Two Experiments on Learning Probabilistic Dependency Grammars from Corpora

Glenn Carroll and Eugene Charniak*
Department of Computer Science
Brown University
Providence RI 02912

Introduction

We present a scheme for learning probabilistic dependency grammars from positive training examples plus constraints on rules. In particular we present the results of two experiments. The first, in which the constraints were minimal, was unsuccessful. The second, with significant constraints, was successful within the bounds of the task we had set.

We will explicate dependency grammars in Section 2. For the moment we simply note that they are a very restricted class of grammars which do not fit exactly into the Chomsky hierarchy, but whose appearance is most like the context-free grammars.

We assume that the goal of learning a context-free grammar needs no justification. The problem has attracted a fair amount of attention, ([1,4] are good surveys.) but no good solutions have been found. Our choice of learning from only positive training examples needs only a little more justification. Obviously, if it is possible, a scheme which only uses positive training examples is preferable in that one can simply give the learner the examples (henceforth called the "corpus") and no further work is required in the form of a "teacher." That such a scheme may be possible is suggested by the fact that children seem to learn languages with little correction to their speech. To keep things simple, initially, we assume that the corpus contains parts of speech, and not words. That is, the input is a tagged corpus, minus the words.

We have chosen to learn *probabilistic* grammars because the scheme we use requires it. Such a grammar assigns a probability to every string in the language. As we will see, our algorithm tries to find a probabilistic dependency grammar which makes the training-corpus as likely as possible. The idea is that rules which generate sentences not in the language, but which nevertheless have non-zero probability will make the probability assigned to the corpus lower than need be. (Since the incorrect sentences would have non-zero probability, a better scheme which did not have the rule could, in effect, divide up this probability among the correct sentences, making the ones found in the corpus more likely.)

Moving down one layer of detail, our scheme works as follows:

0. Divide the corpus into two parts, hereafter called the rule corpus and the training corpus.
1. For all of the sentences in the rule corpus, generate all rules which might be used to generate (and/or parse) the sentence, subject to constraints which we will specify later.
2. Estimate the probabilities for the rules.
3. Using the training corpus, improve our estimate of the probabilities.
4. Delete all rules with probability $\leq \delta$ for some small δ. What remains is the grammar.

Steps 0 and 4 are trivial. Step 3 is not, but the technology is now reasonably well established. We have used the inside-outside algorithm [2,5]. This is an iterative algorithm which takes a context-free grammar and initial probability estimate and sees how often each rule would be used in generating the training corpus. These counts then form the basis of an improved estimate of rule probabilities. These probabilities are then used to get an improved count and the process repeats. At each iteration the inside-outside algorithm is guaranteed to lower (or keep constant) the estimated cross-entropy (the negative log of the probability assigned by the model) of the text given the probabilistic context-free grammar. Unfortunately, the entropy minimum found need not be the global minimum. At the beginning of this research we assumed that this would not be a problem. The first experiment shows that this assumption is incorrect.

Returning to the outline of our scheme, we also need to make an initial estimate for the probabilities. We have adopted a scheme which uses the fact that the possible rules are generated for each sentence (step 1). It is possible (indeed likely) that a rule will be generated several times, each for a different sentence (or a different part of the same sentence). We keep a count of how many times each rule is generated. Let C_r be the count for rule r. Let R_A be the set of rules with with

*We would like to thank Mark Johnson for many useful discussions. This research was supported in part by NSF contract IRI-8911122 and ONR contract N0014-91-J-1202.

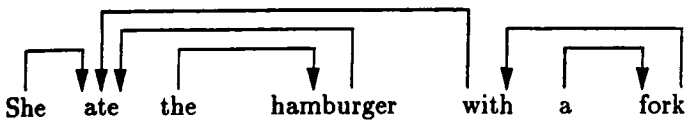

Figure 1: Informal dependency-grammar notation

non-terminal A on the left-hand side. Then the initial probability estimate P' for rule $r \in R_A$ is given by

$$P'(r) = \frac{C_r}{\sum_{r_j \in R_A} C_{r_j}} \quad (1)$$

At this point we have taken care of steps 2-4, leaving only step 1 of our algorithm to be filled out. That is, of course, where the difficulties arise.

Dependency Grammars and Their Properties

The most obvious difficulty with step 1 is that there are an unbounded number of rules which could lead to a given sentence, given an unbounded set of non-terminals. If we specify a finite set of non-terminals then the set of rules is finite, but would be, in general, very large. Thus we need some grammar *schema* which will limit the number of applicable rules to some more manageable number. The schema we adopt for the purpose of this experiment is dependency grammar.[1]

Informally, a dependency grammar produces a set of terminals connected by a set of directed arcs — one arc for every terminal except the root terminal. See Figure 1. Formally, a dependency grammar is a 3-tuple $< S, N, R >$ where S is the start symbol for the rewrite rules, N a set of terminal symbols, and R a set of rewrite rules, where R is a subset of the set defined as follows: $\{S \rightarrow \overline{n} \mid n \in N\} \cup \{\overline{n} \rightarrow \alpha n \beta \mid n \in N, \alpha, \beta \in \Gamma\}$ where Γ is a set of strings of zero or more $\overline{a}$, for $a \in N$. In standard linguistic usage, a bar, as in $\overline{x}$, indicates a function from terminals and non-terminals to non-terminals. In our usage, a bar indicates a function from terminals to non-terminals. Such a rewrite system produces a tree structure in the normal way. For the above example, the tree structure would be as shown in Figure .

Since the set of non-terminals is restricted to $\{\overline{A} \mid A \in N\} \cup \{S\}$, the number of rules which could be used to generate a sentence is finite. To be more precise about this, we will say that a context-free rule r *conforms* to a sentence s (with respect to a set of rules R) if it is possible for any expansion of the right-hand side of r (using the rules in R) to produce a sequence of terminals found in s. Intuitively, a rule conforms if it can be used in some parse of s. We then have the following theorem:

Theorem 1: A sentence of length n, consisting of all distinct terminals will have $n(2^{n-1}+1)$ dependency grammar rules which conform to it (with respect to the set of all dependency grammar rules on the terminals in the sentence). (Proof is omitted.)

For example, Figure shows the $15 = 3(2^{3-1}+1)$ rules conforming to the string "det noun verb". When not all of the terminals are the same, then the number of distinct rules is less, but still grows quickly with sentence length.

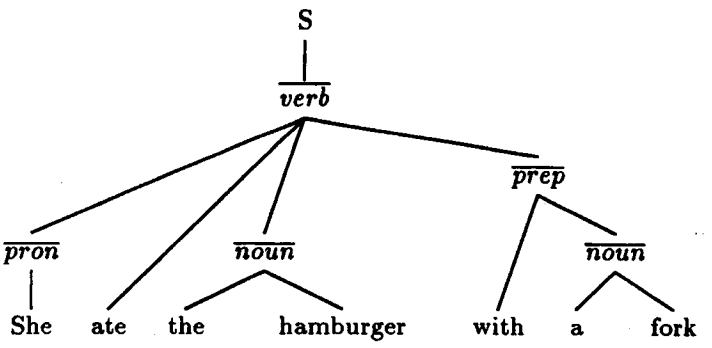

Figure 2: Tree-structure version of a dependency-grammar parse

Ordering the Corpus

It should be clear that even restricting consideration to dependency grammars, the algorithm described in section 1 is not feasible. The first sentence picked at random from a book one of the authors was reading had 41 terminal symbols. The number of rules which conform to such a sentence is

$$(41(2^{40}+1) \approx 41((2^{10})^4) \approx 40((10^3)^4) \approx 4 \cdot 10^{13}$$

It is also true that generating all conforming rules for the *entire* corpus (as was implied by the algorithm in Section 1) will generate a large number of rules. We need some way to trim the number of rules we will consider.

The scheme we present starts from the observation that children are exposed to simple language before seeing more complex language. Thus it occurred to us that we might order the sentences in terms of their length,[2] and apply the above learning algorithm iteratively, over groups of successively longer sentences. The idea is that the learning algorithm will make decisions about which simple rules to keep—and which simple rules to discard—after seeing only a small subset of the corpus. Once a rule has been rejected, we do not allow it to be re-constructed later on in the learning process. Thus, as we move to more complex sentences, we need only generate new rules which conform to the sentences, *given our restricted set of rules.* We need never generate all 10^{13} rules for the example above, nor need we ever try and parse a sentence using such an enormous rule set.

[1] We got this idea from Martin Kay, who mentioned it to Mark Johnson, who in turn mentioned it to us. Thanks to both of them.

[2] One might also order by "complexity" measures other than length, such as the use of comparatively rare parts of speech. Currently we are only using length.

S	→	$\overline{det}$	$\overline{noun}$	→	noun
S	→	$\overline{noun}$	$\overline{noun}$	→	$\overline{det}$ noun
S	→	$\overline{verb}$	$\overline{noun}$	→	noun $\overline{verb}$
$\overline{det}$	→	det	$\overline{noun}$	→	$\overline{det}$ noun $\overline{verb}$
$\overline{det}$	→	det $\overline{noun}$	$\overline{verb}$	→	verb
$\overline{det}$	→	det $\overline{verb}$	$\overline{verb}$	→	$\overline{det}$ verb
$\overline{det}$	→	det $\overline{noun}$ $\overline{verb}$	$\overline{verb}$	→	$\overline{noun}$ verb
			$\overline{verb}$	→	$\overline{det}$ $\overline{noun}$ verb

Figure 3: The 15 rules which conform to "det noun verb"

1.0	S	→	$\overline{verb}$
.2	$\overline{verb}$	→	verb
.4	$\overline{verb}$	→	$\overline{noun}$ verb
.4	$\overline{verb}$	→	verb $\overline{noun}$
.5	$\overline{noun}$	→	noun
.5	$\overline{noun}$	→	$\overline{det}$ noun

Figure 5: Intended solution for the simple corpus

1.0	S	→	$\overline{verb}$
.2	$\overline{verb}$	→	verb
.2	$\overline{verb}$	→	$\overline{noun}$ verb
.2	$\overline{verb}$	→	verb $\overline{noun}$
.2	$\overline{verb}$	→	$\overline{det}$ $\overline{noun}$ verb
.2	$\overline{verb}$	→	verb $\overline{det}$ $\overline{noun}$

Figure 6: Solution actually found for the simple corpus

To further reduce the number of rules, we added an arbitrary limit on the number of symbols which can appear on the right-hand side of a rule. With these restrictions we anticipated that the number of rules suggested would be kept to a manageable number.

An Example

To get some feel for what is involved in this algorithm, consider the problem of inferring the dependency grammar for the following toy corpus:

"noun verb" "verb noun" "verb"
"det noun verb" "verb det noun"

The program generated the 22 rules of Figure as those which conform with the sentences of the corpus. The probabilities there are those assigned by equation 1. We expected that the program would eventually choose rules (and probabilities) shown in Figure 5. In point of fact, it turns out that there are two different grammars which both assign an entropy of 1.054 bits/word to the corpus. The second is shown in Figure 6. It was this second grammar which our program found. It is interesting, however, to see how it arrived at this result.

The reader will remember that the algorithm for obtaining the (locally) best probabilities for the rules is iterative. The cross-entropy of the corpus was calculated after every iteration. It decreased rapidly for 6 iterations, and then at at more leisurely pace for another 14 iterations, until the decrease was less than the .001 bits/word limit we imposed. Figure shows the probabilities assigned to the rules after 0, 6, and 20 iterations. Note the probabilities after 6 iterations. All of the rules which are in neither of the rule sets found in Figures 5 or 6 have received zero probabilities. The rules which appear in both sets have probability one, and those which appear in one of the sets have probabilities between those they assume in the two sets. To put it another way, the algorithm effectively first narrows the possible theories down to two, and then chooses one over the other.

Upon thinking about this, we realized that it was not a coincidence that there was a second equally good grammar. Note that the preferred grammar had one rule for each sentence. We will call such grammar a "memorizing grammar." It is not too hard to see that a memorizing grammar for a corpus will assign the highest probability of any probabilistic context-free grammar for that corpus. [3]

This is a problem, since memorizing grammars are not very interesting. However, in our outline of the algorithm in Section 1 we specified that it generates the rules on one set of examples and evaluates the probabilities on a disjoint set of examples from the same corpus. What memorized one set, will, of course, not work on the other, while the more general solution should do fine. It is also the case that the limit on rule length will inhibit the emergence of a memorizing grammar (provided that we have sentences of length greater than the maximum rule length).

Experiment One

In both experiments we used the dependency grammar of Figure 7 to randomly generate sentences until a certain number of words had been generated. In this way

[3] Although there may be other equally good grammars.

RULE			0 ITERS	6 ITERS	20 ITERS
S	→	$\overline{det}$	.181818	0.0	0.0
S	→	$\overline{noun}$	.363636	0.0	0.0
S	→	$\overline{verb}$	.454545	1.0	1.0
$\overline{det}$	→	det	.250000	1.0	1.0
$\overline{det}$	→	det $\overline{noun}$	.25	0.0	0.0
$\overline{det}$	→	det $\overline{verb}$	.125	0.0	0.0
$\overline{det}$	→	det $\overline{noun}$ $\overline{verb}$	.125000	0.0	0.0
$\overline{det}$	→	$\overline{verb}$ det	.125000	0.0	0.0
$\overline{det}$	→	$\overline{verb}$ det $\overline{noun}$	.125000	0.0	0.0
$\overline{noun}$	→	noun	.333333	.781317	.998847
$\overline{noun}$	→	$\overline{det}$ noun	.166667	.218683	.001153
$\overline{noun}$	→	noun $\overline{verb}$	.166667	0.0	0.0
$\overline{noun}$	→	$\overline{verb}$ noun	.166667	0.0	0.0
$\overline{noun}$	→	$\overline{det}$ noun $\overline{verb}$	.083333	0.0	0.0
$\overline{noun}$	→	$\overline{verb}$ $\overline{det}$ noun	.083333	0.0	0.0
$\overline{verb}$	→	verb	.384615	.20	.20
$\overline{verb}$	→	$\overline{det}$ verb	.076923	0.0	0.0
$\overline{verb}$	→	$\overline{noun}$ verb	.153846	.286749	.200461
$\overline{verb}$	→	$\overline{det}$ $\overline{noun}$ verb	.076923	.113251	.199539
$\overline{verb}$	→	verb $\overline{det}$ $\overline{noun}$	.076923	.111803	.199539
$\overline{verb}$	→	verb $\overline{det}$	.076923	0.0	0.0
$\overline{verb}$	→	verb $\overline{noun}$	.153846	.288197	.200461

Figure 4: The 22 rules which conform to the simple corpus

we created two sets of sentences with slightly more than 1000 words for the rule corpus, and 9000 words for the training corpus. Each subset was sorted by length. The goal, of course, was to see how close the learning algorithm could come to the original grammar.

To keep things simple for the grammar learner, rules were required to have fewer than five symbols on their right-hand side, as the longest rule in the grammar of Figure 7 had four symbols. The learning algorithm worked as shown in Figure 8.

The algorithm deletes rules with low probability after a trial period in which they have time to "prove" themselves by increasing in probability. To keep track of a rule's age, the algorithm assigns each terminal, t, a "date", d_t, which is the length of the shortest sentence in which t appears. It also assigns d_t to $d_{\overline{t}}$. Informally, we say that a terminal or non-terminal c is part of ($\in$) a rule r if it appears on the right- or left-hand side of the rule. A rule is eligible for deletion when:

$$| \, sentence \, | \leq 2 \, | \, rule - rhs \, | + \max_{c \in r} d_c \qquad (2)$$

We have tried several variations on this decision criterion, none of which have had any effect on the negative results of this section.

As we have indicated, the results obtained by the algorithm were uniformly awful. Figure 9 shows the eight most probable expansions for $\overline{pron}$ found by the system after seeing sentences of up to length eight. The correct rule, $\overline{pron}$ →pron, does not appear in the grammar at all, and was in fact eliminated by the test of equation 2 earlier in the run.

If one really were only interested in reducing the cross entropy of the corpus, then this grammar is not too bad, with a cross entropy of 1.245 bits/word on the training corpus sentences of length up to eight. However, it is not the minimum, since the correct grammar, when trained on these sentences, produced a cross entroy of 1.220 bits/word.

So the inside-outside algorithm did not find a global minimum, but rather a local one. This set us to wondering if this was a fluke, or if, as seemed more likely now, there might be lots of local minima. (This grammar was sufficiently bizarre that it seemed unlikely that it was somehow special.) If there were lots of local minima then the particular grammar found would be very sensitive to the initial probability estimates. To test this, we took all of the rules generated from sentences of length ≤ 8 and repeatedly assigned random probabilities to them. For each assignment we trained the rules and observed the resulting grammars. If there were lots of local minima we should end up with lots of different grammars. We classified two grammars as the same if they had the same rules, and the rules for expanding a particular non-terminal, when sorted by probability, were in the same order. Of the 300 starting points tried, we found 300 different local minima. It should also be noted that *none* of the grammars found in the experiment were the correct grammar. This indicates that

1.0	S	→	÷
1.0	÷	→	$\overline{verb}$.
.1	$\overline{verb}$	→	verb
.05	$\overline{verb}$	→	$\overline{noun}$ verb
.05	$\overline{verb}$	→	$\overline{pron}$ verb
.05	$\overline{verb}$	→	verb $\overline{noun}$
.05	$\overline{verb}$	→	verb $\overline{pron}$
.1	$\overline{verb}$	→	$\overline{noun}$ verb $\overline{noun}$
.05	$\overline{verb}$	→	$\overline{pron}$ verb $\overline{noun}$
.05	$\overline{verb}$	→	$\overline{noun}$ verb $\overline{pron}$
.05	$\overline{verb}$	→	$\overline{pron}$ verb $\overline{noun}$ $\overline{noun}$
.05	$\overline{verb}$	→	$\overline{noun}$ verb $\overline{noun}$ $\overline{noun}$
.1	$\overline{verb}$	→	$\overline{noun}$ verb $\overline{noun}$ $\overline{prep}$
.1	$\overline{verb}$	→	$\overline{pron}$ verb $\overline{noun}$ $\overline{prep}$
.1	$\overline{verb}$	→	$\overline{noun}$ verb $\overline{pron}$ $\overline{prep}$
.05	$\overline{verb}$	→	$\overline{noun}$ verb $\overline{prep}$
.05	$\overline{verb}$	→	$\overline{pron}$ verb $\overline{prep}$
.1	$\overline{noun}$	→	noun
.3	$\overline{noun}$	→	$\overline{det}$ noun
.1	$\overline{noun}$	→	$\overline{det}$ $\overline{adj}$ noun
.2	$\overline{noun}$	→	$\overline{det}$ noun $\overline{prep}$
.2	$\overline{noun}$	→	$\overline{det}$ noun $\overline{wh}$
.05	$\overline{noun}$	→	noun $\overline{prep}$
.05	$\overline{noun}$	→	noun $\overline{wh}$
1.0	$\overline{pron}$	→	pron
1.0	$\overline{adj}$	→	adj
1.0	$\overline{det}$	→	det
1.0	$\overline{wh}$	→	wh $\overline{verb}$
.7	$\overline{prep}$	→	prep $\overline{noun}$
.3	$\overline{prep}$	→	prep $\overline{pron}$

Figure 7: Dependency grammar used to generate test corpus

Loop for i from 2 until i > sentence-length-stopping-point

- Add rules required for the sentences with length i from the rule-creation subset.
- Estimate the probabilities for all (non-removed) based upon all sentences of length ≤ i from the rule-training subset.
- Remove any rules with probability ≤ .001 if equation 2 is true of the rule.

Figure 8: Flow of control for the learning algorithm

.220	$\overline{pron}$	→	pron $\overline{verb}$
.214	$\overline{pron}$	→	$\overline{prep}$ pron
.139	$\overline{pron}$	→	pron $\overline{verb}$ $\overline{det}$
.118	$\overline{pron}$	→	$\overline{verb}$ pron
.117	$\overline{pron}$	→	$\overline{det}$ $\overline{verb}$ pron
.038	$\overline{pron}$	→	pron $\overline{verb}$ $\overline{noun}$
.023	$\overline{pron}$	→	$\overline{noun}$ $\overline{verb}$ pron
.013	$\overline{pron}$	→	pron $\overline{verb}$ $\overline{det}$ $\overline{det}$

Figure 9: Some expansions for $\overline{pron}$ found by the unrestricted system

lhs	noun	verb	pron	det	prep	adj	wh	.
noun	o	o	o	○	○	○	○	o
verb	○	o	○	o	○	o	o	o
pron	o	×	o	o	o	o	o	o
det	×	o	o	o	o	×	o	o
prep	○	o	○	o	o	o	o	o
adj	×	o	o	×	o	o	o	o
wh	×	○	o	o	o	o	o	o
.	o	○	o	o	o	o	o	o

Figure 10: For each left-hand side, non-terminals allowed on right

there are, in fact, a considerable number of local minima, and that this is indeed a problem. Wyard [8] found a similar problem with local minima in his genetic algorithm approach.

In general, the incorrect grammars found in these experiments are characterized by large numbers of long rules (compared to the correct grammar). Some grammar learning schemes maximize not just the fit of the grammar to the corpus, but that times a term intended to represent the prior probability of the grammar, where the latter goes down quickly with the complexity of the grammar [3,7]. We are, in general, positively inclined to such a move, and it might well help things here, but at the moment we see no way to include such a number in our calculations. (We did try a modification of the initial probability estimate calculation in which longer rules were given lower initial probabilities. This did produce grammars with slightly shorter rules, but still a very large number of rules, and overall the grammars were equally bizarre.)

Experiment Two

Given the terrible results of experiment one, we decided to place more restrictions on the grammar. In particular, we gave the system a table of what non-terminals may appear on the right-hand side of a rule with a particular non-terminal on the left. (Because we are using a dependency grammar, the terminals are already fixed.) Figure 10 has the left-hand side non-terminals on the left. For each non-terminal on the right ○ indicates that this dependency is used in the grammar (and is

therefore had to be allowed). A cross (×) indicates that the dependency had to be disallowed for the correct grammar to be learned. A dot (o) indicates an optional constraint; the correct grammar could be learned with or without it.

We started out cautiously, allowing only those dependencies absolutely required by the correct grammar. The parameter "sentence-length-stopping-point" of Figure 8 was set to 20. We found that the algorithm converged on the correct grammar after sentences of length 14 or so, the exact number depending on the constraints, but never higher than 15. We then discovered that we could allow almost all dependencies, and still have our procedure converge to the correct grammar, before reaching the stopping point. Only the six indicated restrictions are, in fact, required for the grammar to be learned correctly. Indeed it would make more sense to talk about what constituents are *not* allowed, rather than those which are. In this sense we have something close in spirit to the "distituent grammar" of [6].

Having looked at the results of learning runs in which the above constraints were violated, it is possible to assign "purposes" to most of the above restrictions although to what degree these are idiosyncratic to the grammar at hand only further experimentation will show. The restrictions on adjectives, determiners, and wh prevent $\overline{det}$, $\overline{adj}$, and $\overline{wh}$ from being used as substitutes for $\overline{noun}$ when they are present in a sentence. The restriction preventing $\overline{verb}$ in the expansion of $\overline{pron}$ prevents the strange rules of Figure 9.

We cannot make any claim that such restrictions are "reasonable" in the sense that a child would necessarily have them. However these restrictions were not completely arbitrary either. Obviously, a child has access to semantic information which presumably influences grammar selection. For example, suppose a child knows that the color of an object depends on the object in a way that the object does not depend on the color. Obviously such information could not be input to our program, but—extrapolating rather a lot—we could say that the child has information that while an adjective might depend on the noun, the reverse is not true. This could be expressed, and looking across the line for adjectives in Figure 10, we see that we do not allow nouns and determiners to be part of adjective phrases.

For the moment, however, we justify these constraints in more practical terms. They are not hard to think up or specify, and as Figure 10 shows, it may not be necessary to be all that accurate in their specification.

Future Research

There are three avenues for future work which immediately present themselves: (a) lessening the constraints imposed upon the rules in experiment two, (b) trying the algorithm on real corpora (e.g. the Brown Corpus) and (c) finding and programming other grammar schema. The import of the first two of these should be obvious, the third requires discussion.

Although dependency grammar is used by some grammarians, it has, in fact, severe limitations. To take an example we immediately stumbled over, there does not seem to be a good way to handle wh-movement within the dependency grammar formalism. In unrestricted context-free grammars wh-movement is handled using the so-called "slash categories". For example, restrictive relatives are handled with some rules such as:

$$\begin{array}{lcl} \text{NP} & \rightarrow & \text{NP wh S/NP} \\ \text{S/NP} & \rightarrow & \text{VP} \\ \text{S/NP} & \rightarrow & \text{NP VP/NP} \\ & \text{etc.} & \end{array}$$

Here, of course, S/NP is to be understood as a nonterminal which expands into an S missing an NP. The observation that such dodges exist for a wide variety of cases, and that the necessary context-free rules can be generated automatically from a smaller set of rules, engendered the explosion of work on context-free representations for language.

Unfortunately, this dodge does not translate over into dependency grammar. The trick requires two distinct symbols for a sentence: S and S/NP. In dependency grammar we use $\overline{verb}$ for S, but there are no free nonterminals to use for S/NP. We could, of course ignore the difference between S and S/NP. This is what we did in the grammar in Figure 7, but this is hardly a good solution.

Naturally, the fact that dependency grammar seemingly cannot handle wh-movement does not prevent us looking for a dependency grammar for the Brown Corpus. It simply means that the grammar is bound to over-generate, and thus will not produce as low a cross-entropy as one might otherwise obtain. So we still intend to see what sort of grammar our algorithm finds. However, we are also on the lookout for other grammar schemas which have the desirable property of dependency grammars (small numbers of conforming rules) while being more robust in their handling of natural-language idiosyncrasies.

References

1. ANGLUIN, D. AND SMITH, C. H. Inductive inference: theory and methods. *Computing Surveys 15*3 (1983).
2. BAKER, J. K. Trainable grammars for speech recognition. Presented at *Proceedings of the Spring Conference of the Acoustical Society of America* (1979).
3. COOK, C. M., ROSENFELD, A. AND ARONSON, A. R. Grammatical inference by hill climbing. *Informational Sciences 10* (1976), 59–80.
4. FU, K. S. AND BOOTH, T. L. Grammatical inference: introduction and survey, parts 1 and 2. *IEEE Transations on Systems Man and Cybernetics SMC-5* (1975), 95–111 and 409-423.

5. JELINEK, F., LAFFERTY, J. D. AND MERCER, R. L. Basic methods of probabilistic context free grammars. Continuous Speech Recognition Group IBM T.J. Watson Research Center, 1991.

6. MAGERMAN, D. M. AND MARCUS, M. P. Parsing a natural language using mutual information statistics. Presented at *Proceedings of the Eighth National Conference on Artificial Intelligence* (1990).

7. MUDE, A. V. DER AND WALKER, A. On the inference of stochastic regular grammars. Presented at *Information and Control* (1978).

8. WYARD, P. Context free grammar induction using genetic algorithms. Presented at *Proceedings of the 4th International Conference on Genetic Algorithmss*, San Mateo (1991).