Taggers for Parsers

Eugene Charniak,* Glenn Carroll,*
John Adcock,** Anthony Cassandra,*
Yoshihiko Gotoh,** Jeremy Katz,*
Michael Littman,* and John McCann*

Department of Computer Science* and
Division of Engineering**
Brown University, Providence RI 02912

Correspondence should be addressed to Eugene Charniak, Department of Computer Science, Box 1910 Brown University, Providence RI 02912.

Abstract

We consider what tagging models are most appropriate as front ends for probabilistic context-free-grammar parsers. In particular, we ask if using a tagger that returns more than one tag, a "multiple tagger," improves parsing performance. Our conclusion is somewhat surprising: single-tag Markov-model taggers are quite adequate for the task. First of all, parsing accuracy, as measured by the correct assignment of parts of speech to words, does not increase significantly when parsers select the tags themselves. In addition, the work required to parse a sentence goes up with increasing tag ambiguity, though not as much as one might expect. Thus, for the moment, single taggers are the best taggers.

1 Introduction

Recent years have seen a spate of research on various techniques for "tagging" — assigning a part of speech (or "tag") to each word in a text [1,2,8,9,11,12, 13,15,16]. Consider the following example:

The	can	will	rust
article	modal-verb	**modal-verb**	noun
	noun	noun	**verb**
	verb	verb	

Under each word we give some of its possible parts of speech in order of frequency. The correct tag is given in bold font.

One justification for tagging research is that a tagger can serve as a front end to a parser: the tagger assigns the tags to the incoming words and thus the parser can work at the tag level, where parsers do best. This raises questions of how well different types of taggers work as front ends to parsers. Despite the abundance of work on taggers, these questions have yet to be addressed; it is still uncommon actually to read of a tagger used with a parser, and when one is so used there is no analysis of suitability.

This question becomes more important because of two strands within tagging research. While most taggers return a single best tag for each word (we call these "single taggers"), some work has been done on taggers that return a list of possible tags in those cases where a second (or even third) best choice might be close to the best according to the tagger's metric [3,13, 15] (we call these "multiple taggers"). One obvious reason to do this would be to let the parser make the final decision. For example, the section on multiple taggers in [15] starts by observing that

> even with a rather low error rate of 3.7%, there are cases in which the system returns the wrong tag, which can be fatal for a parsing system trying to deal with sentences averaging more than 20 words in length.(p. 366)

In this paper we address the question whether single or multiple taggers work best with current probabilistic context-free grammar (PCFG) parsers. We also consider the extreme "multiple-tagger" position of allowing the parser to select among all tags to which the word model (i.e., dictionary) assigns a non-zero probability. This is equivalent to having a grammar which has actual words as terminals.

2 The Taggers

All of the taggers used in our experiments are statistically based. (This is as opposed to, e.g., the transformationally based taggers of [2,3]. We have no reason to believe that anything said here hinges on this choice; we are simply more familiar with the statistical tagging technology.) Creating a statistical tagger first requires a tagged corpus — a text or set of texts in which every word has been assigned its correct tag by hand. The tagged corpus is then divided into two disjoint sets of sentences, a large set used for "training" — collecting the statistics needed by the tagger — and a smaller set for "testing" — determining how well the tagger can find the correct tag sequence.

We built three statistically based taggers. The first is a traditional single tagger. This kind of tagger returns the tag sequence $t_{1,n}$ maximizing $P(t_{1,n} \mid w_{1,n})$, where $w_{1,n}$ is a sequence of n words and $t_{1,n}$ are the corresponding n tags. To put this into words, for a sentence of length n the tagger tries to find the tag sequence $t_{1,n}$ that has the highest probability given the words of the sentence, $w_{1,n}$. (For those familiar with the statistical literature, it finds this sequence using the standard Markov-model Viterbi algorithm.)

Second, we built a tagger that computes $P(t_i \mid w_{1,n})$ for each tag. This differers from the first tagger in that the first finds a tag sequence for the entire sentence "all at once," while the second looks at each position in the sentence and computes the probability for each possible tag for that word. This kind of tagger is better if one wants to find multiple tags for a given word. For example, for the sentence given earlier, "The can will rust." the tagger computes the probability that "will" is a noun, that it is a modal, etc. Thus one knows not just the most probable part of speech, but also the second most probable, etc. We also know how great the difference is between the first choice and the second, the second and third, etc. So while the first tagger returns what it considers the best overall tag sequence, the second tagger can identify alternative tags at a position with tag probabilities close to the best. (More formally, it computes the tag probabilities using the standard Markov-model forward-backward algorithm (as in [15]) and returns all tags t_j such that $P(t_j \mid w_{1,n}) \geq \sigma P(t_b \mid w_{1,n})$, where t_b is the best (most probable) tag for that position and σ is a system parameter such that $0 \leq \sigma \leq 1$. The closer σ is to 0, the more tags are returned.)

The third tagger we created is an "all tagger" that simply returns all tags with non-zero probability for that word. E.g., for "will" it returns "modal-verb," "noun," etc.

All the taggers share the same probabilistic model, that is, the same way of computing the probabilities of tag sequences given the words of the sentence. The model is based upon the reasonably standard bigram tagging model:

$$\arg\max_{t_{1,n}} \prod_{i=1}^{n} P(w_i \mid t_i) P(t_i \mid t_{i-1}) \tag{1}$$

Here $\arg\max_{t_{1,n}}$ says to find the tag sequence $t_{1,n}$ that maximizes the quantity that follows. Within the product, for each tag t_i we compute the product $P(w_i \mid t_i)P(t_i \mid t_{i-1})$. The first of these terms, $P(w_i \mid t_i)$, is often called the "word model" in that it causes the tagger to prefer tags that are common for the word in question. The second term, $P(t_i \mid t_{i-1})$, is called the "tag-context model" as it tends to make the tagger prefer tags that are likely to come after the tag for the previous word. For more on such equations, see [7].

It is the responsibility of the training phase to collect these two kinds of probabilities. However, a common problem for statistical taggers is that the set of examples found in the training data is not exhaustive, so that in the test data the tagger encounters unforeseen situations. A typical case is when the tagger encounters a word it has not previously seen. In this case $P(w_i \mid t_i)$, the probability of the word given the tag, is zero for all possible tags and the tagger "blows up." The solution is to "smooth" the data collected in the training phase so that these situations have not zero probability, but rather some low probability, presumably based upon some kind of auxiliary evidence. The rest of this section describes how this is done in our tagger. It is included for completeness — nothing in the rest of the paper depends on it — and can be skipped without penalty.

We used a 300,000-word subset of the tagged Brown Corpus [10] for training. We assume that, even with this relative small training corpus, it is not necessary to smooth the estimated $P(t_i \mid t_{i-1})$. In these experiments we deal with 19 tags, so estimating $P(t_i \mid t_{i-1})$. requires finding only 361 ($= 19^2$) parameters. As noted above, however, it is crucial to smooth $P(w_i \mid t_i)$ to estimate this probability for words that do not appear in the training corpus. Our approach is to handle words that appear in the training corpus separately from words that do not. For the latter, we combine the probability that an unknown word has a particular tag with the probability that a word with a given tag ends with a certain pair of letters. The second probability approximates the information provided by a morphological analyzer: words that end in "ed" are likely to be verbs, "ly" adverbs, etc. (We thank L. Boggess for this suggestion.) More formally, let e_i denote the final two letters of w_i and let k_i indicate whether w_i has been seen in the training data, in which case $k_i = 1$, otherwise $k_i = 0$. Probabilities that are estimated from counts in our corpus are designated with a circumflex, as in $\hat{P}(x \mid y)$:

$$\begin{aligned} P(w_i \mid t_i) &= P(w_i, k_i \mid t_i) && (2)\\ &\cong \hat{P}(k_i \mid t_i) P(w_i \mid t_i, k_i) && (3)\\ &\cong \hat{P}(k_i \mid t_i) \cdot && (4)\\ &\quad \left[k_i \hat{P}(w_i \mid k_i = 1, t_i) + (1 - k_i)\, P(w_i \mid t_i, k_i = 0)\right] \end{aligned}$$

Equation 2 follows because once we know w_i we know k_i. In Equation 4 we consider the two k_i cases separately. If we know the w_i (if it was in the training corpus) then $k_i = 1$. In this case we use the information collected on the word from the training data. If we have not seen the word (this is the $(1 - k_i)$ case) we need to come up with the probability $P(w_i \mid t_i, k_i = 0)$. We consider this term next. (In this discussion we represent $k_i = 0$ as $\neg k_i$.)

$$\begin{aligned}
P(w_i \mid t_i, \neg k_i) &= P(w_i, e_i \mid t_i, \neg k_i) && (5)\\
&= P(e_i \mid t_i, \neg k_i) P(w_i \mid t_i, \neg k_i, e_i) && (6)\\
&\cong P(e_i \mid t_i) P(w_i \mid e_i) && (7)\\
&\propto P(e_i \mid t_i) && (8)\\
&\propto (1-\epsilon)\, \hat{P}(e_i \mid t_i) + \epsilon && (9)
\end{aligned}$$

In Equation 5 we can add e_i because it is determined by w_i. Equation 7 incorporates two independence assumptions that allow us to collect reasonable statistics:

$$\begin{aligned}
P(e_i \mid t_i, \neg k_i) &= P(e_i \mid t_i) && (10)\\
P(w_i \mid t_i, \neg k_i, e_i) &= P(w_i \mid e_i) && (11)
\end{aligned}$$

Equation 10 assumes that the probability of a given word ending is independent of whether the word is in our training corpus (reasonable enough). Equation 11 assumes that the probability of a word given its ending is independent of both (a) its absence from the training corpus, and (b) its part of speech. The second of these is not true, but seems unlikely to affect us much. Going back to Equation 8, we drop consideration of $P(w_i \mid e_i)$: we are looking for the tags that make our overall probability highest, and since the words and endings are the same for all tag sequences, $P(w_i \mid e_i)$ can be ignored. Finally, in Equation 9 we smooth the $\hat{P}(e_i \mid t_i)$ statistics so that if this word ending has never appeared at all, all tags are considered equally likely. (With 26 characters and 19 tags, $\hat{P}(e_i \mid t_i)$ involves 12844 ($= 26 \cdot 26 \cdot 19$) parameters. Because of our comparatively small training corpus this requires smoothing.)

3 The Parser

The parser used in this experiment is a relatively standard chart parser. Chart parsers take a grammar for a language and a string (called a "sentence") and, if the string is in the language, output a "chart" that encodes in compact form all of the parses for the string. The grammar is expressed as a set of probabilistic context-free rewrite rules. The parser works by matching the rules of the grammar against the constituents of the sentence (or constituents produced by the successful completion of other rules). In so doing it creates "edges," a data-structure that records the fact that the initial portion of some rule is matched by some sequence of constituents.

The input to the parser is not the actual words of the sentence, but rather the output of the tagger. Thus in the case of our single tagger, the parser's input is a sequence of tags, one corresponding to each of the words in the sentence. For a multiple tagger we have not a single tag corresponding to a word but rather a set of tags, namely the tags that the multiple tagger reported for that word. In the case in which we do not really use a tagger at all, the "all tagger," this set corresponds to all possible tags for the word. When the parser gets more than one possible tag, it constructs all parses with all possible combinations of the tags. That is to say, the parses do not agree on the tags for the words. For example, in the standard ambiguous sentence "Time flies like an arrow" the parse that comments on how time is fleeting will tag "time" as a noun, while the parse which commands to listener to measure how long flies take to do something will tag "time" as a verb. It is possible, however, for the parser to (in effect) pick a single preferred set of tags by choosing those used in the most probable parse. Thus if the "time is fleeting" parse is most probable, "time" will have a noun tag.

For the reader familiar with chart-parser technology, we note that our parser incorporates two reasonably common optimizations. First, it uses top-down filtering (an edge starting at location l is not produced unless there is some possible parse of the words $w_{1,l-1}$ that could use an edge of this sort starting at l). Second, several would-be edges are collapsed into one when they are identical except for their predictions of subsequent constituents. This significantly reduces the number of edges.

The grammar used with the parser is the product of some related work on PCFG induction [5]. It consists of about 3500 rewrite rules (brevity was sacrificed for ease of learning) with 19 tags. We expected it to outperform our tagger in its ability to tag ambiguous words, as its per-tag cross entropy is about .07 bits/tag lower than the tagger's (more on the significance of this later). To get some feeling for what such numbers mean, we used our PCFG to construct an artificial corpus consisting of tags only. For this corpus our PCFG should give the lowest bits/tag of any model and in fact it outperforms the single-tagger by .15 bits/tag. That our grammar (when tested on the real corpus) performs at about half this level we consider quite good.

One detail of the linkage between parser and tagger deserves further discussion. When we allow the tagger to report more than one tag to the probabilistic parser, what probabilities should be associated with the tags? The answer to this is fairly straightforward, though not completely intuitive. If the words $w_{1,n}$ are a sentence, we want the parser to find the most probable parse tree π for this sentence, as shown here:

$$\arg\max_{\pi} P(\pi \mid w_{1,n}) \tag{12}$$

This is equivalent to maximizing $P(\pi, w_{1,n})$. In a context-free grammar model this is in turn equivalent to maximizing the product of the probabilities of the rules used in the parse. If $\mathcal{P}(\pi)$ are the rules used in the parse π, then we want to find the π which maximizes the following quantity:

$$\prod_{r \in \mathcal{P}(\pi)} P(r) \tag{13}$$

Note that if we want this to produce $P(\pi, w_{1,n})$, then the rules in $\mathcal{P}(\pi)$ must have as their terminal symbols the actual words of the sentence, not just the parts of speech. Thus we can think of the rules of our grammar as being of two sorts, the phrase-marker rules, and a set of "lexical" rules of the form $t \rightarrow w$ (e.g., for the noun "banana" we would have $\mathsf{noun} \rightarrow banana$). Naturally, in the computation we need associated with each lexical rule its probability $P(t \rightarrow w)$. With a little thought one can show the following equality:

$$P(t \rightarrow w) = P(w \mid t) \tag{14}$$

See, for example, [4].

Now when we talk of attaching a parser to a tagger we are assuming that the parser has rules that break constituents down to the part-of-speech level, what we called the phrase-marker rules, but that the tagger replaces the lexical rules. Nevertheless, we still want to maximize the quantities in Equations 12 and 13. Thus the probability associated with the tags provided by the tagger should be $P(t \rightarrow w)$ to keep the computation the same. But according to Equation 14 this is just $P(w \mid t)$, which, as already seen in Equation 1, we need for our tagger anyway. Thus associated with each recommend tag the tagger also provides $P(w \mid t)$ for the word-tag combination.

Note in particular that this means we should not try to do anything "fancy" like returning with the tag the probability assigned to it by the tagger. That is, in the case of multiple taggers we compute $P(t_i \mid w_{1,n})$, and one might consider using this number. The above argument suggests that this is not the right thing to do. Note that this is somewhat analogous to using $P(t \mid w)$ rather than $P(w \mid t)$ in Equation 1; for some time there was confusion on this point in the tagging literature, but this leads to a suboptimal result (see [7])

4 The Performance Measures

We use five performance measures in this study, four of which are straightforward. First, we measure the parser's computational effort in terms of the average number of edges generated in the course of parsing a sentence. Theory says that edges should be linear in parser effort, and a quick check shows a very good straight-line fit between number of edges and parser time (with a coefficient of about 2000 edges/second for our Sun Sparc 10's). Second, we measure how many tags are handed to the parser by the tagger in terms of average number of tags per word. Our third and fourth measures are percentage of words and sentences parsed by the parser. We use both measures because word percentage is a more natural figure for a tagger, while sentence percentage is the more natural for a parser. When given just the correct tags (each assigned probability 1.0), the grammar we use is able to parse 99.5% of the words and 99.6% of the sentences.

The final performance measure is designed to capture correctness of parser behavior as we change how many tags the parser sees. We decided to use tagging accuracy as this measure. More specifically, for each sentence parsed we extract the most probable parse, and from that we obtain the parts of speech assigned to the words in that parse. We then count the percentage of these words that agree with the hand-assigned tags. Arguably this is not the best measure of parsing accuracy, since ultimately the measure of a parser's performance is its ability to find the correct phrase-marker for a sentence. However, to measure this one needs a source of agreed-upon parses for the sentences. While there are now tree-banks of some size [14], they of necessity make assumptions about grammatical formalism. As these assumptions do not fit the grammar we use, we cannot exploit these resources.

On the other hand, using tagging accuracy for our performance measure has some advantages. First, tagging is much less sensitive than parsing to

Tagger	Percent Sentences Parsed	Percent Words Tagged (Parsed)	% Tagged Words Tagged Correctly	Tags per Word	Average Edges per Sentence
Single Tagger	N/A	100%	95.9%	1.0	N/A
Single Tagger + Parser	99.2%	99.5%	95.9%	1.0	1246
All Tagger + Parser	100%	100%	96.1%	2.15	4988
Pure Parser	99.6%	99.5%	N/A	1.0	1231

Figure 1: Taggers and their effect on parsing

the grammatical formalism one adopts. Second, although it is perfectly possible that a parser could get the tags correct while completely botching the parses, one would nevertheless expect, barring some "conspiracy," that the two measures would go up hand in hand.

5 Results

Training and testing of taggers and the PCFG were done on a 307885 word subset of the tagged Brown Corpus [10]. This includes all sentences of length greater than 1 (i.e., sentences having a symbol other than the final punctuation mark) and less than 23 that do not include foreign words, titles, or certain symbols, most notably parentheses. Testing was done on a 9631 word reserved subset of this subset, training was done on the rest. The Brown Corpus tagset contains between 70 odd and 400 odd tags, depending on how one counts. We map these automatically into the 19 tags used by the tagger and parser.

The overall results of the experiment are summarized in Figure 1. We give results for our single tagger, for the single tagger combined with our parser (single tagger + parser), and for the parser when it receives all non-

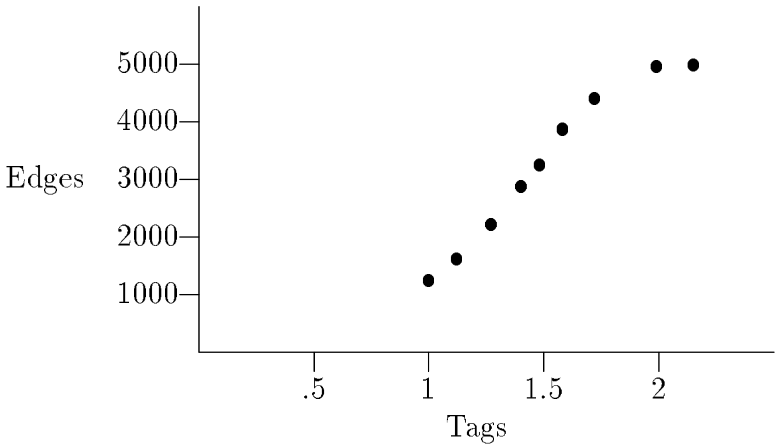

Figure 2: Average number of edges per sentence as a function of average number of tags per word

zero-probability tags for each word (all tagger + parser). Lastly we include data for the parser when it receives just the correct tags (pure parser) to provide a baseline for comparison. Some results for the multiple tagger are discussed later.

The percentage of sentences and words parsed is high. The percentage of words parsed (99.5%) with the single tagger is the same as that for the grammar when given the correct tags, the percentage of sentences parsed is slightly lower (99.2% vs. 99.6%). Thus the single tagger does not have a significant effect on the coverage of the parser. When the parser is given all non-zero-probability tags the percentage of sentences parsed goes up to 100%, but, as we shall see, one should not make too much of this. More importantly, the restriction of our parser measurements to those sentences that were successfully parsed should have no significant biasing effect, since the parsing percentage is so high.

The two rightmost columns of Figure 1 show that as the number of tags per word goes up the number of edges does as well. Figure 2 shows in more detail the relations between the number of tags and the number of edges produced by the parser (controlled by varying σ in the multiple tagger). Somewhat surprisingly, we do not see a blowup in parsing expense, even at the most permissive tagging rate. Our intuition was that an average of two tags per word would lead to an explosion in the number of parses, particularly for sentences of length 10 or more (a little more than half the corpus). In a technical report version of this paper [6] we show that adding a single tag could lead to a cubic increase in the number of edges. Figure 2 indicates that we are far from such a worst case. Part of this efficiency doubtless comes from the encoding of parses in the chart, allowing a few additional edges to represent a great many additional parses. In addition, our worst-case analysis requires a starting parse in which the edge count is linear in the length of the sentence; for our data, the edge count grows approximately as the square of sentence length.

The critical data in the table, however, are the percentage of words tagged correctly. Here we consider only the words that are actually tagged (i.e., unparsable sentences are ignored). We observe that when given all non-zero-probability tags, the parser is hardly more accurate than the tagger. The difference between 95.9 and 96.1 is at the ragged edge of statistical significance. We believe this difference is probably real, but the important question is whether anyone wants such a small improvement when it comes with a fourfold increase in work. Thus, from this data it does not seem that giving our parser extra tags is worth the effort. This is why we do not bother with the more detailed statistics for the multiple tagger with different σ's. This analysis also suggests that the increase in parsing percentage for the "all-tagger" case comes about not because the parser is correcting the tagger's mistakes but because, given enough tag options, the parser can manage to find at least one tag sequence that fits its grammatical rules.

6 Analysis

The result crying out for explanation is the parser's inability to tag words significantly better than the tagger. This section looks at this issue in detail. One possible explanation is that our grammar is simply a poor one: perhaps better grammars would get better results. In this section we argue that this is not the case. We hold that any PCFG offers only the most modest tag accuracy improvements over single taggers, and that these improvements are simply not worth their cost in extra parsing effort.

We start with a heuristic argument. As we have already noted, the idea that a parser should be able to tag better than a tagger stems from the assumption that the parser can predict the next tag better than a tagger.

That is, in standard statistical tagging models we look for:

$$\arg\max_{t_{1,n}} \prod_{i=1}^{n} P(w_i \mid t_{1,i}) P(t_i \mid t_{1,i-1}) \tag{15}$$

We called $P(w_i \mid t_{1,i})$ the "word model" and $P(t_i \mid t_{1,i-1})$ the "tag-context model." The idea is that as we plug in better tag-context models our tagging improves.

This is surely true, but *to what degree*? To suggest an answer to this question, note that there is an independent measure of the quality of tag-context models, their per-tag cross entropy. We do not go into detail here (see [4]) but simply note that, given a well-behaved corpus of n words, the per-tag cross entropy is well approximated by

$$-\frac{1}{n} \log P(t_{1,n}) \tag{16}$$

The lower the cross entropy the better the model. So let us look at how tagging improves as the cross entropy decreases.

A common reference point for tagging models is the tagger that simply assigns to each word its most common tag. We can recast this model in the form of Equation 15 as follows:

$$\begin{aligned} \arg\max_{t_{1,n}} \prod_{i=1}^{n} P(w_1 \mid t_{1,i}) P(t_i \mid t_{1,i-1}) &\equiv \arg\max_{t_{1,n}} \prod_{i=1}^{n} P(t_i \mid w_i) && (17) \\ &= \arg\max_{t_{1,n}} \prod_{i=1}^{n} \frac{P(w_i \mid t_i) P(t_i)}{P(w_i)} && (18) \\ &= \arg\max_{t_{1,n}} \prod_{i=1}^{n} P(w_i \mid t_i) P(t_i) && (19) \end{aligned}$$

(In Equation 18 we expand using Bayes' law, and then in Equation 19 ignore the denominator, $P(w_i)$, as it is constant for all $t_{1,n}$.) Thus for this tagger the tag-context model is simply the probability of the tag out of context. For our tag set, the cross entropy of this model is 3.61 bits/tag. It is common knowledge that such models give about 90% tagging accuracy. A result of 91.5% is given in [7] and that is the figure we use here.[1]

[1] This figure was for a large tag set, and thus it would probably be higher for our smaller tag set. Correcting for this would make our heuristic argument even stronger.

Next, consider a common tag-context model: $P(t_i \mid t_{1,i-1}) \equiv P(t_i \mid t_{i-1})$. Our version of this model gives a cross entropy of about 2.75 bits/tag and when combined with our word model achieves a tagging accuracy of 95.9%. Suppose, just for the sake of argument, that tagging accuracy is linear in cross-entropy improvement. (There is no justification for this assumption — it simply gives us some way of estimating what should happen as we change cross entropy.) With this assumption we see that we should expect an increased tagging accuracy of about 5.1% per bit (95.9 - 91.5)/(3.61 - 2.75). But, as noted earlier, our grammar is only .07 bits better than our single tagger. Thus we would expect it to perform .36% ($5.1 \cdot .07$) better than the single tagger. In fact, we saw about a .2% improvement but judged this unimportant.

We can carry this argument one step further. We remarked in passing that we had constructed an artificial corpus from our grammar and measured the per-tag cross entropy for both the grammar and the single tagger on it. As we noted, the grammar was .15 bits/tag better than the tagger. Thus, according to the linear improvement model, we would expect that this grammar would perform about .77% better than the tagger. That is, we would get a tagging accuracy of 96.7% rather than 95.9%. This is not a great increase, especially since we assume here a *perfect* grammar — the actual grammar that generated the corpus. In real life we never get anything this good. Thus this suggests that generally parsers will not do much better than taggers at assigning tags to the words.

Of course, this argument is based upon the assumption that there is a linear relation between accuracy and cross-entropy improvement, and thus is heuristic only. However, we can test it. We took the artificial corpus and assigned to each of its tags an actual word from the real corpus. For example, the first noun in the artificial corpus was assigned the first word in the real corpus that had a noun tag. (In the few cases where we needed more words of a given tag, we just "wrapped around.") The results, as shown in Figure 3, are very clear. The "all tagger" when applied to the artificial corpus achieved a tagging accuracy of 96.8%, almost exactly what the linear model predicted. However, note that this corpus seems to be about .2% easier to tag (the single tagger's accuracy went from 95.9% to 96.1%). Thus the improvement over the single tagger was .7% compared to the linear model's prediction of .8% (again, quite close). Figure 4 plots the tagging accuracy of our several models as a function of the cross-entropy of the tag-context

Tagger	Percent Sentences Parsed	Percent Words Tagged (Parsed)	% Tagged Words Tagged Correctly	Tags per Word	Number Edges per Sentence
Single Tagger	N/A	100%	96.1%	1.0	N/A
Single Tagger + Parser	99.4%	99.2%	96.1%	1.0	1221
All Tagger + Parser	99.9%	99.7%	96.8%	2.16	5664

Figure 3: Taggers and their effect on the artificial corpus

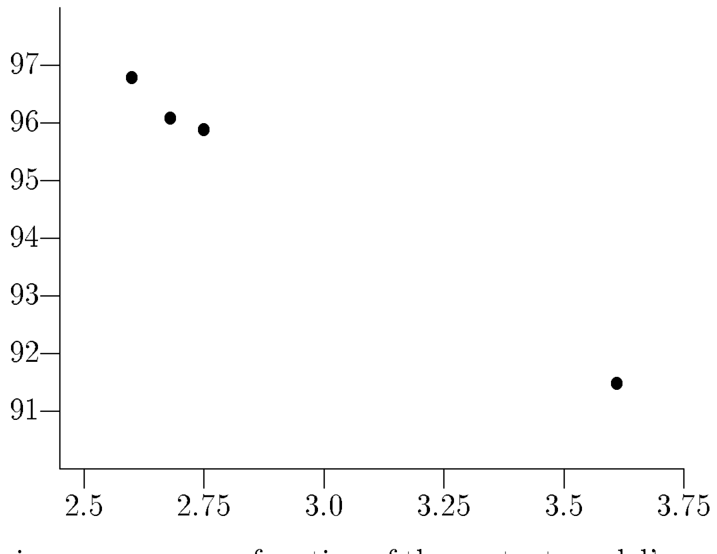

Figure 4: Tagging accuracy as a function of the context-model's cross entropy

model they use. While the straight-line fit is pretty good, we do not regard this as any confirmation of our linearity assumption. However, we do regard these results as confirming our overall hypothesis: PCFG parsers do not benefit from multiple taggers because they cannot tag much better than single taggers, and this is true because, when viewed as tagging context models, PCFGs provide only the most modest improvement over single taggers. Furthermore, this result applies not just to our grammar but to any grammar based on the traditional restriction to a small number of non-terminals (21 in our case).

We make this point about non-terminals because, if one relaxes this assumption and allows virtually unlimited numbers of non-terminals, then one can begin to include lexical information in the non-terminal symbols. That is, we get into the area of what we might call "probabilistic lexicalized grammars" (PLGs). We believe PLGs to be a promising area of research, but this is not the kind of PCFG parsers a tagger encounters today.

One final point. It could be argued that the inability of a parser to tag better than a single tagger is not of interest because the goal of parsing is producing a phrase marker and, as we have already noted, tagging accuracy is not necessarily a good indicator of parsing accuracy. This argument is not plausible. When we made the point about parsing vs. tagging accuracy, we said that getting the tags right did not necessarily mean getting the phrase marker right. But surely getting the tags wrong implies getting the phrase marker wrong.

7 Conclusion

We have contrasted two approaches to taggers for parsers: single taggers and multiple taggers. We suggest that, given a "normal" context free parsing system, i.e., one with a comparatively small number of non-terminals (20, say, not 2000), single taggers are preferable since the parser cannot do a significantly better job of tagging than current state-of-the-art single taggers. Furthermore, since increasing tag ambiguity increases the amount of work performed by the parser, there are good reasons *not* to pass on extra tags. This is not the result we expected upon commencing this research, but from the heuristic argument as to why this should be so and, in particular, from the simulation results in Figure 3, these results seem general.

However, one should not read too much into this result. First, it does not preclude further improvements in tagging. Work that looks for such improvements from collecting finer statistics based upon more lexical information still seems promising (e.g., [16]). Second, our result certainly does not imply that parsers are useless. One does not parse to get tags, one parses to find phrase markers. We may have ruled out multiple taggers as a route to improved parsing accuracy, but the need for parsers remains.

But this result *does* hint at another, quite interesting conclusion. Is there, in fact, a limit on how well standard syntactic parsers can do in the face of tag ambiguity? If a perfect grammar cannot get more than 96.8% of the tags correct, how well can it do on the phrase markers? More study will be required to make this more than a hint, but the question is an intriguing one.

References

1. Boggess, L., Agarwal, R. and Davis, R. *Disambiguation of prepositional phrases in automatically labeled technical text.* In *Proceedings of the Ninth National Conference on Artificial Intelligence*. AAAI Press/MIT Press, Menlo Park, 1991, 155–159.
2. Brill, E. *A simple rule-based part of speech tagger.* In *Proceedings of the Third Conference on Applied Natural Language Processing*. 1992.
3. Brill, E. *Some advances in transformation-based part of speech tagging.* In *Proceedings of the Twelfth National Conference on Artificial Intelligence*. AAAI Press/MIT Press, Menlo Park, 1994, 722–727.
4. Charniak, E. *Statistical Language Learning*. MIT Press, Cambridge, 1993.
5. Charniak, E. and Carroll, G. *Context-sensitive statistics for improved grammatical language models.* In *Proceedings of the Twelfth National Conference on Artificial Intelligence*. AAAI Press/MIT Press, Menlo Park, 1994.
6. Charniak, E., Carroll, G., Adcock, J., Cassandra, A., Gotoh, Y., Katz, J., Littman, M. and McCann, J. *Taggers for Parsers.* Department of Computer Science, Brown University, CS-94-06, 1994.

7. 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.

8. Church, K. W. *A stochastic parts program and noun phrase parser for unrestricted text.* In *Second Conference on Applied Natural Language Processing*. ACL, 1988, 136–143.

9. DeRose, S. J. Grammatical category disambiguation by statistical optimization. *Computational Linguistics 14* (1988), 31–39.

10. Francis, W. N. and Kučera, H. *Frequency Analysis of English Usage: Lexicon and Grammar*. Houghton Mifflin, Boston, 1982.

11. Jelinek, F. *Markov source modeling of text generation.* In *The Impact of Processing Techniques on Communications*, J. K. Skwirzinski, Ed. Nijhoff, Dordrecht, 1985.

12. Kupiec, J. and Maxwell, J. *Training stochastic grammars from unlabeled text corpora.* In *Workshop Notes, Statistically-Based NLP Techniques*. AAAI, 1992, 14–19.

13. deMarcken, C. G. *Parsing the LOB corpus.* In *Proceedings of the 1990 Conference of the Association for Computational Linguistics*. 1990, 243–251.

14. 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.

15. Weischedel, R., Meteer, M., Schwartz, R., Ramshaw, L. and Palmucci, J. Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics 19* (1993), 359–382.

16. Zernik, U. *Shipping departments vs. shipping pacemakers: using thematic analysis to improve tagging accuracy.* In *Proceedings of the Tenth National Conference on Artificial Intelligence*. 1992, 335–342.