Statistical Techniques for Natural Language Parsing *

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

August 7, 1997

Abstract

We review current statistical work on syntactic parsing and then consider part-of-speech tagging, which was the first syntactic problem to be successfully attacked by statistical techniques and also serves as a good warmup for the main topic, statistical parsing. Here we consider both the simplified case in which the input string is viewed as a string of parts of speech, and the more interesting case in which the parser is guided by statistical information about the particular words in the sentence. Finally we anticipate future research directions.

1 Introduction

Syntactic parsing is the process of assigning a "phrase marker" to a sentence — that is, the process that given a sentence like "The dog ate," produces a structure like that in Figure 1. In this example we adopt the standard abbreviations: `np` for "noun phrase," `vp` for "verb phrase," and `det` for "determiner."

It is generally accepted that finding the sort of structure shown in Figure 1 is useful in determining the meaning of a sentence. Consider a sentence like "Salespeople sold the dog biscuits." Figure 2 shows two structures for this sentence. Note that the two have different meanings: on the left the salespeople are selling dog biscuits, while on the right they are selling biscuits to dogs. Thus finding the correct parse corresponds to determining the correct meaning.

Figure 2 also exemplifies a major problem in parsing, "syntactic ambiguity" — sentences with two or more parses. In such cases it is necessary for the parser (or the understanding system in which the parser is embedded) to choose the correct one among the possible parses.

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

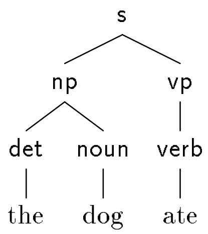

Figure 1: A simple parse

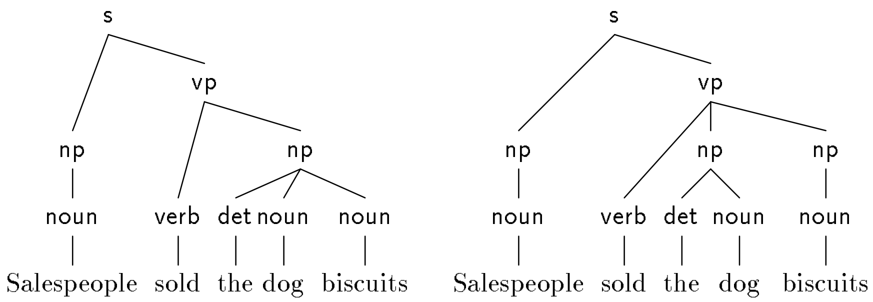

Figure 2: Two structures for an ambiguous sentence

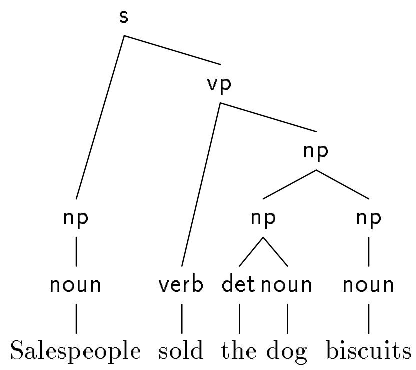

Figure 3: A third structure for an ambiguous sentence

This example is, however, misleading in a fundamental respect: it implies that we can assign at least a semi-plausible meaning to all of the possible parses. For most grammars (certainly for the ones statistical parsers typically deal with), this is not the case. Such grammars would assign dozens, possibly hundreds of parses to this sentence, ranging from the reasonable to the uninterpretable, with the majority at the uninterpretable end of things. To take but one example, a grammar I have been using has the rule

$$\mathsf{np} \rightarrow \mathsf{np}\ \mathsf{np}$$

This would be used in the analysis of a noun phrase like "10 dollars a share" where the two nps "10 dollars" and "a share" are part of the same np. The point here is that this rule would allow the third parse of the sentence shown in Figure 3, and this parse has no obvious meaning associated with it — the best I can do is an interpretation in which "biscuits" is the name of the dog. In fact, most of the parses that wide-coverage grammars find are like this one — pretty senseless.

A usually unstated, but widely accepted, assumption in the non-statistical community has it that some comparatively small set of parses for a sentence are legitimate ambiguities and that these parses have interpretations associated with them, albeit pretty silly ones sometimes. Furthermore, it is assumed that deciding between the "legitimate" parses is the responsibility not of the parser, but rather of some syntactic disambiguation unit working either in parallel with the parser or as a post-parsing process. Thus our hypothetical non-statistical traditionalist might say that the parser must rule out the structure in Figure 3, but would be within its rights to remain undecided between those in Figure 2.

By contrast, statistical parsing researchers assume that there is a continuum and that the only distinction to be drawn is between the correct parse and all the rest. The fact that we were able to find *some* interpretation for the parse in Figure 3 supports this "continuum" view. To put it another way, in this view of the problem there is no difference between parsing on one hand and syntactic disambiguation on the other: it's parsing all the way down.

2 Part-of-speech Tagging

The view of disambiguation as inseparable from parsing is well illustrated by the first natural-language processing task to receive a thoroughgoing statistical treatment — part-of-speech tagging (henceforth just "tagging"). A tagger assigns to each word in a sentence the part of speech that it assumes in the sentence. Consider the following example:

The	can	will	rust
det	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. Typically, for English there will be somewhere between 30 to 150 different parts of speech, depending on the tagging scheme. While most English words have only one possible part of speech (and thus it is impossible to get them wrong) many words have multiple possible parts of speech and it is the responsibility of a tagger to choose the correct one for the sentence at hand.

Suppose you have a 300,000-word training corpus in which all the words are already marked with their parts of speech. (At the end of this section we consider the case when no corpus is available.) You can parlay this corpus into a tagger that achieves 90% accuracy using a very simple algorithm. Record for each word its most common part of speech in the training corpus. To tag a new text, simply assign each word its most common tag. For words that do not appear in the training corpus, guess **proper-noun**. (While 90% may sound high, it is worth remembering that if we restricted consideration to words that have tag ambiguity, the accuracy figures would be much lower.)

Let us now express our algorithm in more mathematical terms, not so much to illuminate the algorithm as to introduce some mathematical notation. We ignore for the moment the possibility of seeing a new word. Let t vary over all possible tags. Then the most common tag for the ith word of a sentence, w_i, is the one that maximizes the probability $p(t \mid w_i)$. Or, to put it another way, this algorithm solves the tagging problem for a single word by finding

$$\arg\max_t p(t \mid w_i) \tag{1}$$

Here $\arg\max_t$ says "find the t that maximizes the following quantity," in this case the probability of a tag given the word. We could extend this to an entire text by looking for the sequence of tags that maximize the product of the individual word probabilities:

$$\arg\max_{t_{1,n}} \prod_{i=1}^{n} p(t_i \mid w_i) \tag{2}$$

Here we are looking for the sequence of n tags $t_{1,n}$ that maximizes the probabilities.

As we said, this simple algorithm achieves 90% accuracy; while this is not bad, it is not too hard to get up to 96% and the best taggers are now creeping toward 97%.[1] The basic problem with this algorithm is that it completely ignores a word's context, so that in "the can will rust" the word "can" is tagged as a modal rather than a noun, even though it follows the word "the."

To allow a bit of context, suppose we collect statistics on the probability of tag t_i following tag t_{i-1}. Now consider a tagger that follows the equation

$$\arg\max_{t_{1,n}} \prod_{i} p(t_i \mid t_{i-1}) p(w_i \mid t_i) \tag{3}$$

As before, we are taking the product over the probabilities for each word, but whereas before we considered the probability of each word out of context, here we use two probabilities — the first, $p(t_i \mid t_{i-1})$, is the probability of a tag (t_i) given the previous tag (t_{i-1}) as "context" and the second, $p(w_i \mid t_i)$, relates the word to its possible tags. This second probability, the probability of a word given a possible tag, may look odd, but it is correct. Many would assume that we would want $p(t_i \mid w_i)$, the probability of the tag given the word, but if we were to derive Equation 3 from first principles we would see that the less intuitive version shown here is right. It is also the case that if you try the system with both equations, Equation 3 outperforms the seemingly more intuitive one by about 1% in accuracy. I note this because of the moral: a bit of mathematical care can improve program performance — not merely impress journal referees.

Expressions like Equation 3 correspond to a well understood mathematical construct, hidden Markov models (HMMs) (Levinson, Rabiner & Sondhi [1983]). Basically, an HMM is a finite automaton in which the state transitions have probabilities and whose output is also probabilistic. For the tagging problem as defined in Equation 3, there is one state for each part of speech, and the output of the machine is the words of the sentence. Thus the probability of going from one state to another is given by the $p(t_i \mid t_{i-1})$ and for state t_i, the probabilistic output of the machine is governed by $p(w_i \mid t_i)$.

Figure 4 shows a fragment of an HMM for tagging. We see the state for det, and from it transitions to adj and noun with relatively high probabilities (.218 and .475 respectively) and a transition returning to det with fairly low probability (.0016), the latter reflecting the fact that two determiners in a row are unusual in English. We also see some possible outputs of each state, along with their probabilities.

[1] Human taggers are consistent with one another at the 98% level, which gives an upper bound on the sort of performance one can expect.

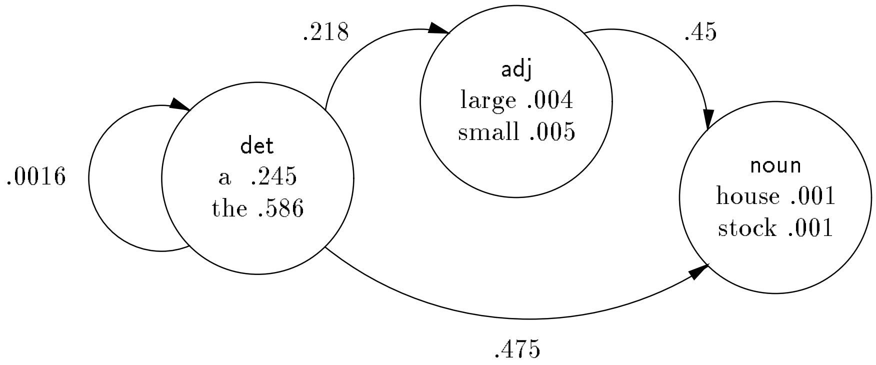

Figure 4: A fragment of an HMM for tagging

From this point of view the tagging problem is simply this: given a string of words, find the sequence of states the machine went through in order to output the sentence at hand. The HMM is "hidden" in the sense that the machine could have gone through many possible states sequences to produce the output, and thus we want to find the set of states with the highest probability. Again, this is exactly what is required in Equation 3.

The important point here is that there are many well understood algorithms for dealing with HMMs. We note two of them. First, there is a simple algorithm for solving Equation 3 in linear time (the Viterbi algorithm), even though the number of possible tag sequences to be evaluated goes up exponentially in the length of the text. Second, there is an algorithm (the forward-backward algorithm) for adjusting the probabilities on the states and outputs to better reflect the observed data.

It is instructive to consider what happens in this model when we allow unknown words in the input. For such words $p(w_i \mid t_i)$ is zero for all possible tags, which is not good. This is a special case of the "sparse-data problem" — what to do when there is not enough training data to cover all possible cases. As a practical issue, simply assigning unknown words some very low probability for any possible tag lets the tagger at least process the sentence. Better is to base one's statistics on less detailed information. For example, word endings in English give part-of-speech clues, e.g., words ending in "-ing" are typically progressive verbs. One can collect statistics on, say, the last two letters of the word and use them. More generally this process is called "smoothing" and it is a major research problem in its own right.

The tagger we just outlined is close to what might be called the canonical statistical tagger (Church [1988]; Weischedel et al. [1993]). There are, however, many different ways of using statistical information to make part-of-speech tagging decisions. Let us consider a second such method, "transformational tagging". It will loosen the grip of this first example on our imagination, and it also has interesting properties of its own. (An analogous transformational approach to parsing (Brill [1993]) is not covered here.)

The transformational scheme takes as its starting point the observation that a simple method such as choosing the most common tag for each word does very well. It then proposes and evaluates rules for altering the tags to others we hope are more accurate. Rule formats are kept very simple to make them easy to learn. For example, in (Brill [1995a]) one of the formats is

> Change the tag of a word from X to Y if the tag of the previous word is Z

We already noted that in "the can will rust" the trivial algorithm chooses the modal meaning of "can." A rule in the above format that would fix this problem is "change modal-verb to noun after det." More generally, if we have, say, 40 possible parts of speech (a common number) then this rule format would correspond to $40^3 = 6.4 \cdot 10^4$ possible rules, if we substitute each possible part of speech for X, Y, and Z. (In fact, all possible rules need not be created.)

The system then uses the training data as follows. It measures the initial accuracy (about 90%, as noted above) and then tries each possible rule on the training data, measuring the accuracy that would be achieved by applying the rule across the board. Some rules make things worse, but many make things better, and we pick the rule that makes the accuracy the highest. Call this rule 1.

Next we ask, given we have already applied rule 1, which of the other rules does the best job of correcting the remaining mistakes. This is rule 2. We keep doing this until the rules have little effect. The result is an ordered list of rules, typically numbering 200 or so, to apply to new examples we want tagged.

Besides illustrating the diversity of statistical parsers, transformation tagging has several nice properties that have not been duplicated in the conventional version. One is speed. Roche and Schabes (Roche & Schabes [1995]) report a method for turning the transformational tagger's rule list into a finite automaton and compiling the automaton into very efficient code. This is able to tag at the rate of about 11,000 words/second — effectively the speed at which the program is able to look up words in its dictionary. This contrasts with 1200 words/second for their implementation of the standard HMM tagger. (And, from my experience, 1200 words/second is a very fast implementation indeed of an HMM tagger.)

Another way in which the transformational version seems superior has to do with the assumption we made at the outset, that we have at our disposal a 300,000-word hand-tagged corpus. Is it possible to do without such a corpus? The current answer to this question is confusing. As already noted, the standard tagger uses HMM technology, and there are standard techniques for "training" HMMs — that is, for adjusting their parameters to fit the data better even when the data is not marked with the answers (the tags). Unfortunately, (Merialdo [1994]) reports that HMM training does not seem to improve HMM taggers. For example, suppose you start out with a dictionary for the language, but no statistics. Thus you know the possible parts of speech for the words, but know neither their relative frequency nor probabilities such as $p(t_i \mid t_{i-1})$. Starting with this information, HMM training gets little higher than the 90% achieved by the trivial algorithm. On the other hand, (Brill [1995b]) reports on a learning mechanism for a transformational tagger using untagged data that achieves a 95-96% level of performance. At first glance the algorithm looks a lot like the traditional HMM training algorithm adapted to a transformational milieu. Why it works, while training the statistic version with standard HMM training does not, is currently unexplained.

3 Statistical Parsing

We started our discussion of statistical taggers by assuming we had a corpus of hand-tagged text. For our statistical parsing work we assume that we have a corpus of hand-parsed text. Fortunately, for English there are such corpora, most notably the Penn tree-bank (Marcus, Santorini & Marcinkiewicz [1993]). In this article we concentrate on statistical parsers that use such corpora in order to produces parses mimicking the tree-bank style. While some work has used tree-banks to learn a grammar that is reasonably different from that used in the corpus (Briscoe & Waegner [1993]; Pereira & Schabes [1992]), this has proved more difficult and less successful than the tree-bank-mimicking variety.

Deciding to parse in the tree-bank style obviously makes testing a program's behavior easier — just compare the program output to the reserved testing portion from the tree-bank. However, we still need to define precisely how to measure accuracy. In this article we concentrate on two such measures, labeled precision and recall. Precision is the number of correct constituents found by the parser (summed over all the sentences in the test set) divided by the total number of non-terminal constituents the parser postulated. Recall is the number correct divided by the number found in the tree-bank version. (We consider the parts of speech to be the terminal symbols in this counting and thus ignore them. Otherwise we would be conflating parsing accuracy with part-of-speech tagging accuracy.) A constituent is considered correct if it starts in the right place, ends in the right place, and is labeled with the correct non-terminal. For example, suppose our tree-bank has the following parse:

```
(s (np (det The) (noun stranger))
   (vp (verb ate)
       (np (det the) (noun doughnut))
       (pp (prep with) (np (det a) (noun fork)))))
```

and our parser instead comes up with

```
(s (np (det The) (noun stranger))
   (vp (verb ate)
       (np (det the) (noun doughnut)
           (pp (prep with) (np (det a) (noun fork))))))
```

The parser has postulated six non-terminals (one s, three nps, one vp, and one pp), of which all are correct except the np headed by "doughnut," which should have ended after "doughnut" but instead ends after "fork." Thus the precision is 5/6, or .83. Since the tree-bank version also has six non-terminals, the recall is also .83.

To give some idea of how good current parsers are according to these measures, if we give a parser just the parts of speech and ask it to parse the sentence (i.e., if it does not see the actual words in the sentence), a good statistical parser that does not try to do anything too fancy can achieve about 75% average precision/recall. On the other hand, a state-of-the-art parser that has access to the actual words and makes use of fine-grained statistics on how particular English words fit into parses achieves labeled precision/recall rates of about 87-88%. These figures are for the Penn Wall Street Journal corpus mentioned earlier. This corpus contains the text of many articles taken from the Wall Street Journal without modification, except for the separation of punctuation marks from the words the adjoin. The average sentence length is 23 words and punctuation. I have not measured how many parses there typically are for these sentence, and, of course, it would depend on the grammar. But I would guess that for the kinds of grammars we discuss in the following pages, a million parses per sentence would be conservative. At any rate, we are not talking about toy examples.

Statistical parsers work by assigning probabilities to possible parses of a sentence, locating the most probable parse, and then presenting that parse as the answer. Thus to construct a statistical parser one must figure out how to (a) find possible parses, (b) assign probabilities to them, and (c) pull out the most probable one.

One of the simplest mechanisms for this is based upon "probabilistic context-free grammars" or PCFGs — context-free grammars in which every rule is assigned a probability (Charniak [1993]). The probabilities are to be interpreted as the probability of expanding a constituent, say an np, using this particular rule, as opposed to any of the other rules that could be used to expand this kind of constituent. For example, this toy PCFG generates the three parses for our ambiguous sentence from Figures 2 and 3:

s	→	np vp	(1.0)	np	→	det noun	(0.5)
vp	→	verb np	(0.8)	np	→	noun	(0.3)
vp	→	verb np np	(0.2)	np	→	det noun noun	(0.15)
				np	→	np np	(0.05)

Note, for example, how the probabilities of all of the rules for np sum to one.

Given the probability of individual rules, we calculate the probability of an entire parse by taking the product of the probabilities for each of the rules used therein. That is, if s is the entire sentence, π is a particular parse of s, c ranges over the constituents of π, and $r(c)$ is the rule used to expand c, then

$$p(s,\pi) = \prod_c p(r(c)) \tag{4}$$

In discussing the parses for "The salespeople sold the dog biscuits", we were particularly concerned about the nonsensical parse that considered "the dog biscuits" a single noun phrase containing two completely distinct noun phrases inside. Note that if, as in the above grammar, the rule np $\rightarrow$ np np has a fairly low probability, this parse would be ranked very low. According to our PCFG fragment, the leftmost parse of Figure 2 has probability $1.0 \cdot .3 \cdot .8 \cdot .15 = .036$, whereas that of Figure 3 has probability .0018.

PCFGs have many virtues. First and foremost, they are the obvious extension in the statistical domain of the ubiquitous context-free grammars with which most computer scientists and linguists are already familiar. Second, the parsing algorithms used for context-free parsing carry over to PCFGs (in particular, all possible parses can be found in n^3 time where n is the length of the sentence). Also, given the standard compact representation of CFG parses, the most probable parse can be found in n^3 time as well, so PCFGs have the same time complexity as their non-probabilistic brethren. There are more complicated but potentially useful algorithms as well (Goodman [1996a]; Jelinek & Lafferty [991]; Stolcke [1995]). Nevertheless, as we see in the following sections, PCFGs by themselves do not make particularly good statistical parsers, and many researchers do not use them.

4 Obtaining a Grammar

Suppose that we do want to use a PCFG as our statistical parsing mechanism and that, as we said earlier, we have a tree-bank. In order to parse a new sentence we need the following:

1. an actual grammar in PCFG form such that new (presumably novel) sentences have parses according to the grammar

2. a parser that applies the PCFG to a sentence and finds some or all of the possible parses for the sentence

3. the ability to find the parses with the highest probability according to Equation 4.

As noted at the end of the last section, there are well understood and reasonably efficient algorithms for (2) and (3), so these are not pressing problems. This leaves (1).

In addition, there is a trivial way to solve (1). Suppose we are not concerned about parsing novel sentences, and only want to make sure that we produce a grammar capable of assigning one or more parses to all of the training data. This is easy. All we need do is read in the parses and record the necessary rules. For example, suppose the leftmost parse in Figure 2 is in our tree-bank. Since this parse has an s node with np and vp immediately below it, our grammar must therefore have the rule s $\rightarrow$ np vp. Similarly, since there are three noun phrases, two consisting of a det followed by a noun and one with a det noun pp, our grammar would need the rules np $\rightarrow$ det noun and np $\rightarrow$ det noun pp.

It is possible to read off all the necessary rules in this fashion. Furthermore, we can assign probabilities to the rules by counting how often each rule is used. For example, if the rule np $\rightarrow$ det noun is used, say, 1,000 times, and overall np rules are used 60,000 times, then we assign this rule the probability 1,000/60,000 = .017.

"Tree-bank grammars" of this sort are evaluated in (Charniak [1996]) and a generalization of them is used in (Bod [1993]). They seem to be reasonably effective. That is to say, grammars of this sort achieve an average of about 75% average labeled precision and recall, the figure mentioned above as pretty good for grammars that only look at the parts of speech for the words.

In some respects this result is surprising. It was widely assumed that reading off a grammar in this fashion would not lead to very accurate parsing. The number of possible grammatical constructions is very large, and even the Penn tree-bank, with its nearly 50,000 hand-parsed sentences, is not large enough to contain them all, especially since the tree-bank style is relatively "flat"(constituents often contain little substructure). Thus one would expect new sentences to require rules not in the derived grammar. But while this indeed happens, first, the rules not in the tree-bank are *so* rare that missing them has very little effect on parsing, and second, when the tree-bank grammar is missing a rule used in the correct parse, the ambiguity we talked about earlier ensures that there will be lots of incorrect parses that are just a little off. Thus the missing rules have very little effect on statistical measures like average precision/recall.

To get some idea of how these grammars perform, we give a real example in Figure 5 cleaned up only by removing all of the parts of speech from the words in order to concentrate our attention on the non-terminals. The parser gets things right except for the area around the phrase "due out tomorrow", which seems to confuse it, and this leads to the very unintuitive ADJP (adjectival phrase) "the August merchandise trade deficit due." In terms of precision/recall, the parse got seven constituents correct of the nine it proposed (precision = 7/9) and of the 10 found in the tree-bank version (recall = 7/10).

Nevertheless, it is troubling that a tree-bank grammar is doomed from the start to misparse certain sentences simply because it does not have the right rule. An alternative that has been used in two state-of-the-art statistical parsers (Collins [1996]; Magerman [1995]) are "Markov grammars." (This is a term we have made up; neither (Magerman [1995]) nor (Collins [1996]) use it.)

Rather than storing explicit rules, a Markov grammar stores probabilities that allow it to invent rules on the fly. For example, to invent np rules we might know the probability that an np starts with a determiner (high) or a preposition (low). Similarly, if we are creating a noun-phrase and we have seen a determiner, we might know what the probability is that the next constituent is an adjective (high) or another determiner (low). It should be clear that we can collect such statistics from the tree-bank in much the same way as we collected statistics about

```
(S (NP The (ADJP  most troublesome) report)
   (VP  may
    (VP  be
     (NP (NP the August merchandise trade deficit)
      (ADJP due (ADVP out) (NP tomorrow)))))
   (. .))
```

```
(S (NP The (ADJP most troublesome) report)
   (VP may
    (VP be
     (ADJP (NP the August merchandise trade deficit)
        due)
     (PP out (NP tomorrow))))
   (. .))
```

Figure 5: A real tree-bank example and the parse found by a tree-bank grammar

individual rules. Having collected these statistics, we can then assign a probability that *any* sequence of constituents is any part of speech. Some of these probabilities will be high (e.g., np $\rightarrow$ det adj noun), but most of them will be very low (e.g., np $\rightarrow$ preposition).

To formalize this slightly, we capture the idea of using the probability of adj appearing after det inside an np with probabilities of the form $p(t_i \mid l, t_{i-1})$, where t_{i-1} is the previous constituent and l is the constituent type we are expanding. Naturally, this is but one way to try to capture the regularities; we could condition instead on l and the *two* previous constituents.

We can relate this to our more traditional rule-based scheme by noting that the probability of a rule is the product of the probabilities of its individual components. That is,

$$p(r \mid l) = \prod_{t_i \in r} p(t_i \mid l, t_{i-1}) \tag{5}$$

There are no formal studies on how well different Markov grammars perform or on how they compare to tree-bank grammars. I have looked at this some, and found that for non-lexicalized parsers (i.e., parsers that do not use information about the words, just about the parts of speech), tree-bank grammars seem to work slightly better, thought my colleague Mark Johnson found a Markov scheme that worked ever so slightly better than the tree-bank grammar. But either way, the parsers that have used Markov grammars have mostly been lexicalized, and that is where the real action is.

5 Lexicalized Parsing

The biggest change in statistical parsing over the last few years has been the introduction of statistics on the behavior of individual words. As noted earlier, rather than the 75% precision/recall accuracy of parsers that use statistics based only upon a word's part of speech, lexicalized parsers now achieve 87-88% precision/recall.

Gathering statistics on individual words immediately brings to the fore the sparse data problems we first mentioned in our discussion of tagging. Some words we will have never seen before, and even if we restrict ourselves to those we have already seen, if we try to collect statistics on very detailed combinations of words, the odds of seeing the combination in our training data become increasingly remote. Consider again the noun phrase from Figure 5, "the August merchandise trade deficit." This does not seem a terribly unusual combination (at least, not for text from the Wall Street Journal), but it does not appear in our 900,000-word training set.

To minimize the combinations to be considered, a key idea is that each constituent has a "head," its most important lexical item. For example, the head of a noun phrase is the main noun, which is typically the rightmost. More generally, heads are computed bottom up and the head of a constituent c is a deterministic function of the rule used to expend c. For example, the c is expanded using $\mathsf{s} \rightarrow$ np vp, the function would indicate that one should find the head of the c by looking for the head of the vp. In "the August merchandise trade deficit" the head is "deficit," and even though this noun phrase does not appear in the training set, all of the words in the noun phrase do appear under the head "deficit." This suggests that if we restrict ourselves to statistics on at most pairs of words, the best ones to gather would relate the heads of phrases to the heads of all their subphrases. (A different scheme for lexicalization is proposed by Bod in (Bod [1993]), but its efficacy is still under debate (Goodman [1996b]).)

Lexicalized statistical parsers collect, to a first approximation, two kinds of statistics. One relates the head of a phrase to the rule used to expand the phrase, which we denote $p(r \mid h)$, and the other relates the head of a phrase to the head of a subphrase, which we denote $p(h \mid m, t)$, where h is the head of the subphrase, m the head of the mother phrase, and t the type of subphrase. For example, consider the vp "be ... tomorrow" from Figure 5. Here the probability $p(h \mid m, t)$ would be $p(\text{be} \mid \text{may}, \mathsf{vp})$ while $p(r \mid h)$ would be $p(\text{ posvp} \rightarrow \mathsf{AUX}\ \mathsf{np} \mid \text{be})$. Parsers that use Markov grammars do not actually have $p(r \mid h)$, since they have no rules per se. Rather, this probability is spread out, just as in Equation 5, except now each of the probabilities is be conditioned on the head of the constituent h. In what follows we talk as if all of the systems used rules, since the differences do not seem crucial.

Thus, for a lexicalized parser Equation 4 is replaced by

$$p(s, \pi) = \prod_{c} p(h(c) \mid m(c)) \cdot p(r(c) \mid h(c)) \tag{6}$$

Here we first find the probability of the head of the constituent $h(c)$ given the head of the mother $m(c)$ and then the probability of the rule $r(c)$ given the head of c.

In general, conditioning on heads tightens up the probabilities considerably. For example, consider the probability of our noun phrase "the August merchandise trade deficit." In the following table we give the probabilities for the word "August" given (a) no prior information (i.e., what percentage of all words are "August"), (b) the part of speech (what percentage of all proper nouns are "August"), and (c) part of speech and the head of the phrase above ("deficit"). We do the same for the rule used in this noun phrase np $\rightarrow$ det propernoun noun noun noun.

Conditioning events	p("August")	p(rule)
Nothing	$2.7 \cdot 10^{-4}$	$3.8 \cdot 10^{-5}$
Part of speech	$2.8 \cdot 10^{-3}$	$9.4 \cdot 10^{-5}$
Also $h(c)$ is "deficit"	$1.9 \cdot 10^{-1}$	$6.3 \cdot 10^{-3}$

In both cases the probabilities increase as the conditioning events get more specific.

Another good example of the utility of lexical head information is the problem of prepositional-phrase attachment. Consider the following example from (Hindle & Rooth [1991]):

> Moscow sent more than 100,000 soldiers into Afghanistan.

The problem for a parser is deciding if the pp "into Afghanistan" should be attached to the verb "sent" or the noun-phrase "more than 100,000 soldiers." Hindle and Rooth (Hindle & Rooth [1991]) suggest basing this decision on the compatibility of the preposition "into" with the respective heads of the vp ("sent") and the np ("soldiers"). They measure this compatibility by looking at $p(\text{into} \mid \text{sent})$ (the probability of seeing a pp headed by "into" given it is under a vp headed by "sent" and $p(\text{into} \mid \text{soldiers})$ (its probability given it is inside an np headed by "soldiers"). The pp is attached to the constituent for which this probability is higher. In this case the probabilies match our intuition that it should be attached to "sent", since $p(\text{into} \mid \text{sent}) = .049$ whereas $p(\text{into} \mid \text{soldiers}) = .0007$. Note that the probabilites proposed here are exactly those used in the parsing model based upon Equation 6 where the probability of a constituent c incorporates the probability $p(h(c) \mid m(c))$, the probability of the head of c given the head of the parent of c. For the two pp analyses these would be the probability of the head of the pp (e.g., "into") given the head of the mother (e.g., "sent" or "soldiers").

Figure 6 shows our lexicalized parser's parse of the sentence in Figure 5. Note that this time the parser is not quite as confused by the expression "due out tomorrow" although it makes it a prepositional phrase, not adjective phrase. These improvements can be traced to the variety of ways in which the probabilities conditioned on heads better reflect the way English works. For example, consider the bad ADJP from Figure 5. The probability of the rule adjp $\rightarrow$ np adj is .0092 while the observed probability of this rule given that the head of the phrase is "due" is zero — this combination does not occur in the training corpus.

```
(S (NP The (ADJP most troublesome) report)
   (VP may
    (VP be
      (NP the August merchandise trade deficit)
      (PP due out (NP tomorrow))))
   (. .))
```

Figure 6: The example tree-bank sentence parsed using word statistics

There is one somewhat sobering fact about this last example. Even though the parse in Figure 6 is much more plausible than that in Figure 5, the precision and recall figures are much the same: in both cases the parser finds seven correct constituents. Obviously the number of constituents in the tree-bank parse stays the same (10), so the recall remains 70%. In the lexically based version the number of constituents proposed by the parser decreases from nine to eight, so the precision goes up from 78% to 87%. This hardly reflects our intuitive idea that the second parse is much better than the first. Thus the precision/recall figures do not completely capture our intuitive ideas about the goodness of a parse. But quantification of intuitive ideas is always hard, and most researchers are willing to accept some artificiality in exchange for the boon of being able to measure what they are talking about.

Conditioning on lexical heads, while important, is not the end of the line in what current parsers use to guide their decisions. We also find (1) using the type of the parent to condition the probability of a rule (Charniak [1997]), (2) using information about the immediate left context of a constituent (Collins [1996]), (3) using the classification of noun phrases inside a `vp` as optional or required (Collins [1997]), and (4) considering a very wide variety of possible conditioning information and using a decision-tree learning scheme to pick those that seem to give the most purchase (Magerman [1995]). We expect this list to increase over time.

6 Future Research

The precision and recall measures of the best statistical parsers have been going up year by year for several years now, and as we have noted the best of them is now up to 88%. How much higher can these numbers be pushed? To answer this question one thing we should know is how well people do on this task. I know of no published figures on this question, but the figure that is bandied about in the community is 95%. Given that people achieve 98% at tagging and parsing is obviously more complicated, 95% has a reasonable ring to it, and it is a nice roundish number.

But while there is room for improvement, I believe that it is now (or will soon be) time to stop working on improving labeled precision/recall per se. There are several reasons for this. We have already noted the artificiality that can accompany these measurements, and we should always keep in mind that parsing is a means to an end, not an end in itself. At some point we should move on. And many of the tasks we could move on to might themselves improve parsing accuracy as a byproduct.

One area that deserves more research is statistical parsers that produce trees unlike those in extant tree-banks. We have already noted that this is a tough problem since in this circumstance the tree-bank provides only approximate information about the rules needed in the grammar and when they should be applied. For this reason tree-bank-style parsers typically perform better than those that aim for a style further from the parent tree. Nevertheless, eventually we must move beyond tree-bank styles. Anyone who studies, say, the Penn tree-bank will have objections to some of their analyses, and many of these objections will prove correct. But producing a new tree-bank for each objection is impossible: tree-banks are much too labor-intensive for this to be practicable. Rather, we must find ways to move beyond, using the tree-bank as a first step.

For example, look again at Figure 5, in particular at the np "the August merchandise trade deficit." Notice how flat it is, particularly with regard to the sequence of nouns "August merchandise trade deficit." The problem of noun-noun compounds is an old one. Lauer (Lauer [1995]) suggests statistical techniques for attacking this problem. Interestingly, he too recommends looking at head-head relations. However, nobody has yet incorporated his suggestions into a complete parser, and this is but one of many things that could be improved about the trees produced by tree-bank style parsers.

However, the most important task to be tackled now is parsing into a semantic representation. There are several aspects to this, many of which are covered in the accompanying article on semantic processing. One of these, null elements, deserves special attention here because of its close relation to parsing. A standard example of a null element is in wh-clauses, e.g., "the bone that the dog chewed," where we should recognize that "the dog chewed *the bone*." At least one statistical parser has attacked this problem (Collins [1997]). Another example of null elements is gapping, as in "Sue ate an apple and Fred a pear." Here there is a gap in the phrase "Fred a pear," which obviously should be understood as "Fred *ate* a pear." While there have been numerous theoretical studies of gapping, to the best of my knowledge there has been no statistical work on the topic.

Other aspects of parsing also deserve attention. One is speed. Some of the best statistical parsers are also quite fast (Collins [1996]) and there are the beginnings of theory and practice on how to use statistical information to better guide the parsing process (Caraballo & Charniak [ming]). I would not be surprised if the statistical information at hand could be parlayed into something approaching deterministic parsing.

Before closing, we should at least briefly mention applications of parsing technology. Unfortunately, most natural-language applications do not use parsing, statistical or otherwise. While some statistical parsing work has been directly aimed at particular applications (e.g., parsing for machine translation (Wu [1995])) and applications like speech recognition require parsers with particular features not present in current statistical models (e.g., fairly strict left-to-right parsing), we believe that the greatest impetus to parsing applications will be the natural course of things toward better, faster, and, for parsing, deeper.

References

Rens Bod [1993], "Using an annotated language corpus as a virtual stochastic grammar," in *Proceedings of the Eleventh National Conference on Artificial Intelligence*, AAAI Press/MIT Press, Menlo Park, 778–783.

Eric Brill [1993], "Automatic grammar induction and parsing free text: a transformation-based approach," in *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, 259–265.

Eric Brill [1995a], "Transformation-based error-driven learning and natural language processing: a case study in part-of-speech tagging," *Computational Linguistics* 21, 543–566.

Eric Brill [1995b], "Unsupervised learning of disambiguation rules for part of speech tagging," in *Proceedings of the Third Workshop on Very Large Corpora*, 1–13.

Ted Briscoe & Nick Waegner [1993], "Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars," *Computational Linguistics* 19, 25–69.

Sharon Caraballo & Eugene Charniak [forthcomingg], "Figures of merit for best-first probabilistic chart parsing," *Computational Linguistics* .

Eugene Charniak [1993], *Statistical Language Learning*, MIT Press, Cambridge.

Eugene Charniak [1996], "Tree-bank grammars," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, AAAI Press/MIT Press, Menlo Park, 1031–1036.

Eugene Charniak [1997], "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.

Kenneth Ward Church [1988], "A stochastic parts program and noun phrase parser for unrestricted text," in *Second Conference on Applied Natural Language Processing*, ACL, 136–143.

Michael John Collins [1996], "A new statistical parser based on bigram lexical dependencies," in *Proceedings of the 34th Annual Meeting of the ACL*.

Michael John Collins [1997], "Three generative lexicalised models for statistical parsing," in *Proceedings of the 35th Annual Meeting of the ACL*.

Joshua Goodman [1996a], "Parsing algorithms and metrics," in *Proceedings of the 34th Annual Meeting of the ACL*, 177–183.

Joshua Goodman [1996b], "Efficient algorithms for parsing the DOP model," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 143–152.

Donald Hindle & Mats Rooth [1991], "Structural ambiguity and lexical relations," in *Proceedings of the Association for Computational Linguistics*, ACL, 229 – 236.

Frederick Jelinek & John D. Lafferty [1991], "Computation of the probability of initial substring generation by stochastic context-free grammars," *Computational Linguistics* 17, 315–324.

Mark Lauer [1995], "Corpus statistics meet the noun compound: some empirical results," in *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, 47–55.

S. E. Levinson, L. R. Rabiner & M. M. Sondhi [1983], "An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition," *The Bell System Technical Journal* 62 , 1035–1074.

David M. Magerman [1995], "Statistical decision-tree models for parsing," in *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, 276–283.

Mitchell P. Marcus, Beatrice Santorini & Mary Ann Marcinkiewicz [1993], "Building a large annotated corpus of English: the Penn treebank," *Computational Linguistics* 19, 313–330.

Bernard Merialdo [1994], "Tagging English text with a probabilistic model," *Computational Linguistics* 20, 155–172.

Fernando Pereira & Yves Schabes [1992], "Inside-outside reestimation from partially bracketed corpora," in *27th Annual Meeting of the Association for Computaitonal Linguistics*, ACL, 128–135.

Emmanuel Roche & Yves Schabes [1995], "Deterministic part-of-speech tagging with finite-state transducers," *Computational Linguistics* 21, 227–254.

Andreas Stolcke [1995], "An efficient probabilistic context-free parsing algorithm that computes prefix probabilities," *Computational Linguistics* 21, 165–202.

Ralph Weischedel, Marie Meteer, Richard Schwartz, Lance Ramshaw & Jeff Palmucci [1993], "Coping with ambiguity and unknown words through probabilistic models," *Computational Linguistics* 19, 359–382.

Dekai Wu [1995], "An algorithm for simultaneously bracketing parallel texts by aligning words," in *Proceedings of the 33rd Annual Meeting of the ACL*, 244–251.