PARSING THE LOB CORPUS

Carl G. de Marcken
MIT AI Laboratory Room 838
545 Technology Square
Cambridge, MA 02142
Internet: cgdemarc@ai.mit.edu

ABSTRACT

This paper[1] presents a rapid and robust parsing system currently used to learn from large bodies of unedited text. The system contains a multivalued part-of-speech disambiguator and a novel parser employing bottom-up recognition to find the constituent phrases of larger structures that might be too difficult to analyze. The results of applying the disambiguator and parser to large sections of the Lancaster/Oslo-Bergen corpus are presented.

INTRODUCTION

We have implemented and tested a parsing system which is rapid and robust enough to apply to large bodies of unedited text. We have used our system to gather data from the Lancaster/Oslo-Bergen (LOB) corpus, generating parses which conform to a version of current Government-Binding theory, and aim to use the system to parse 25 million words of text

The system consists of an interface to the LOB corpus, a part of speech disambiguator, and a novel parser. The disambiguator uses multivaluedness to perform, in conjunction with the parser, substantially more accurately than current algorithms. The parser employs bottom-up recognition to create rules which fire top-down, enabling it to rapidly parse the constituent phrases of a larger structure that might itself be difficult to analyze. The complexity of some of the free text in the LOB demands this, and we have not sought to parse sentences completely, but rather to ensure that our parses are accurate. The parser output can be modified to conform to any of a number of linguistic theories. This paper is divided into sections discussing the LOB corpus, statistical disambiguation, the parser, and our results.

[1] This paper reports work done at the MIT Artificial Intelligence Laboratory. Support for this research was provided in part by grants from the National Science Foundation (under a Presidential Young Investigator award to Prof. Robert C. Berwick); the Kapor Family Foundation; and the Siemens Corporation.

THE LOB CORPUS

The Lancaster/Oslo-Bergen Corpus is an online collection of more than 1,000,000 words of English text taken from a variety of sources, broken up into sentences which are often 50 or more words long. Approximately 40,000 different words and 50,000 sentences appear in the corpus.

We have used the LOB corpus in a standard way to build several statistical tables of part of speech usage. Foremost is a dictionary keying every word found in the corpus to the number of times it is used as a certain part of speech, which allows us to compute the probability that a word takes on a given part of speech. In addition, we recorded the number of times each part of speech occurred in the corpus, and built a digram array, listing the number of times one part of speech was followed by another. These numbers can be used to compute the probability of one category preceding another. Some disambiguation schemes require knowing the number of trigram occurrences (three specific categories in a row). Unfortunately, with a 132 category system and only one million words of tagged text, the statistical accuracy of LOB trigrams would be minimal. Indeed, even in the digram table we have built, fewer than 3100 of the 17,500 digrams occur more than 10 times. When using the digram table in statistical schemes, we treat each of the 10,500 digrams which never occur as if they occur once.

STATISTICAL DISAMBIGUATION

Many different schemes have been proposed to disambiguate word categories before or during parsing. One common style of disambiguators, detailed in this paper, rely on statistical cooccurance information such as that discussed in the section above. Specific statistical disambiguators are described in both DeRose 1988 and Church 1988. They can be thought of as algorithms which maximize a function over the possible selections of categories. For instance, for each word A^z in a sentence, the DeRose algorithm takes a set of categories $\{a_1^z, a_2^z, \ldots\}$ as input. It outputs a particular category $a_{i_z}^z$ such

that the product of the probability that A^z is the category $a^z_{i_z}$ and the probability that the category $a^z_{i_z}$ occurs before the category $a^{z+1}_{i_{z+1}}$ is maximized. Although such an algorithm might seem to be exponential in sentence length since there are an exponential number of combinations of categories, its limited leftward and rightward dependencies permit linear time dynamic programming method. Applying his algorithm to the Brown Corpus[2], DeRose claims the accuracy rate of 96%. Throughout this paper we will present accuracy figures in terms of how often words are incorrectly disambiguated. Thus, we write 96% correctness as an accuracy of 25 (words per error).

We have applied the DeRose scheme and several variations to the LOB corpus in order to find an optimal disambiguation method, and display our findings below in Figure 1. First, we describe the four functions we maximize:

Method A: Method A is also described in the DeRose paper. It maximizes the product of the probabilities of each category occurring before the next, or

$$\prod_{z=1}^{n-1} \mathrm{P}\left(a^z_{i_z} \text{is-flwd-by } a^{z+1}_{i_{z+1}}\right)$$

Method B: Method B is the other half of the DeRose scheme, maximizing the product of the probabilities of each category occurring for its word. Method B simply selects each word's most probable category, regardless of context.

$$\prod_{z=1}^{n} \mathrm{P}\left(A^z \text{is-cat } a^z_{i_z}\right)$$

Method C: The DeRose scheme, or the maximum of

$$\prod_{z=1}^{n} \mathrm{P}\left(A^z \text{is-cat } a^z_{i_z}\right) \prod_{z=1}^{n-1} \mathrm{P}\left(a^z_{i_z} \text{is-flwd-by } a^{z+1}_{i_{z+1}}\right)$$

Method D: No statistical disambiguator can perform perfectly if it only returns one part of speech per word, because there are words and sequences of words which can be truly ambiguous in certain contexts. Method D addresses this problem by on occasion returning more than one category per word.

The DeRose algorithm moves from left to right assigning to each category a^z_j an optimal path of categories leading from the start of the sentence to a^z_j, and a corresponding probability. It then extends each path with the categories of the word A^{z+1} and computes new probabilities for the new paths. Call the greatest new probability P. Method D assigns to the word A^z those categories $\{a^z_i\}$ which occur in those new paths which have a probability within a factor F of P. It remains a linear time algorithm.

Naturally, Method D will return several categories for some words, and only one for others, depending on the particular sentence and the factor F. If $F = 1$, Method D will return only one category per word, but they are not necessarily the same categories as DeRose would return. A more obvious variation of DeRose, in which alternate categories are substituted into the DeRose disambiguation and accepted if they do not reduce the overall disambiguation probability significantly, would approach DeRose as F went to 1, but turns out not to perform as well as Method D.[3]

Disambiguator Results: Each method was applied to the same 64,000 words of the LOB corpus. The results were compared to the LOB part of speech pre-tags, and are listed in Figure 1.[4] If a word was pre-tagged as being a proper noun, the proper noun category was included in the dictionary, but no special information such as capitalization was used to distinguish that category from others during disambiguation. For that reason, when judging accuracy, we provide two metrics: one simply comparing disambiguator output with the pre-tags, and another that gives the disambiguator the benefit of the doubt on proper nouns, under the assumption that an "oracle" pre-processor could distinguish proper nouns from contextual or capitalization information. Since Method D can return several categories for each word, we provide the average number of categories per word returned, and we also note the setting of the parameter F, which determines how many categories, on average, are returned.

The numbers in Figure 1 show that simple statistical schemes can accurately disambiguate parts of speech in normal text, confirming DeRose and others. The extraordinary

[2] The Brown Corpus is a large, tagged text database quite similar to the LOB.

[3] To be more precise, for a given average number of parts of speech returned V, the "substitution" method is about 10% less accurate when $1 \leq V < 1.1$ and is almost 50% less accurate for $1.1 \leq V < 1.2$.

[4] In all figures quoted, punctuation marks have been counted as words, and are treated as parts of speech by the statistical disambiguators.

Method:	A	B	C	D(1)	D(.3)
Accuracy:	7.9	17	23	25	41
with oracle:	8.8	18	30	31	54
of Cats:	1	1	1	1	1.04

Method:	D(.1)	D(.03)	D(.01)	D(.003)
Accuracy:	70	126	265	1340
with oracle:	105	230	575	1840
No. of Cats:	1.09	1.14	1.20	1.27

Figure 1: Accuracy of various disambiguation strategies, in number of words per error. On average, the dictionary had 2.2 parts of speech listed per word.

accuracy one can achieve by accepting an additional category every several words indicates that disambiguators can predict when their answers are unreliable.

Readers may worry about correlation resulting from using the same corpus to both learn from and disambiguate. We have run tests by first learning from half of the LOB (600,000 words) and then disambiguating 80,000 words of random text from the other half. The accuracy figures varied by less than 5% from the ones we present, which, given the size of the LOB, is to be expected. We have also applied each disambiguation method to several smaller (13,000 word) sets of sentences which were selected at complete random from throughout the LOB. Accuracy varied both up and down from the figures we present, by up to 20% in terms of words per error, but relative accuracy between methods remained constant.

The fact the Method D with $F = 1$ (with $F = 1$ Method D returns only one category per word) performs as well or even better on the LOB than DeRose's algorithm indicates that, with exceptions, disambiguation has very limited rightward dependence: Method D employs a one category lookahead, whereas DeRose's looks to the end of the sentence. This suggests that Church's strategy of using trigrams instead of digrams may be wasteful. Church manages to achieve results similar or slightly better than DeRose's by defining the probability that a category A appears in a sequence ABC to be the number of times the sequence ABC appears divided by the number of times the sequence BC appears. In a 100 category system, this scheme requires an enormous table of data, which must be culled from tagged text. If the rightward dependence of disambiguation is small, as the data suggests, then the extra effort may be for naught. Based on our results, it is more efficient to use digrams in general and only mark special cases for trigrams, which would reduce space and learning requirements substantially.

Integrating Disambiguator and Parser: As the LOB corpus is pretagged, we could ignore disambiguation problems altogether, but to guarantee that our system can be applied to arbitrary texts, we have integrated a variation of disambiguation Method D with our parser. When a sentence is parsed, the parser is initially passed all categories returned by Method D with $F = .01$. The disambiguator substantially reduces the time and space the parser needs for a given parse, and increases the parser's accuracy. The parser introduces syntactic constraints that perform the remaining disambiguation well.

THE PARSER

Introduction: The LOB corpus contains unedited English, some of which is quite complex and some of which is ungrammatical. No known parser could produce full parses of all the material, and even one powerful enough to do so would undoubtably take an impractical length of time. To facilitate the analysis of the LOB, we have implemented a simple parser which is capable of rapidly parsing simple constructs and of "failing gracefully" in more complicated situations. By trading completeness for accuracy, and by utilizing the statistical disambiguator, the parser can perform rapidly and correctly enough to usefully parse the entire LOB in a few hours. Figure 2 presents a sample parse from the LOB.

The parser employs three methods to build phrases. CFG-like rules are used to recognize lengthy, less structured constructions such as NPs, names, dates, and verb systems. Neighboring phrases can connect to build the higher level binary-branching structure found in English, and single phrases can be projected into new ones. The ability of neighboring phrase pairs to initiate the CFG-like rules permits context-sensitive parsing. And, to increase the efficiency of the parser, an innovative system of deterministically discarding certain phrases is used, called "lowering".

Some Parser Details: Each word in an input sentence is tagged as starting and ending at a specific numerical location. In the sentence "I saw Mary." the parser would insert the locations 0-4, `0 I 1 SAW 2 MARY 3 .`

```
MR MICHAEL FOOT HAS PUT DOWN A RESOLUTION ON THE
SUBJECT AND HE IS TO BE BACKED BY MR WILL
GRIFFITHS , MP FOR MANCHESTER EXCHANGE .

> (IP
   (NP (PROP (N MR) (NAME MICHAEL) (NAME FOOT)))
   (I-BAR (I (HAVE HAS) (RP DOWN))
      (VP (V PUT) (NP (DET A) (N RESOLUTION)))))
> (PP (P ON) (NP (DET THE) (N SUBJECT)))
> (CC AND)
> (IP (NP HE)
   (I-BAR (I)
     (VP (IS IS)
         (I-BAR (I (PP (P BY) (NP (PROP (N MR)
              (NAME WILL) (NAME GRIFFITHS)))))
            (TO TO) (IS BE)) (VP (V BACKED))))))
> (*CMA ",")
> (NP (N MP))
> (PP (P FOR) (NP (PROP (NAME MANCHESTER)
                        (NAME EXCHANGE))))
> (*PER ".")
```

Figure 2: The parse of a sentence taken verbatim from the LOB corpus, printed without features. Notice that the grammar does not attach PP adjuncts.

4. A phrase consists of a category, starting and ending locations, and a collection of feature and tree information. A verb phrase extending from 1 to 3 would print as `[VP 1 3]`. Rules consist of a state name and a location. If a verb phrase recognition rule was firing in location 1, it would get printed as `(VP0 at 1)` where `VP0` is the name of the rule state. Phrases and rules which have yet to be processed are placed on a queue. At parse initialization, phrases are created from each word and its category(ies), and placed on the queue along with an end-of-sentence marker. The parse proceeds by popping the top rule or phrase off the queue and performing actions on it. Figure 3 contains a detailed specification of the parser algorithm, along with parts of a grammar. It should be comprehensible after the following overview and parse example.

When a phrase is popped off the queue, rules are checked to see if they fire on it, a table is examined to see if the phrase automatically projects to another phrase or creates a rule, and neighboring phrases are examined in case they can pair with the popped phrase to either connect into a new phrase or create a rule. Thus the grammar consists of three tables, the "rule-action-table" which specifies what action a rule in a certain state should take if it encounters a phrase with a given category and features; a "single-phrase-action-table" which specifies whether a phrase with a given category and features should project or start a rule; and a "paired-phrase-action-table" which specifies possible actions to take if two certain phrases abut each other.

For a rule to fire on a phrase, the rule must be at the starting position of the phrase. Possible actions that can be taken by the rule are: accepting the phrase (shift the dot in the rule); closing, or creating a phrase from all phrases accepted so far; or both, creating a phrase and continuing the rule to recognize a larger phrase should it exist. Interestingly, when an enqueued phrase is accepted, it is "lowered" to the bottom of the queue, and when a rule closes to create a phrase, all other phrases it may have already created are lowered also.

As phrases are created, a call is made to a set of transducer functions which generate more principled interpretations of the phrases, with appropriate features and tree relations. The representations they build are only for output, and do not affect the parse. An exception is made to allow the functions to project and modify features, which eases handling of subcategorization and agreement. The transducers can be used to generate a constant output syntax as the internal grammar varies, and *vice versa*.

New phrases and rules are placed on the queue only after all actions resulting from a given pop of the queue have been taken. The ordering of their placement has a dramatic effect on how the parse proceeds. By varying the queuing placement and the definition of when a parse is finished, the efficiency and accuracy of the parser can be radically altered. The parser orders these new rules and phrases by placing rules first, and then pushes all of them onto the stack. This means that new rules will always have precedence over newly created phrases, and hence will fire in a successive "rule chain". If all items were eventually popped off the stack, the ordering would be irrelevant. However, since the parse is stopped at the end-of-sentence marker, all phrases which have been "lowered" past the marker are never examined. The part of speech disambiguator can pass in several categories for any one word, which are ordered on the stack by likelihood, most probable first. When any lexical phrase is lowered to the back of the queue (presumably because it was accepted by some rule) all other lexical phrases associated with the same word are also lowered. We have found that this both speeds up parsing and increases accuracy. That this speeds up parsing should be obvious. That it increases accuracy is much less so. Remember that disambiguation Method D is

The Parser Algorithm

To parse a sentences S of length n:

Perform multivalued disambiguation of S.
Create empty queue Q. Place End-of-Sentence marker on Q.
Create new phrases from disambiguator output categories, and place them on Q.
Until Q is empty, or top(Q) = End-of-Sentence marker.
  Let I = pop(Q). Let *new-items* = nil
  If I is phrase [*cat i j*]
    Let *rules* = all rules at location *i*.
    Let *lefts* = all phrases ending at location *i*.
    Let *rights* = all-phrases starting at location *j*.
    Perform **rule-actions**(*rules*,{I})
    Perform **paired-phrase-actions**(*lefts*,{I})
    Perform **paired-phrase-actions**({I},*rights*)
    Perform **single-phrase-actions**(I).
  If I is rule (*state* **at** *i*)
    Let *phrases* = all phrases starting at location *i*.
    Perform **rule-actions**({I},*phrases*).
  Place each item in *new-items* on Q, rules first.
Let $i = 0$. Until $i = n$,
  Output longest phrase [*cat i j*]. Let $i = j$.

To perform rule-actions(*rules*,*phrases*):

For all rules R = (*state* **at** *i*) in *rules*,
  And all phrases P = [*cat+features i j*] in *phrases*,
    If there is an action A in the **rule-action-table** with key (*state, cat+features*),
      If A = (**accept** *new-state*) or (**accept-and-close** *new-state new-cat*),
        Create new rule (*new-state* **at** *j*).
      If A = (**close** *new-cat*) or (**accept-and-close** *new-state new-cat*),
        Let *daughters* = the set of all phrases which have been accepted in the rule chain which led to R, including the phrase P.
        Let l = the leftmost starting location of any phrase in *daughters*. Create new phrase [*new-cat l j*] with daughters *daughters*.
        For all phrases p in *daughters*, perform **lower**(p).
        For all phrases p created (via **accept-and-close**) by the rule chain which led to R, perform **lower**(p).

To perform paired-phrase-actions(*lefts*,*rights*):

For all phrases Pl = [*left-cat+features l i*] in *lefts*,
  And all phrases Pr = [*right-cat+features i r*] in *rights*,
    If there is an action A in the **paired-phrase-action-table** with key (*left-cat+features, right-cat+features*),
      If A = (**connect** *new-cat*),
        Create new phrase [*new-cat l r*] with daughters Pl and Pr.
      If A = (**project** *new-cat*),
        Create new phrase [*new-cat i r*] with daughter Pr.
      If A = (**start-new-rule** *state*),
        Create new rule (*state* **at** *i*).
      Perform **lower**(Pl) and **lower**(Pr).

To perform single-phrase-actions([*cat+features i j*]):

If there is an action A in the **single-phrase-action-table** with key *cat+features*,
  If A = (**project** *new-cat*),
    Create new phrase [*new-cat i j*].
  If A = (**start-rule** *new-state*),
    Create new rule (*state* **at** *i*).

To perform lower(I):

If I is in Q, remove it from Q and reinsert it at end of Q.
If I is a lexical level phrase [*cat i i+1*] created from the disambiguator output categories,
  For all other lexical level phrases p starting at i, perform **lower**(p).

When creating a new rule R:

Add R to list of *new-items*.

When creating a new phrase P = [*cat+features i j*] with daughters D:

Add P to list of *new-items*.
If there is a hook function F in the **hook-function-table** with key *cat+features*, perform $F(P,D)$. Hook functions can add features to P.

A section of a rule-action-table.

Key(State, Cat)	Action
DET0, DET	(accept DET1)
DET1, JJ	(accept DET1)
DET1, N +pl	(close NP)
DET1, N	(accept-and-close DET2 NP)
JJ0, JJ	(accept-and-close JJ0 AP)
VP1, ADV	(accept VP1)

A section of a paired-phrase-action-table.

Key(Cat, Cat)	Action
COMP, S	(connect CP)
NP +poss, NP	(connect NP)
NP, S	(project CP)
NP, VP ext-np +tense expect-nil	(connect S)
NP, CMA*	(start-rule CMA0)
VP expect-pp, PP	(connect VP)

A section of a single-phrase-action-table.

Key(Cat)	Action	Key	Action
DET +pro	(start-rule DET0)	PRO	(project NP)
	(project NP)	V	(start-rule VP3)
N	(start-rule DET1)	IS	(start-rule VP1)
NAME	(start-rule NM1)		(start-rule ISQ1)

A section of a hook-function-table.

Key(Cat)	Hook Function
VP	Get-Subcategorization-Info
S	Check-Agreement
CP	Check-Comp-Structure

Figure 3: A pseudo-code representation of the parser algorithm, omitting implementation details. Included in table form are representative sections from a grammar.

substantially more accurate the DeRose's algorithm only because it can return more than one category per word. One might guess that if the parser were to lower all extra categories on the queue, that nothing would have been gained. But the top-down nature of the parser is sufficient in most cases to "pick out" the correct category from the several available (see Milne 1988 for a detailed exposition of this).

A Parse in Detail: Figure 4 shows a parse of the sentence "The pastry chef placed the pie in the oven." In the figure, items to the left of the vertical line are the phrases and rules popped off the stack. To the right of each item is a list of all new items created as a result of it being popped. At the start of the parse, phrases were created from each word and their corresponding categories, which were correctly (and uniquely) determined by the disambiguator.

The first item is popped off the queue, this being the [DET 0 1] phrase corresponding to the word "the". The single-phrase action table indicates that a DET0 rule should be started at location 0 and immediately fires on "the", which is accepted and the rule (DET1 at 1) is accordingly created and placed on the queue. This rule is then popped off the queue, and accepts the [N 1 2] corresponding to "pastry", also closing and creating the phrase [NP 0 2]. When this phrase is created, all queued phrases which contributed to it are lowered in priority, *i.e.*, "pastry". The rule (DET2 at 2) is created to recognize a possibly longer NP, and is popped off the queue in line 4. Here much the same thing happens as in line 3, except that the [NP 0 2] previously created is lowered as the phrase [NP 0 3] is created. In line 5, the rule chain keeps firing, but there are no phrases starting at location 3 which can be used by the rule state DET2.

The next item on the queue is the newly created [NP 0 3], but it neither fires a rule (which would have to be in location 0), finds any action in the single-phrase table, or pairs with any neighboring phrase to fire an action in the paired-phrase table, so no new phrases or rules are created. Hence, the verb "placed" is popped and the single-phrase table indicates that it should create a rule which then immediately accepts "placed", creating a VP and placing the rule (VP4 at 4) in location 4. The VP is popped off the stack, but not attached to [NP 0 3] to form a sentence, because the paired-phrase table specifies that for those two phrases to connect to become an S, the verb phrase must have the feature (expect . nil), indicating that all of its argument positions have been filled. However when the VP was created, the VP transducer call gave it the feature (expect . NP), indicating that it is lacking an NP argument.

In line 15, such an argument is popped from the stack and pairs with the VP as specified in the paired-phrase table, creating a new phrase, [VP 3 6]. This new VP then pairs with the subject, forming [S 0 6]. In line 18, the preposition "in" is popped, but it does not create any rules or phrases. Only when the NP "the oven" is popped does it pair to create [PP 6 9]. Although it should be attached as an argument

```
0 The 1 pastry 2 chef 3 placed 4 the 5 pie 6 in
7 the 8 oven 9 . 10

1.  Phrase  [DET 0 1]      | (DET0 at 0)
2.  Rule    (DET0 at 0)    | (DET1 at 1)
3.  Rule    (DET1 at 1)    | [NP 0 2] (DET1 at 2)
                             Lowering: [N 1 2]
4.  Rule    (DET2 at 2)    | [NP 0 3] (DET2 at 3)
                             Lowering: [NP 0 2]
                             Lowering: [N 2 3]
5.  Rule    (DET2 at 3)    |
6.  Phrase  [NP 0 3]       |
7.  Phrase  [V 3 4]        | (VP3 at 3)
8.  Rule    (VP3 at 3)     | [VP 3 4] (VP4 at 4)
9.  Rule    (VP4 at 4)     |
10. Phrase  [VP 3 4]       |
11. Phrase  [DET 4 5]      | (DET0 at 4)
12. Phrase  (DET0 at 4)    | (DET1 at 5)
13. Rule    (DET1 at 5)    | [NP 4 6] (DET2 at 6)
                             Lowering: [N 5 6]
14. Rule    (DET2 at 6)    |
15. Phrase  [NP 4 6]       | [VP 3 6]
16. Phrase  [VP 3 6]       | [S 0 6]
17. Phrase  [S 0 6]        |
18. Phrase  [P 6 7]        |
19. Phrase  [DET 7 8]      | (DET0 at 7)
20. Rule    (DET0 at 7)    | (DET1 at 8)
21. Rule    (DET1 at 8)    | [NP 7 9] (DET2 at 9)
                             Lowering: [N 8 9]
22. Rule    (DET2 at 9)    |
23. Phrase  [NP 7 9]       | [PP 6 9]
24. Phrase  [PP 6 9]       |
25. Phrase  [*PER 9 10]    |

> (IP (NP (DET "The") (N "pastry") (N "chef"))
      (I-BAR (I) (VP (V "placed")
                (NP (DET "the") (N "pie")))))
> (PP (P "in") (NP (DET "the") (N "oven")))
> (*PER ".")

Phrases left on Queue: [N 1 2] [N 2 3] [NP 0 2]
                       [N 5 6] [N 8 9]
```

Figure 3: A detailed parse of the sentence "The pastry chef placed the pie in the oven". Dictionary look-up and disambiguation were performed prior to the parse.

to the verb, the subcategorization frames (contained in the **expect** feature of the VP) do not allow for a prepositional phrase argument. After the period is popped in line 25, the end-of-sentence marker is popped and the parse stops. At this time, 5 phrases have been lowered and remain on the queue. To choose which phrases to output, the parser picks the longest phrase starting at location 0, and then the longest phrase starting where the first ended, etc.

The Reasoning behind the Details: The parser has a number of salient features to it, including the combination of top-down and bottom-up methods, the use of transducer functions to create tree structure, and the system of lowering phrases off the queue. Each was necessary to achieve sufficient flexibility and efficiency to parse the LOB corpus.

As we have mentioned, it would be naive of us to believe that we could completely parse the more difficult sentences in the corpus. The next best thing is to recognize smaller phrases in these sentences. This requires some bottom-up capacity, which the parser achieves through the single-phrase and paired-phrase action tables. In order to avoid overgeneration of phrases, the rules (in conjunction with the "lowering" system and method of selecting output phrases) provide a top-down capability which can prevent some valid smaller phrases from being built. Although this can stifle some correct parses[5] we have not found it to do so often.

Readers may notice that the use of special mechanisms to project single phrases and to connect neighboring phrases is unnecessary, since rules could perform the same task. However, since projection and binary attachment are so common, the parser's efficiency is greatly improved by the additional methods.

The choice of transducer functions to create tree structure has roots in our previous experiences with principle-based structures. Modern linguistic theories have shown themselves to be valuable constraint systems when applied to sentence tree-structure, but do not necessarily provide efficient means of initially generating the structure. By using transducers to map between surface structure and more principled trees, we have eliminated much of the computational cost involved in principled representations.

The mechanism of lowering phrases off the stack is also intended to reduce computational cost, by introducing determinism into the parser. The effectiveness of the method can be seen in the tables of Figure 5, which compare the parser's speed with and without lowering.

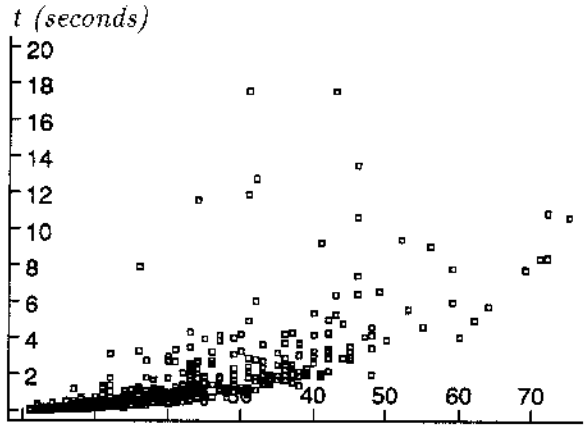
RESULTS

We have used the parser, both with and without the lexical disambiguator, to analyze large portions of the LOB corpus. Our grammar is small; the three primary tables have a total of 134 actions, and the transducer functions are restricted to (outside of building tree structure) projecting categories from daughter phrases upward, checking agreement and case, and dealing with verb subcategorization features. Verb subcategorization information is obtained from the Oxford Advanced Learner's Dictionary of Contemporary English (Hornby *et al* 1973), which often includes unusual verb aspects, and consequently the parser tends to accept too many verb arguments.

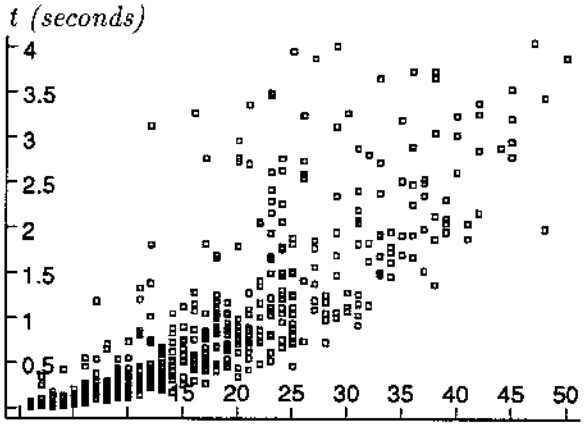
The parser identifies phrase boundaries surprisingly well, and usually builds structures up to the point of major sentence breaks such as commas or conjunctions. Disambiguation failure is almost nonexistent. At the end of this paper is a sequence of parses of sentences from the corpus. The parses illustrate the need for a better subcategorization system and some method for dealing with conjunctions and parentheticals, which tend to break up sentences.

Figure 5 presents some plots of parser speed on a random 624 sentence subset of the LOB, and compares parser performance with and without lowering, and with and without disambiguation. Graphs 1 and 2 (2 is a zoom of 1) illustrate the speed of the parser, and Graph 3 plots the number of phrases the parser returns for a sentence of a given length, which is a measure of how much coverage the grammar has and how much the parser accomplishes. Graph 4 plots the number of phrases the parser builds during an entire parse, a good measure of the work it performs. Not surprisingly, there is a very smooth curve relating the number of phrases built and parse time. Graphs 5 and 6 are included to show the necessity of disambiguation and lowering, and indicate a substantial reduction in speed if either is absent. There is also a substantial reduction in accuracy. In the no disambiguation case, the parser is passed all cate-

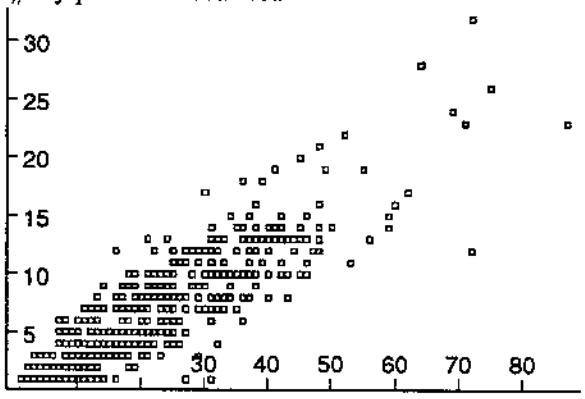
[5] For instance, the parser always generates the longest possible phrase it can from a sequence of words, a heuristic which can in some cases fail. We have found that the only situation in which this heuristic fails regularly is in verb argument attachment; with a more restrictive subcategorization system, it would not be much of a problem.



Graph 1: *# of words in sentence*

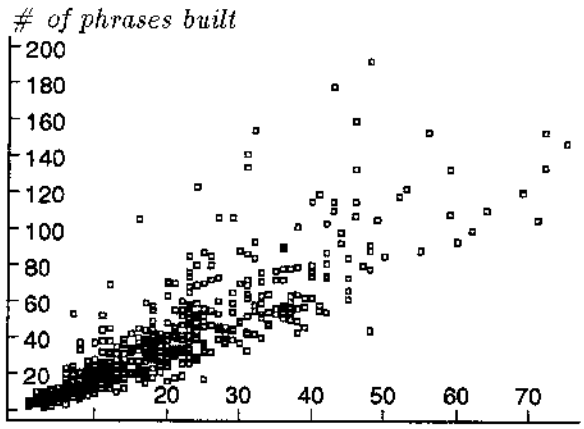


Graph 2: *# of words in sentence*

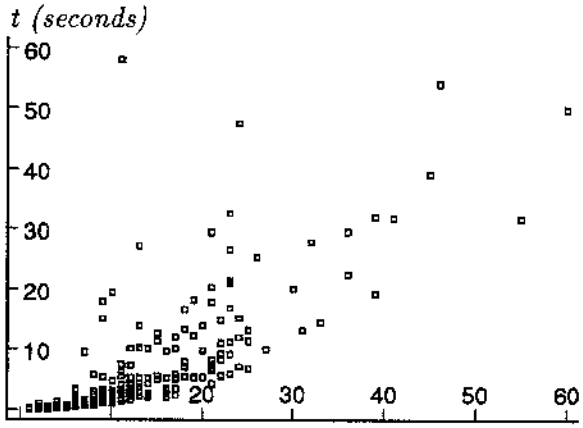


Graph 3: *# of words in sentence*

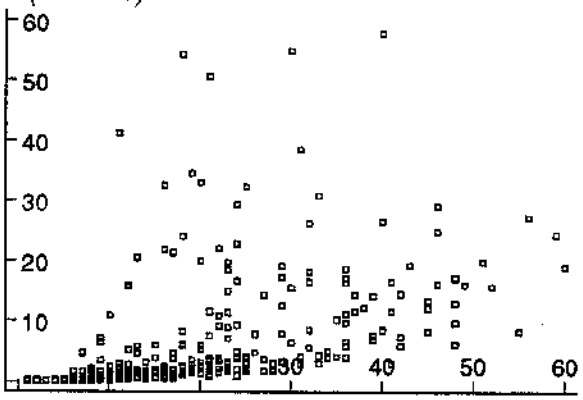
Figure 4: Performance graphs of parser on subset of LOB. See text for explanations.



Graph 4: *# of words in sentence*



Graph 5[No Dis.]: *# of words in sentence*



Graph 6[No Lowering]: *# of words in sentence*

Figure 5: Performance graphs of parser on subset of LOB. See text for explanations.

gories every word can take, in random order.

Parser accuracy is a difficult statistic to measure. We have carefully analyzed the parses assigned to many hundreds of LOB sentences, and are quite pleased with the results. Although there are many sentences where the parser is unable to build substantial structure, it rarely builds incorrect phrases. A pointed exception is the propensity for verbs to take too many arguments. To get a feel for the parser's ac-

curacy, examine the Appendix, which contains unedited parses from the LOB.

BIBLIOGRAPHY

Church, K. W. 1988 A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. *Proceedings of the Second Conference on Applied Natural Language Processing*, 136-143

DeRose, S. J. 1988 Grammatical Category Disambiguation by Statistical Optimization. *Computational Linguistics* 14: 31-39

Oxford Advanced Learner's Dictionary of Contemporary English, eds. Hornby, A.S., and Covie, A. P. (Oxford University Press, 1973)

Milne, R. Lexical Ambiguity Resolution in a Deterministic Parser, in *Lexical Ambiguity Resolution*, ed. by S. Small *et al* (Morgan Kaufmann, 1988)

APPENDIX: Sample Parses

The following are several sentences from the beginning of the LOB, parsed with our system. Because of space considerations, indenting does not necessarily reflect tree structure.

```
A MOVE TO STOP MR GAITSKELL FROM NOMINATING ANY
MORE LABOUR LIFE PEERS IS TO BE MADE AT A
MEETING OF LABOUR MPS TOMORROW .

> (NP (DET A) (N MOVE))
> (I-BAR (I (TO TO)) (VP (V STOP)
       (NP (PROP (N MR) (NAME GAITSKELL)))
       (P FROM)))
> (I-BAR (I) (VP (V NOMINATING)
        (NP (DET ANY) (AP MORE) (N LABOUR)
            (N LIFE) (N PEERS))))
> (I-BAR (I) (VP (IS IS)
     (I-BAR (I (NP (N TOMORROW))
               (TO TO) (IS BE))
            (V MADE) (P AT)
           (NP (NP (DET A) (N MEETING))
               (PP (P OF)
                (NP (N LABOUR) (N MPS))))))))
> (*PER .)

THOUGH THEY MAY GATHER SOME LEFT-WING SUPPORT ,
A LARGE MAJORITY OF LABOUR MPS ARE LIKELY TO
TURN DOWN THE FOOT-GRIFFITHS RESOLUTION .

> (CP (C-BAR (COMP THOUGH))
     (IP (NP THEY)
         (I-BAR (I (MD MAY))
           (VP (V GATHER)
               (NP (DET SOME) (JJ LEFT-WING)
                   (N SUPPORT))))))
> (*CMA ,)
> (IP (NP (NP (DET A) (JJ LARGE) (N MAJORITY))
          (PP (P OF) (NP (N LABOUR) (N MPS))))
      (I-BAR (I) (VP (IS ARE) (AP (JJ LIKELY)))))
> (I-BAR (I (TO TO) (RP DOWN))
           (VP (V TURN)
               (NP (DET THE)
                   (PROP (NAME FOOT-GRIFFITHS))
                   (N RESOLUTION))))
> (*PER .)

MR FOOT'S LINE WILL BE THAT AS LABOUR MPS OPPOSED
THE GOVERNMENT BILL WHICH BROUGHT LIFE PEERS INTO
EXISTENCE , THEY SHOULD NOT NOW PUT FORWARD
NOMINEES .

> (IP (NP (NP (PROP (N MR) (NAME FOOT)))
          (NP (N LINE)))
      (I-BAR (I (MD WILL)) (VP (IS BE) (NP THAT))))
> (CP (C-BAR (COMP AS))
      (IP (NP (N LABOUR) (N MPS))
          (I-BAR (I) (VP (V OPPOSED)
     (NP (NP (DET THE) (N GOVERNMENT) (N BILL))
         (CP (C-BAR (COMP WHICH))
                (IP (NP)
                   (I-BAR (I) (VP (V BROUGHT)
                        (NP (N LIFE) (N PEERS)))))))
                 (P INTO) (NP (N EXISTENCE))))))
> (*CMA ,)
> (IP (NP THEY)
      (I-BAR (I (ADV FORWARD) (MD SHOULD) (XNOT NOT)
                (ADV NOW))
             (VP (V PUT) (NP (N NOMINEES)))))
> (*PER .)

THE TWO RIVAL AFRICAN NATIONALIST PARTIES OF
NORTHERN RHODESIA HAVE AGREED TO GET TOGETHER
TO FACE THE CHALLENGE FROM SIR ROY WELENSKY ,
THE FEDERAL PREMIER .

> (IP (NP (NP (DET THE) (NUM (CD TWO)) (JJ RIVAL)
               (JJ AFRICAN) (JJ NATIONALIST)
               (N PARTIES))
           (PP (P OF) (NP (PROP (NAME NORTHERN)
                                (NAME RHODESIA)))))
      (I-BAR (I (HAVE HAVE)) (VP (V AGREED)
         (I-BAR (I (ADV TOGETHER) (TO TO))
           (VP (V GET)
             (I-BAR (I (TO TO))
                (VP (V FACE)
                    (NP (DET THE) (N CHALLENGE))
                    (P FROM)
                    (NP (NP (PROP (N SIR) (NAME ROY)
                                  (NAME WELENSKY)))
                        (*CMA ,)
                        (NP (DET THE) (JJ FEDERAL)
                            (N+++ PREMIER)))))))))))
> (*PER .)
```