Statistical Parsing with a Context-free Grammar and Word Statistics *

Eugene Charniak
Department of Computer Science, Brown University
ec@cs.brown.edu

Abstract

We describe a parsing system based upon a language model for English that is, in turn, based upon assigning probabilities to possible parses for a sentence. This model is used in a parsing system by finding the parse for the sentence with the highest probability. This system outperforms previous schemes. As this is the third in a series of parsers by different authors that are similar enough to invite detailed comparisons but different enough to give rise to different levels of performance, we also report on some experiments designed to identify what aspects of these systems best explain their relative performance.

Introduction

We present a statistical parser that induces its grammar and probabilities from a hand-parsed corpus (a *tree-bank*). Parsers induced from corpora are of interest both as simply exercises in machine learning and also because they are often the best parsers obtainable by any method. That is, if one desires a parser that produces trees in the tree-bank style and that assigns some parse to all sentences thrown at it, then parsers induced from tree-bank data are currently the best.

Naturally there are also drawbacks. Creating the requisite training corpus, or tree-bank, is a Herculean task, so there are not many to choose from. (In this paper we use the Penn Wall Street Journal Treebank [6].) Thus the variety of parse types generated by such systems is limited.

At the same time, the dearth of training corpora has at least one positive effect. Several systems now exist to induce parsers from this data and it is possible to make detailed comparisons of these systems, secure in the knowledge that all of them were designed to start from the same data and accomplish the same task. Thus an unusually large portion of this paper is devoted to the comparison of our parser to previous work, in which we attempt to trace performance differences to particular decisions made in the construction of these parsing systems.

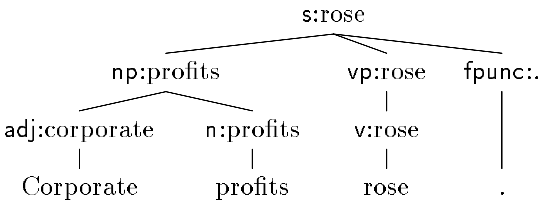

Figure 1: Parse of a simple sentence

The Probabilistic Model

The system we present here is probabilistic in that it returns the parse π of a sentence s that maximizes $p(\pi \mid s)$. More formally, we want our parser to return $\mathbf{P}(s)$ where

$$\mathbf{P}(s) \quad = \quad \arg\max_{\pi} \frac{p(\pi, s)}{p(s)} = \arg\max_{\pi} p(\pi, s) \quad (1)$$

Thus the parser operates by assigning probabilities $p(\pi, s)$ to the sentence s under all its possible parses π (or at least all the parses it constructs) and then choosing the parse for which $p(\pi, s)$ is highest.

To illustrate how our model assigns a probability to a sentence under a given parse, consider the sentence "Corporate profits rose." under the parse shown in Figure 1. We can think of a parse as a bag of context-free grammar rules specifying how each parse constituent is expanded. Indeed, this is exactly how our system considers it, since it uses a context-free grammar and finds a set of (we hope) high-probability parses for the sentence. In what follows we pretend that the probability model is applied separately to each possible parse. In actuality this is too inefficient; once the set of parses has been found their probabilities are determined in one bottom-up pass.

Returning to Figure 1, at each non-terminal node we note the type of node (e.g., a noun-phrase, np) and

*This research was supported in part by NSF grant IRI-9319516 and by ONR grant N0014-96-1-0549.

the *head* of the constituent (its most important lexical item). For example, the head of an np is the main noun, the head of a vp is the main verb, and the head of an s is the head of the sentence's vp. Formally, the head is assigned by a deterministic function of the grammar rule used to make up the constituent. Since heads of constituents are often specified as heads of sub-constituents (e.g., the head of the s is the head of the vp), heads are determined bottom up. Note that if a constituent can be created using several different rules, it may have several heads — but only one for any particular parse of the sentence. We are concerned with the constituent heads because of the common linguistic intuition that the forms of a constituent and its subconstituents are determined more by the constituent's head than any other of its lexical items.

Given the heads for each constituent, it is possible to determine the probability of all parses of a sentence in either a top-down or a bottom-up fashion. Bottom-up is more efficient and is used in the program. Top-down is more intuitive, and we use that method here.

Suppose that we have worked our way top down and are now about to determine the probability of a constituent, say the np "Corporate profits." This proceeds by first determining the probability of its head, then the probability of the form of the constituent given the head, and finally recursing to find the probabilities of sub-constituents. Consider the first of these — computing the probability of the head s given all the information previously established about the sentence. We assume that s is dependent only on its type t, the type of the parent constituent l, and the head of the parent constituent h. Thus we use $p(s \mid h, t, l)$. For the head "profits" of the np "Corporate profits" this would be: $p(\text{profits} \mid \text{rose}, \mathsf{np}, \mathsf{s})$. That is, we compute the probability that a np is headed by "profits" given that it is a np and that the constituent above it is an s headed by the lexical item "rose."

This is only an approximation of the true dependencies, but it is also already so specific a probability that we have no real chance of obtaining the data empirically. Thus we approximate $p(s \mid h, t, l)$ as follows:

$$\begin{aligned} p(s \mid h,t,l) \quad = \quad & \lambda_1(e)\hat{p}(s \mid h,t,l) \\ & +\lambda_2(e)\hat{p}(s \mid \mathbf{c}_h,t,l) \\ & +\lambda_3(e)\hat{p}(s \mid t,l) + \lambda_4(e)\hat{p}(s \mid t) \end{aligned} \tag{2}$$

Here and in what follows $\hat{p}$ denotes a distribution obtained empirically from the training data. Equation 2 can thus be characterized as a smoothing equation employing (to a first approximation) the deleted interpolation method for smoothing. Equation 2 differs from standard deleted interpolation in how the interpolation parameters $\lambda_i(e)$ are computed. The e here is an estimate, given the amount of training data used, of how often one would expect the particular concurrence of events, e.g., given the amount of training data used, how many times we should see "profits" as the head of an np under an s headed by "rose." Our method is described in [2] and is not discussed further here.

The other aspect of Equation 2 that is not standard deleted interpolation is the term $\hat{p}(s \mid \mathbf{c}_h, t, l)$. The idea here is to cluster the heads h according to how they behave in $\hat{p}(s \mid h, t, l)$ and then compute the probability of s based not on the head of the parent, h, but on h's cluster $\mathbf{c}_h$. We do not describe the clustering method here except to note that it uses a scheme something like that in [7].

To give some idea of how Equation 2 works in practice, we give here the values of the various empirical distributions used therein when estimating the probability of "profits" given "rose" $p(\text{prf} \mid \text{rose}, \mathsf{np}, \mathsf{s})$ and of "corporate" given "profits" $p(\text{crp} \mid \text{prf}, \mathsf{adj}, \mathsf{np})$.

	$p(\text{prf} \mid \text{rose}, \mathsf{np}, \mathsf{s})$	$p(\text{crp} \mid \text{prf}, \mathsf{adj}, \mathsf{np})$
$\hat{p}(s \mid h,t,l)$	0	0.2449
$\hat{p}(s \mid \mathbf{c}_h,t,l)$	0.00352223	0.0149821
$\hat{p}(s \mid t,l)$	0.0006274	0.00533
$\hat{p}(s \mid t)$	0.000556527	0.004179

For example, the probability of "profits" given only that it is the head of a np is .00056. If we add the conditioning information that it is under an s node (which almost always means the np is the subject of the sentence), the probability is slightly higher. If we add the fact that the main verb is "rose" the observed probability is zero, indicating that the training corpus did not have a sentence with "profits" as the subject of "rose." On the other hand, if we consider the cluster of verbs similar to "rose," "profits" was a reasonably common subject, with a relatively high probability of .0035. The various probabilities for "corporate" are even more orderly — as we add more conditioning information, the observed probability is always higher.

Now we turn to the second major probability in our model, the probability of the form of the constituent given its head, or more formally, the probability that a constituent c is expanded using the grammar rule r given that c is of type t, is headed by h, and has parent of type l, $p(r \mid h, t, l)$. We smooth this probability using deleted interpolation with the formula

$$\begin{aligned} p(r \mid h,t,l) \quad = \quad & \lambda_1(e)\hat{p}(r \mid h,t,l) \\ & +\lambda_2(e)\hat{p}(r \mid h,t) + \lambda_3(e)\hat{p}(r \mid \mathbf{c}_h,t) \\ & +\lambda_4(e)\hat{p}(r \mid t,l) + \lambda_5(e)\hat{p}(r \mid t) \end{aligned} \tag{3}$$

As an example of how this works in practice, consider the probability of the grammar rule np $\rightarrow$ adj plural-n (as used in the np "corporate profits") and how it varies depending on the conditioning events:

$\hat{p}(r \mid h,t,l)$	$\hat{p}(r \mid h,t)$	$\hat{p}(r \mid \mathbf{c}_h,t)$	$\hat{p}(r \mid t,l)$	$\hat{p}(r \mid t)$
0.1707	0.1875	0.1192	0.0176	0.0255

Because this is a relatively common example, we see that with two small exceptions the more precise the conditioning events, the higher the probability.

The Algorithm

We now consider in more detail how the probability model just described is turned into a parser.

Before parsing we train the parser using the pre-parsed training corpus. First we read a context-free grammar (a *tree-bank grammar*) off the corpus, as described in [3]. We then collect the statistics used to compute the empirically observed probability distributions needed for Equations 2 and 3.

We parse a new (test) sentence s by first obtaining a set of parses using relatively standard context-free chart-parsing technology. No attempt is made to find all possible parses for s. Rather, techniques described in [1] are used to select constituents that promise to contribute to the most probable parses, where parse probability is measured according to the simple probabilistic context-free grammar distribution $p(r \mid t)$. Because this is not the official distribution described by Equations 2 and 3, we cannot just find the most probable parse according to this distribution, but the scheme does allow us to ignore improbable parses. The resulting chart contains the constituents along with information on how they combine to form parses.

We next compute for each constituent in the chart the probability of the constituent given the full distributions of Equations 2 and 3.[1] The parser then pulls out the Viterbi parse (the parse with the overall highest probability) according to the full distribution as its choice for the parse of the sentence. In testing this is compared to the tree-bank parse as described in the next section.

In one set of tests we attempted to assess the utility of unsupervised training so we used the parser just outlined to parse about 30 million words of unparsed Wall Street Journal text. We treated the Viterbi parses returned by the parser as "correct" and collected statistical data from them. This data was combined with that obtained from the original parsed training data to create new versions of the empirical distributions used in Equations 2 and 3. This version also used class information about the attachment points of pps. The effect of this modification is small (about .1% average precision and recall) and discussion is omitted here.

Results

We trained our parser on sections 02-21 (about one million words) of the Penn Wall Street Journal Treebank and tested the parser on section 23 (50,000 words). Preliminary testing was done on section 24, to avoid repeated testing of section 23 with the risk of unconsciously fitting the model to that test sample. This arrangement was chosen because it is exactly what was used in [4] and [5]. The next section compares our results to theirs.

After training we parsed the testing corpus using five versions of our system. In each case the program pulled out the most probable parse according to the probability model under consideration. The models for which we tested the system are: **PCFG** (no statistics other than the probabilities associated with each probabilistic context-free rule $p(r \mid t)$), **Minimal** (adds $\hat{p}(r \mid h, t, l)$ to the probability mix), **No Classes** (uses all of the probabilities in Equations 2 and 3 except $\hat{p}(r \mid \mathbf{c}_h, t, l)$ and $\hat{p}(s \mid \mathbf{c}_h, t, l)$), **Basic** (uses Equations 2 and 3) and **Full** (the basic model plus statistics based on unsupervised learning on about 30 million words of Wall Street Journal text).

We give results according to seven figures of merit: **LR** (*labeled recall* — the number of correct non-terminal labeled constituents divided by the number of such constituents in the tree-bank version) **LR2** (LR, but using the slightly idiosyncratic definition of correctness used in [4]), **LP** (*labeled precision* — the number of correct non-terminal labeled constituents divided by the number of such constituents produced by the parser), **LP2** (LP, but using the definition of correctness from [4]), **CB** (the average number of cross-brackets per sentence), **0CB** (percentage of sentences with zero cross-brackets), and **2CB** (percentage of sentences with ≤ 2 cross-brackets).

A non-terminal labeled constituent produced by the parser is considered correct if there exists a constituent in the tree-bank version with (1) the same starting point, (2) the same ending point, and (3) the same label (e.g., vp). To allow better comparison to previous work, we also give results using the slightly different definition of correctness used by Collins and Magerman (see LP2 and LR2). This differs from the standard definition in that (a) the non-terminal labels advp and prt are considered the same and (b) mistakes in position that only put punctuation in the wrong constituent are not considered mistakes. Since anything that is correct according to the traditional measure is also correct according to this less obvious one, we would expect the LP2 and LR2 to be slightly higher than LP and LR. As in previous work, we give our results for all sentences of length ≤ 40 and also those of length ≤ 100.

The results are shown in Figure 2. In the next section we compare these results to those achieved by previous systems. For now we simply note a few points. First, most of this data is as one would have expected. Restricting consideration to sentences of length ≤ 40 improves performance, though since almost all the sentences are in this length category, the difference is not large. Second, as we give the system more information its performance improves. Third, the differences between the two labeled constituent precision measures (LP and LP2) and those for labeled constituent recall

[1] For efficiency we first reduce the number of constituents by computing $p(c \mid s)$ and removing from consideration any c for which this is less than .002. The equations for this are reasonably standard. Again, this is according to the distribution $p(r \mid t)$. The ability to do this is the reason we first compute a set of parses and only later apply the full probability model to them.

	LR	LR2	LP	LP2	CB	0CB	2CB
			$\leq$ 40 words (2245 sentences)				
PCFG	71.2	71.7	75.3	75.8	2.03	39.5	68.1
Minimal	82.9	83.4	83.6	84.1	1.40	53.2	79.0
No Cls	86.2	86.8	85.8	86.4	1.14	59.9	83.4
Basic	86.3	86.8	86.6	87.1	1.09	60.7	84.0
Full	86.9	87.5	86.8	87.4	1.00	62.1	86.1
			$\leq$ 100 words (2416 sentences)				
PCFG	70.1	70.6	74.3	74.8	2.37	37.2	64.5
Minimal	82.0	82.5	82.6	83.1	1.68	50.6	75.7
No Cls	85.4	86.0	84.9	85.5	1.37	57.2	80.6
Basic	85.5	86.0	85.6	86.2	1.32	57.8	81.1
Full	86.1	86.7	86.0	86.6	1.20	59.5	83.2

Figure 2: Results for several versions of the parsing model

(LR and LR2) are small and almost unvarying, always between .5 and .6%. Fourth, all of the performance measures tell pretty much the same story. That is, they all go up and down together and with only one or two exceptions they go up by the same relative amount.

One aspect of this data might not have been anticipated. It is clear that adding a little bit of information (the "minimal" system) improves performance quite a bit over a pure PCFG, and that all additions over and above are much less significant. For example, consider the sequence of values for (lp2 + lr2)/2:

PCFG	Minimal	No Classes	Basic	Full
73.75	83.75	86.6	86.95	87.45

We can see that grouping words into classes for purposes of smoothing adds relatively little (.35%), as do the more heroic methods such as unsupervised learning on 30 million words of text (.5%). This seems to suggest that if our goal is to get, say, 95% average labeled precision and recall, further incremental improvements on this basic scheme may not get us there.

Previous Work

While the quantity of work on English parsing is huge, two prior pieces of work are sufficiently close to that described here that it behooves us to concentrate on that work at the expense of all the rest. In particular, we designed our experiments to conform exactly to those performed on two previous statistical parsing systems that also used the Penn Wall Street Journal Treebank to train parsers, those of Magerman [5] and Collins [4]. Thus our training and testing data are exactly the same sets of sentences used in this previous work, as are the testing measurements. In this section we describe these earlier parsers, and then describe experiments designed to shed light on the performance differences among the three systems.

The three systems have much in common. In all cases the program starts with relatively little knowledge of English and gathers the statistics it needs from the training corpus. In each case the system has a predefined notion of the lexical head of a phrase and uses this information in its statistics. Furthermore, all three systems seem to restrict head information to two levels — i.e., they have statistics that take into consideration the head of a constituent and the head of its parent, but not the head of the grandparent. Finally, all three systems pick as the correct parse the parse with the highest probability according to the smoothed probability distribution they define.

Before discussing the differences among the three systems, let us first note their performance:

	LR2	LP2	CB	0 CB	2CB
		$\leq$ 40 words (2245 sentences)			
Magerman	84.6	84.9	1.26	56.6	81.4
Collins	85.8	86.3	1.14	59.9	83.6
Charniak	87.5	87.4	1.0	62.1	86.1
		$\leq$ 100 words (2416 sentences)			
Magerman	84.0	84.3	1.46	54.0	78.8
Collins	85.3	85.7	1.32	57.2	80.8
Charniak	86.7	86.6	1.20	59.5	83.2

It seems fair to say that no matter what measure one considers, the three systems are at roughly the same level of performance, though clearly later systems work better than earlier ones and the 18% error reduction of our system over Magerman's is not negligible.

We now consider the differences among the three systems, with particular emphasis on teasing out which of them might be responsible for the different levels of performance. The most obvious differences are: (1) What overall probability is calculated? (2) How is part-of-speech tagging done? (3) To what degree does the system use an explicit grammar? (4) What statistics are gathered? (5) How are the statistics smoothed? and (6) Was unsupervised training used? The results in Figure 2 show that unsupervised training accounts for about .4% of the the difference between the performance of our system and the other two. However, the results there clearly indicate that there must be other differences as well, since even without unsupervised training our system outperforms the earlier ones.

In the remainder of this section we look at the other differences. We argue that of all of them, those that have the most impact on performance are statistics and smoothing, with statistics being the most important and smoothing important only insofar as it affects the statistics gathered.

As noted above, the system described here computes $p(s, \pi)$. Both Magerman and Collins, however, compute $p(\pi \mid s)$. The statistic we compute is, of course, more general than that used by Magerman and Collins, in that given $p(s, \pi)$ one can easily compute $p(\pi \mid s)$, but not vice versa. The difference in statistic could be important if one were going to attach these parsing system to a speech- or character-recognition system and use the parser as a language model. Our statistical calculations could compute the overall probability of the sentence, the statistic computed by a language model, whereas the other two could not. On the other hand, as long as the only intended use is parsing, this difference should have no effect since in all cases one picks as the best parse that with the highest probability for the sentence.

If we turn to part-of-speech tagging, the differences are perhaps more apparent than real. Our system has no explicit tagging step. If a word could be more than one part of speech, the system considers all of them and the "correct" tag is simply the one that appears in the "correct" parse. Magerman has an explicit tagging step, but his system stores all possible taggings along with their probabilities and considers all of them when deciding on the best parse. Thus his system too defines the correct tag as the one used in the correct parse. Collins describes his system as having a distinct tagging phase producing a single tag that is used during the rest of the parse. This would be a real distinction. However, the version of his system that worked best (and produced the results reported above) gave up on this and instead moved to a scheme more like Magerman's, with an explicit tagging phase, but one in which all probabilities are kept and then integrated with all the other probabilities affecting the overall probability of the sentence. It is interesting to note that this increased his system's average precision/recall by .6%, suggesting that pretagging is a bad idea when dealing with parsers performing at this level of accuracy. At any rate, all three systems are effectively tagging in pretty much the same way, and none of the performance differences are likely to be the result of tagging.

The role of grammar is probably the most glaring difference among the three schemes. In this regard our system is the most traditional, in that it is the only one of the three with an explicit grammar. Magerman's system has a subcomponent that for any possible constituent in a parse computes the probability that this node (a) starts a new constituent, (b) ends a constituent, (c) is in the middle of a constituent, or (d) both starts and ends a (unary) constituent. This scheme could be thought of as, in effect, making up grammar rules on the fly, but this is approximate at best. Collins's scheme is even more radical. Included in his probability mix is the probability that a phrase headed by lexical item s with part of speech t is directly under a phrase headed by h with non-terminal label n. To get an idea of how far this statistic is from a grammar, observe that it contains nothing requiring constituents even to be continuous; Collins instead adds this requirement to the algorithm that searches for the best parse.

We suspect that our decision to use a formal grammar has both advantages and disadvantages, and that the net result is a wash. The advantages stem from the finer level of control available using a grammar. For example, Collins in discussing of future improvements notes the problem of valency — how particular words get used in particular syntactic constructions. A canonical example is how "give" can take both a direct and indirect object, as in "Sue gave the boy the pizza." Other verbs, like "put", or "eat," cannot. As best we can tell, neither Collins nor Magerman can represent such facts. Our system can because it has the two probabilities $\hat{p}(\mathsf{vp} \rightarrow \mathsf{verb\ np\ np} \mid \text{give}) = .25$ and $\hat{p}(\mathsf{vp} \rightarrow \mathsf{verb\ np\ np} \mid \text{put}) = 0$.

Balanced against this, however, is the comparative lack of coverage of the tree-bank grammar we use. The standard assumption about tree-bank grammars is that they lack coverage because many uncommon grammar rules are not encountered in the particular corpus used to create the grammar. As noted in [3], this problem is not as bad as people expect, and the tests therein showed that lack of coverage was not a significant problem. However, in [3] parsing is done using only tag sequence information, which, as the PCFG results in Figure 2 show, is a poor system. We estimate that lack of coverage due to the use of a tree-bank grammar lowers performance somewhere between .5% and 1% in both precision and recall. While this is not much in a program with 74% precision and recall, it looms much larger when the program's performance is 87.4% and only 1% better than its competitors. Since the use of a tree-bank grammar has both benefits and costs, we expect that overall it comes out neutral.

This leaves two important differences among the systems, the statistics used and smoothing. We combine these two because Magerman's system uses a particular kind of smoothing that has a significant effect on the statistics. Magerman's system does not use individual statistics like those combined in our Equations 2 and 3, but rather a decision-tree scheme for smoothing. For example, suppose you want to label a particular non-terminal node. Rather than directly computing the probability of a particular label given the exact local context, the data for which is inevitably quite sparse, his system finds which questions about the context give the most information about the decision and then fashions a decision tree around these questions. At the leaf nodes of the tree one then finds a probability distribution over the possible answers. In the case of a decision tree for labeling non-terminals, the leaf nodes would specify the probability of all possible labels given the set of questions and answers that lead to that leaf node. Note that each question in the decision tree is binary, and thus questions about individual words are recast as questions about classes of words. Naturally, the decision tree stops long before the questions completely define the context in order to get the required smoothing. Given the number of possible words in each context, it is plausible to assume that the decision-tree questions hardly ever define the words completely, but rather depend on classes of words.

Collins's system uses raw word statistics and something quite similar to deleted interpolation, much like our Equations 2 and 3. On the other hand, in direct opposition to Magerman, he does not use classes of words.

Thus Collins uses nothing like the terms $\hat{p}(s \mid \mathbf{c}_h, t, l)$ and $\hat{p}(r \mid \mathbf{c}_h, t, l)$ in our Equations 2 and 3 respectively. Also, Collins never conditions an attachment decision on a node above those being attached. Thus he has nothing corresponding to the probability $\hat{p}(r \mid h, t, l)$, where l is the the label of the node above that being expanded by r.

We have gone into this level of detail about the probabilities used by the three systems because we believe that these are the major source of the performance differences observed. To test this conjecture we performed an experiment to see how these differences might affect final performance.

As indicated in Equations 2 and 3, probabilities of rules and words are estimated by interpolating between various submodels, some based upon classes, others upon words. Given our belief that these probabilities are the major differences, we hypothesize that one could "simulate" the performance of the other two systems by modifying the equations in our system to better reflect the probability mix used in the other systems and then see how it performs.

Thus we created two probability combinations shown by listing the various empirical distributions used in Equations 2 and 3 and indicating whether a particular distribution is included or not in the Collins model (indicated by a yes/no in the SimCollins column) and the Magerman model (SimMagerman):

	SimCollins	SimMagerman
$\hat{p}(s \mid h, t, l)$	Yes	No
$\hat{p}(s \mid \mathbf{c}_h, t, l)$	No	Yes
$\hat{p}(s \mid t, l)$	Yes	No
$p(s \mid t)$	Yes	Yes
$\hat{p}(r \mid h, t, l)$	No	No
$\hat{p}(r \mid h, t)$	Yes	No
$\hat{p}(r \mid \mathbf{c}_h, t)$	No	Yes
$\hat{p}(r \mid t, l)$	Yes	Yes
$\hat{p}(r \mid t)$	Yes	Yes

The basic idea is that we removed all statistics based upon individual words in SimMagerman, while for SimCollins we removed the statistics based upon word classes, as well as $\hat{p}(r \mid h, t, l)$, which, as noted above, does not correspond to anything that Collins collects. So, for example, the table indicates that the Magerman model does not include $\hat{p}(r \mid h, t)$, the probability of a rule r given the specific head h and the non-terminal t that is being expanded (since this is a statistic conditioned upon a particular word).

The results of these experiments are:

	LR2	LP2	CB	0CB	2CB
Magerman	84.6	84.9	1.26	56.6	81.4
SimMagerman	84.0	84.9	1.32	54.4	80.2
Collins	85.8	86.3	1.14	59.9	83.6
SimCollins	86.0	86.1	1.20	58.1	81.9

The rows show Magerman's results, the results of our Magerman mix, Collins' results, and our Collins mix. So SimMagerman has labeled precision/recall of 84.9/84.0, while the real system had 84.9/84.6.

The correspondences are not bad and support to some degree our conjecture that the probability mix is the major determinant of performance in the three systems. They also suggest two other conclusions:

- All else equal, statistics on individual words outperform statistics based upon word classes, and this may be sufficient to account for the difference in performance between Collins and Magerman.
- When dealing with a training corpus of slightly under a million words of parsed text, it is worth collecting statistics on some more detailed configurations (e.g., $\hat{p}(r \mid h, t, l)$) as well as less detailed ones (in particular, statistics based upon word classes). These statistics probably account for the difference in performance between Collins's system and that described here.

Conclusion

We have presented a parser in which the grammar and probabilistic parameters are induced from a tree bank and have shown that its performance is superior to previous parsers in this area. We also described an experiment that suggests that its superiority stems mainly from unsupervised learning plus the more extensive collection of statistics it uses, both more and less detailed than those in previous systems.

References

1. Caraballo, S. and Charniak, E. *Figures of merit for best-first probabilistic chart parsing.* In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. 1996, 127–132.
2. Charniak, E. Expected-Frequency Interpolation. Department of Computer Science, Brown University, Technical Report CS96-37, 1996.
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. Collins, M. J. *A new statistical parser based on bigram lexical dependencies.* In *Proceedings of the 34th Annual Meeting of the ACL.* 1996.
5. 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.
6. 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.
7. Pereira, F., Tishby, N. and Lee, L. *Distributional clustering of English words.* In *Proceedings of the Association for Computational Linguistics.* ACL, 1993.