned} p(M \mid l, t, l_p, t_p, b, l_g, h) \quad = \quad & p(M \mid l)\frac{p(M \mid l, t)}{p(M \mid l)}\frac{p(M \mid l, t, l_p)}{p(M \mid l, t)}\frac{p(M \mid l, t, l_p, t_p)}{p(M \mid l, t, l_p)} \\ & \frac{p(M \mid l, t, l_p, t_p, l_b)}{p(M \mid l, t, l_p, t_p)}\frac{p(M \mid l, t, l_p, t_p, l_g)}{p(M \mid l, t, l_p, t_p)} \\ & \frac{p(M \mid l, t, l_p, h)}{p(M \mid l, t, l_p)} \end{aligned} \tag{12}$$

$$\begin{aligned} p(h \mid t, l, l_p, t_p, l_g, h_p) \quad = \quad & p(h \mid t)\frac{p(h \mid t, l)}{p(h \mid t)}\frac{p(h \mid t, l, l_p)}{p(h \mid t, l)}\frac{p(h \mid t, l, l_p, t_p)}{p(h \mid t, l, l_p)} \\ & \frac{p(h \mid t, l, l_p, t_p, l_g)}{p(h \mid t, l, l_p, t_p)}\frac{p(h \mid t, l, l_p, t_p, h_p)}{p(h \mid t, l, l_p, t_p)} \end{aligned} \tag{13}$$

$$
\begin{aligned}
&p(L_i \mid l, L_{i-1}, L_{i-2}, L_{i-3}, l_p, t_p, l_g, h_p) \\
&= p(L_i \mid l) \frac{p(L_i \mid l, L_{i-1})}{p(L_i \mid l)} \frac{p(L_i \mid l, L_{i-1}, L_{i-2})}{p(L_i \mid l, L_{i-1})} \frac{p(L_i \mid l, L_{i-1}, L_{i-2}, L_{i-3})}{p(L_i \mid l, L_{i-1}, L_{i-2})} \\
&\quad \frac{p(L_i \mid l, L_{i-1}, L_{i-2}, L_{i-3}, l_p)}{p(L_i \mid l, L_{i-1}, L_{i-2}, L_{i-3})} \frac{p(L_i \mid l, L_{i-1}, L_{i-2}, L_{i-3}, l_p, t_p)}{p(L_i \mid l, L_{i-1}, L_{i-2}, L_{i-3}, l_p)} \\
&\quad \frac{p(L_i \mid l, L_{i-1}, L_{i-2}, L_{i-3}, l_p, t_p, l_g)}{p(L_i \mid l, L_{i-1}, L_{i-2}, L_{i-3}, l_p, t_p)} \\
&\quad \frac{p(L_i \mid l, L_{i-1}, L_{i-2}, L_{i-3}, l_p, t_p, h_p)}{p(L_i \mid l, L_{i-1}, L_{i-2}, L_{i-3}, l_p, t_p)}. \qquad (14)
\end{aligned}
$$

Note in particular that, since L and R are conditioned on three previous labels, we are using a third-order Markov grammar. The reader might also observe that the label of the parent constituent l_p is conditioned upon even when it is not obviously related to the further conditioning events. This is due to the importance of this factor in parsing, as noted in, e.g., [14].

In keeping with the standard methodology [5,9,10,15,17], we used the Penn Wall Street Journal tree-bank [16] with sections 2-21 for training, section 23 for testing, and section 24 for development (debugging and tuning).

Performance on the test corpus is measured using the standard measures from [5,9,10,17]. In particular, we measure labeled precision (LP) and recall (LR), average number of cross-brackets per sentence (CB), percentage of sentences with zero cross brackets (0CB) , and percentage of sentences with ≤ 2 cross brackets (2CB). Again as standard, we take separate measurements for all sentence of length ≤ 40 and all sentences of length ≤ 100. Note that the definitions of labeled precision and recall are those defined in [9] and used in all of the previous work. As noted in [5], these definitions typically give results about 0.4% higher than the more ovbious ones. The results for the new parser as well as for the previous top-three individual parsers on this corpus are given in Figure 1. As is typical, all of the standard measures tell pretty much the same story, with the new parser outperforming the other three parsers. Looking in particular at the precision and recall figures, the new parser's give us a 15% error reduction over the best of the previous work, Coll97 [10].

5 Discussions

In the previous sections we have concentrated on the relation of the parser to a maximum-entropy approach, the aspect of the parser which is most

Parser	LR	LP	CB	0CB	2CB
	$\leq$ 40 words (2245 sentences)				
Char97	87.5	87.4	1.00	62.1	86.1
Coll97	87.9	88.7	0.89	66.3	87.2
Char99	90.1	90.1	0.74	70.1	89.6
	$\leq$ 100 words (2416 sentences)				
Char97	86.7	86.6	1.20	59.9	83.2
Coll97	87.2	88.1	1.06	63.7	85.0
Ratna99	86.3	87.5			
Char99	89.6	89.5	0.88	67.6	87.7

Figure 1: Parsing results compared with previous work

novel. However, we do not think this aspect is the sole, or even the most important reason for its comparative success. Here we list what we believe to be the most important contributions and give some experimental results on how well the program behaves without them.

We take as our starting point the parser labled Char97 in Figure 1 [5], as that is the program from which our current parser derives. That parser, as stated in Figure 1, achieves an average precision/recall of 87.5. As noted in [5], that system is based upon a "tree-bank grammar" — a grammar read directly off the training corpus. This is as opposed to the "Markov-grammar" approach used in the current parser. Also, the prior parser uses two techniques not employed in the current parser. First, it uses a clustering scheme on words to give the system a "soft" clustering of heads and sub-heads. (It is "soft" clustering in that a word can belong to more than one cluster with different weights — the weights express the probability of producing the word given that one is going to produce a word from that cluster.) Second, Char97 uses unsupervised learning in that the original system was run on about thirty million words of unparsed text, the output was taken as "correct", and statistics were collected on the resulting parses. Without these enhancements Char97 performs at the 86.6% level for sentences of length ≤ 40.

In this section we evaluate the effects of the various changes we have made by running various versions of our current program. To avoid repeated evaluations based upon the testing corpus, here our evaluation is based upon sentences of length ≤ 40 from the development corpus. We note here that this corpus is somewhat harder than the "official" test corpus. For example,

System	Precision	Recall
Old	86.3	86.1
Explicit Pre-Term	88.0	88.1
Marked Co-ordination	88.6	88.7
Standard Interpolation	88.2	88.3
MaxEnt Inspired	89.0	89.2
First-order Markov	88.6	87.4
Second-order Markov	89.5	89.3
Best	89.8	89.6

Figure 2: Labeled precision/recall for length ≤ 40, development corpus

the final version of our system achieves an average precision/recall of 90.1% on the test corpus but only an average precision/recall of 89.7% on the development corpus. This is indicated in Figure 2 where the model labeled "Best" has precision of 89.8% and recall of 89.6% for an average of 89.7%, 0.4% lower than the results on the official test corpus. This is in accord with our experience that development-corpus results are from 0.3% to 0.5% lower than those obtained on the test corpus. We believe this is due to a slight difference in average sentence length between the two corpora, with the test corpus having an average sentence length approximately one-half word less.

The model labeled "Old" attempts to recreate the Char97 system using the current program. It makes no use of special maximum-entropy-inspired features (though their presence made it much easier to perform these experiments), it does not guess the pre-terminal before guessing the lexical head, and it uses a tree-bank grammar rather than a Markov grammar. This parser achieves an average precision/recall of 86.2%. This is consistent with the average precision/recall of 86.6% for [5] mentioned above, as the latter was on the test corpus and the former on the development corpus.

Between the Old model and the Best model, Figure 2 gives precision/recall measurements for several different versions of our parser. One of the first and without doubt the most significant change we made in the current parser is to move from two stages of probabilistic decisions at each node to three. As already noted, Char97 first guesses the lexical head of a constituent and then, given the head, it guesses the PCFG rule used to expand the constituent in question. In contrast, the current parser first guesses the head's pre-terminal, then the head, and then the expansion. It turns out that usefulness of this process has already been discovered by Collins [10], who in turn notes (personal communication) that it was previously used by Eisner

[12]. However Collins in [10] does not stress the decision to guess the head's pre-terminal first, and it might be lost on the casual reader. Indeed, it was lost on the present author until he went back after the fact and found it there. In Figure 2 we show that this one factor improves performance by nearly 2%.

It may not be obvious why this should make so great a difference, since most words are effectively unambiguous. (For example, part-of-speech tagging using the most probable pre-terminal for each word is 90% accurate [8].) We believe that two factors contribute to this performance gain. The first is simply that if we first guess the pre-terminal, when we go to guess the head the first thing we can condition upon is the pre-terminal, i.e., we compute $p(h \mid t)$. This quantity is a relatively intuitive one (as, for example, it is the quantity used in a PCFG to relate words to their pre-terminals) and it seems particularly good to condition upon here since we use it, in effect, as the unsmoothed probability upon which all smoothing of $p(h)$ is based.

Contrast this with the situation if one does not first guess the pre-terminal. What can take its place as the initial unsmoothed probability? Char97 uses l, the label of the constituent c that introduces h as its head. Often this is a pre-terminal, but equally often it is some phrasal non-terminal. For example, in sentence "I bought a car," "car" is introduced as the head of an np while "bought" is introduced as the head of root, the label used to root a parse tree. It seems that $p(h \mid l)$ is not nearly as good a base probability for smoothing the probability of h as is $p(h \mid t)$. This one "fix" makes slightly over a percent difference in the results.

The second major reason why first guessing the pre-terminal makes so much difference is that it can be used when backing off the lexical head in computing the probably of the rule expansion. For example, when we first guess the lexical head we can move from computing $p(r \mid l, l_p, h)$ to $p(r \mid l, t, l_p, h)$. So, e.g., even if the word "conflating" does not appear in the training corpus (and it does not), the "ng" ending allows our program to guess with relative security that the word has the vbg pre-terminal, and thus the probability of various rule expansions can be considerable sharpened. For example, the tree-bank PCFG probability of the rule "vp $\rightarrow$ vbg np" is 0.0145, whereas once we condition on the fact that the lexical head is a vbg we get a probability of 0.214. If there is one thing that one should extract from this paper, it is probably the importance of guessing pre-terminals before guessing lexical heads.

The second modification is the explicit marking of noun and verb-phrase coordination. We have already noted the importance of conditioning on the parent label l_p. So, for example, information about an np is conditioned on

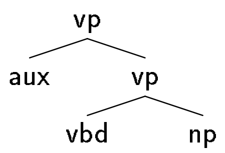

Figure 3: Verb phrase with both main and auxiliary verbs

the parent — e.g., an s, vp, pp, etc. Note that when an np is part of an np coordinate structure the parent will itself be an np, and similarly for a vp. But nps and vps can occur with np and vp parents in non-coordinate structures as well. For example, in the Penn Treebank a vp with both main and auxiliary verbs has the structure shown in Figure 3. Note that the subordinate vp has a vp parent.

Thus np and vp parents of constituents are marked to indicate if the parents are a coordinate structure. A vp coordinate structure is defined here as a constituent with two or more vp children, one or more of the constituents comma, cc, conjp (conjunctive phrase), and nothing else; coordinate np phrases are defined similarly. Something very much like this is done in [15]. As shown in Figure 2, conditioning on this information gives a 0.6% improvement. We believe that this is mostly due to improvements in guessing the sub-constituent's pre-terminal and head. Given we are already at the 88% level of accuracy, we judge a 0.6% improvement to be very much worth while.

Next we add the less obvious conditioning events noted in our previous discussion of the final model — grandparent label l_g and left sibling label l_b. When we do so using our maximum-entropy inspired conditioning, we get another 0.45% improvement in average precision/recall, as indicated in Figure 2 on the line labeled "MaxEnt-Inspired'. Note that we also tried including this information using a standard deleted-interpolation model. The results here are shown in the line "Standard Interpolation". Including this information within a standard deleted interpolation model causes a 0.6% decrease from the results using the less conventional model. Indeed, the resulting performance is worse than not using this information at all.

Up to this point all of the models considered in this section are tree-bank grammar models. That is, the PCFG grammar rules are read directly off the training corpus. As already noted, our best model uses a Markov-grammar approach. As one can see in Figure 2 a first-order Markov grammar (with all the aforementioned improvements) performs slightly worse than the equivalent tree-bank-grammar parser. However, a second-order grammar does slightly better and a third-order grammar does significantly better than the tree-bank parser.

6 Conclusion

We have presented a lexicalized Markov grammar parsing model that achieves (using the now standard training/testing/development sections of the Penn treebank) an average precision/recall of 91.1% on sentences of length ≤ 40 and 89.5% on sentences of length ≤ 100. This corresponds to an error reduction of 15% over the best previously published single parser results on this test set, those of Collins [10]. That the previous three best parsers on this test [5,10,17] all perform within a percentage point of each other, despite quite different basic mechanisms, led some researchers to wonder if there might be some maximum level of parsing performance that could be obtained using the treebank for training, and to conjecture that perhaps we were at it. The results reported here disprove this conjecture. The results of [13] achieved by combining the aforementioned three-best parsers also suggest that the limit on tree-bank trained parsers is much higher than previously thought. Indeed, it may be that substituting this new parser into the mix may yield still higher results.

From our perspective perhaps the two most important numbers to come out of this research are the overall error reduction of 15% over the results in [10] and the intermediate-result improvement of nearly 2% on labeled precision/recall due to the simple idea of guessing the head's pre-terminal before guessing the head. Neither of these results were anticipated at the start of this research.

As noted above, the main methodological innovation presented here is our "maximum-entropy-inspired" model for conditioning and smoothing. Two aspects of this model deserve some comment. The first is the slight, but important, improvement achieved by using this model over conventional deleted interpolation, as indicated in Figure 2. We expect that as we experiment with other, more semantic conditioning information, the importance of this aspect of the model will increase.

More important to our eyes, though, is the flexibility of the maximum-entropy-inspired model. Though in some respects not quite as flexible as true maximum entropy, it is much simpler and, in our estimation, has benefits when it comes to smoothing. Ultimately it is this flexibility that let us try the various conditioning events, to move on to a Markov grammar approach, and to try several Markov grammars of different orders, without significant programming. Indeed, we initiated this line of work in an attempt to create a parser that would be flexible enough to allow modifications for parsing down to more semantic levels of detail. It is to this project that our future parsing work will be devoted.

References

1. Berger, A. L., Pietra, S. A. D. and Pietra, V. J. D. A maximum entropy approach to natural language processing. *Computational Linguistics 22* 1 (1996), 39–71.
2. Caraballo, S. and Charniak, E. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics 24* (1998), 275–298.
3. Charniak, E. *Tree-bank grammars.* In *Proceedings of the Thirteenth National Conference on Artificial Intelligence.* AAAI Press/MIT Press, Menlo Park, 1996, 1031–1036.
4. Charniak, E. Expected-Frequency Interpolation. Department of Computer Science, Brown University, Technical Report CS96-37, 1996.
5. Charniak, E. *Statistical parsing with a context-free grammar and word statistics.* In *Proceedings of the Fourteenth National Conference on Artificial Intelligence.* AAAI Press/MIT Press, Menlo Park, 1997, 598–603.
6. Charniak, E. Statistical techniques for natural language parsing. *AI Magazine 18* 4 (1997), 33–43.
7. Charniak, E., Goldwater, S. and Johnson, M. *Edge-based best-first chart parsing.* In *Proceedings of the Sixth Workshop on Very Large Corpora.* 1998, 127–133.
8. Charniak, E., Hendrickson, C., Jacobson, N. and Perkowitz, M. *Equations for part-of-speech tagging.* In *Proceedings of the Eleventh National Conference on Artificial Intelligence.* AAAI Press/MIT Press, Menlo Park, 1993, 784–789.
9. Collins, M. J. *A new statistical parser based on bigram lexical dependencies.* In *Proceedings of the 34th Annual Meeting of the ACL.* 1996.
10. Collins, M. J. *Three generative lexicalised models for statistical parsing.* In *Proceedings of the 35th Annual Meeting of the ACL.* 1997, 16–23.
11. Darroch, J. N. and Ratcliff, D. Generalized iterative scaling for log-linear models. *Annals of Mathematical Statistics* 33 (1972), 1470–1480.

12. Eisner, J. M. An empirical comparison of probability models for dependency grammar. Institute for Research in Cognitive Science, University of Pennsylvania, Technical Report IRCS-96-11, 1996.

13. Henderson, J. C. and Brill, E. *Exploiting diversity in natural language processing: combining parsers*. In *1999 Joint Sigdat Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*. ACL, New Brunswick NJ, 1999, 187–194.

14. Johnson, M. PCFG models of linguistic tree representations. *Computational Linguistics 24* 4 (1998), 613–632.

15. Magerman, D. M. *Statistical decision-tree models for parsing*. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*. 1995, 276–283.

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

17. Ratnaparkhi, A. Learning to parse natural language with maximum entropy models. *Machine Learning 34*1/2/3 (1999), 151–176.