STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U.S.A.

File Copy

A BEST-FIRST PARSER

by

William H. Paxton
Artificial Intelligence Center

Technical Note 92
SRI Project 1526

Proceedings IEEE Speech Symposium, Carnegie-Mellon University, Pittsburgh, Pennsylvania, April 15-19, 1974.

The work reported herein was sponsored by the Advanced Research Projects Agency of the Department of Defense under Contract DAHC04-72-C-0009 with the U. S. Army Research Office.

A BEST-FIRST PARSER*

William H. Paxton
Artificial Intelligence Center
Stanford Research Institute
Menlo Park, California

Abstract

A parser for a speech understanding system is described. The parser uses a best-first strategy in which alternative paths are assigned priorities and paths are suspended as long as there is a higher priority alternative to explore. Discussions are included on the types of steps in a parse, the assignment of priorities, cooperation among competing parses, and experimental results.

Introduction

This paper describes a parser developed at Stanford Research Institute (SRI) as part of ongoing research in speech understanding systems. The parser uses a best-first strategy in searching for an appropriate parse. This strategy and our initial system using it have been described elsewhere.[1] (See also Reference 2 for an overview of the entire system.) To review briefly, each new path from a choice point reached in the grammar is assigned a priority for further processing. The paths are then added to the set of all paths that have been generated but not yet extended during this parse. The system follows the highest priority path from the comprehensive set until its priority drops or it reaches a choice point. This cycle repeats until a parse is found or some resource bound is reached.

The key features of this approach are the assigning of priorities at each step along a path and the suspending of paths when there is an alternative available with a higher priority. This paper describes the types of steps in a parse, the assignment of priorities for the different types of steps, a facility allowing competing parses to cooperate, and an experiment demonstrating the value of the best-first approach.

Types of Steps in a Parse

Figure 1 shows the successful path taken by the system in reaching a correct parse of the test utterance to be considered in this paper: "What little brass parts are in the box?" (The parser's exploration of alternatives to this path is an added complication that will be discussed later.) Each step along the path is given a name in the left column and a step priority in the center column. The right column shows the cumulative priority that equals 1000 times the product of the step priorities up to that point. It is the cumulative priority that is used by the system to choose which path to extend next.

There are four basic types of steps along the path--syntactic, lexical, word verification, and interparse cooperation. Syntactic steps reflect the selection of a particular grammatical construction and are labeled with the name from the grammar for that alternative. For example, the second step is labeled QUEST and corresponds to the path in the grammar for questions as opposed to imperatives or declaratives. Lexical steps involve the choice of a particular word from a predicted class. The rows in the figure labeled with words from the test sentence are the lexical steps. Immediately following each lexical step is a word verification step labeled *VRFY*. During these steps, proposed words are matched against the acoustic data in a manner described in another paper in this symposium.[3] Finally, the steps labeled *FPARSE* reflect interactions among cooperating parses when a constituent has been found.

In general, syntactic and lexical steps cause the cumulative priority to drop, interparse cooperation steps leave the priorities unchanged, and word verification steps potentially cause the priority to rise. The net result is that the system activity tends to focus around words recognized in the utterance without being compelled to explore all possibilities before considering something else. The following sections describe the priority functions for syntactic and lexical steps, the procedures for adjusting priorities after word verification, and the facilities for interparse cooperation.

Syntactic and Lexical Priority Functions

Associated with each syntactic and lexical alternative is a function to compute the priority of that alternative. Functions for different alternatives can call on different sources of knowledge and consider different aspects of the context. This provides a flexible mechanism for integrating a variety of knowledge sources and ensuring that the relevant tests are made at the appropriate places. Our system currently includes priority functions that use such information as semantic features, case grammar, rules of anaphoric reference, and the system's internal model of the world. We plan to increase the use of these sources and to add new ones such as prosodic information[4] and dialog and task models.[5]

As an illustration of lexical priority functions currently in use, we will describe the procedure for nouns. In determining the priority for a noun, two tests are made: one for number agreement and one for semantic agreement. The noun receives low priority if it conflicts in number with its context. For example, a conflict occurs if the context specifically requires a singular noun and the noun in question is plural. Items relevant for number agreement with nouns include articles (e.g., "a" with a singular noun), demonstratives (e.g., "this" with a singular noun; "these" with

*The work reported herein was sponsored by the Advanced Research Projects Agency of the Department of Defense under Contract DAHC04-72-C-0009 with the U.S. Army Research Office.

a plural noun), quantifiers (e.g., "each" with a singular noun; "all" with a plural noun), and verbs (if the noun is to be the subject of the verb). The second step in establishing the lexical priority for nouns is to compare the semantic features of the noun with the semantic features required by the context, and to lower the priority if the features do not match. Items relevant for semantic agreement with nouns include adjectives (if the noun is to be modified by the adjective), prepositions (if the noun is to be the object of the preposition), and verbs (if the noun is to occur in the case frame of the verb)*. Finally, if both number agreement and semantic agreement are satisfactory, the noun is given a high priority.

To illustrate syntactic priority functions, we will describe the procedure that computes the priority for modifiers after a noun phrase. Actually, this procedure produces two priorities--one for accepting the phrase as it stands and one for finding at least one more modifier.

There are three stages in computing the priorities. First, acoustic data are inspected to determine the distance to the end of the utterance; if little of the utterance remains to be parsed, then the priority for further modifiers is lowered. (Eventually more sophisticated prosodic analysis will replace this simple end-of-utterance test.) Second, the case frame is consulted to determine whether there are obligatory cases remaining to be parsed, and, if so, the priority for further modifiers is lowered according to the general heuristic of giving obligatory elements higher priority. Third, for a definite noun phrase (such as ones beginning with "the" or "these"), additional adjustments to the priorities are made depending on the number of referents presupposed by the phrase, the number of referents possible according to the world model, and the number of referents according to the rules of anaphoric reference.

In the above third stage, there are two main cases to consider: first, one in which a particular number of referents is presupposed (as in "these two bolts" or "this box"); second, one for plurals without a specific number. In each case, there are three relevant subcases depending on whether the correspondence between the phrase and items in the world model is consistent and unambiguous, inconsistent, or ambiguous.

In the first case, with a specific number presupposed, if the correspondence is consistent and unambiguous--that is, the presupposed number equals the number of possible referents determined either by the world model or by anaphoric reference--then the phrase can be understood as it stands. Accordingly, the priority for further modifiers is lowered, and the converse priority is raised.

If the correspondence is inconsistent--that is, the presupposed number is greater than the number of possible referents in the world model--then something is probably wrong, and both priorities are lowered.

If the correspondence is ambiguous--that is, the presupposed number is less than the number of possible referents in the world model but the rules of anaphora cannot pick out an appropriate subset--then the priority for further modifiers is raised in expectation that the speaker will provide a disambiguating qualifier.

In the second case, corresponding to plural definite noun phrases without a specific number, if the correspondence is consistent and unambiguous--that is, the rules of anaphora can pick out a group of referents--then the phrase can be understood as it is and the priority for further modifiers is lowered.

If the correspondence is inconsistent--that is, there are fewer than two possible referents according to the world model--then something is probably wrong, and both priorities are lowered.

If the correspondence is ambiguous--that is, there are multiple referents in the world model but anaphora cannot pick out an appropriate subset--then the phrase may be intended to refer to all the possible referents, or a modifier may follow that will narrow the possibilities. Because of this ambiguity, both priorities are set to a moderate level.

While the description given above shows that the priority function for noun phrase modifiers has reached a modest level of complexity, the overall level of sophistication of priority functions in the system is still low. Most of them simply return a fixed value determined by the grammar writer's intuition of the likelihood of the alternative and the cost of exploring it. We do not yet have a full set of priority functions that incorporate a substantial amount and variety of knowledge, but initial experiments suggest that the rudimentary functions we do have are already having a beneficial impact on the performance of the system. Before discussing these results, however, we must consider the procedures for adjusting priorities after a word has been verified.

Verification Priorities

It is a basic principle of our best-first parser that syntactic and lexical priority functions return values less than 1.0 and thus cause the cumulative priority to drop. This principle helps prevent the system from getting trapped exploring false paths, but, if not balanced, it would have the bad effect of penalizing long paths that may be nearing successful completion. A natural choice to offset the lowering of priorities by syntactic and lexical steps is to raise the priority of paths that have led to successful identification of words in the input. Long paths using many words will then be given high priorities reflecting both the likelihood that they are correct and the cost of creating them. When a word is found it will become a focal point of activity as long as likely extensions to that path exist.

The question of how much the priority should be raised remains: Too great an increase will lead to

---

*The case frame includes the subject and any other noun phrases occurring as semantic arguments of the verb. The case grammar in our system is similar to that described by Celce-Murcia.[7]

problems with false paths, while too small an increase will lead to unfocused activity. Our solution has been to raise the priority an amount proportional to the confidence that the word has been correctly recognized. Two factors enter into this computation. the tendency of the particular word to lead to false recognitions and the score between 0 and 1 returned by the word verification routines, indicating how well the word matched. Ideally, the system would know or be able to compute the likelihood of false acceptance for each word in the lexicon. Lacking such thorough data, we have simply divided the words into two classes commonly referred to as function and content words. Function words, such as articles and prepositions, tend to have a high rate of false acceptances because of their small size and lack of stress, so the maximum allowed increase in priority after recognizing a function word is only 25 percent. (Specifically, the step priority for verification of a function word is 1.25 times the word verification score.) Content words, such as nouns and adjectives, are less likely to be falsely accepted, and so the maximum increase allowed for them is 60 percent. The values of 25 percent and 60 percent were arrived at empirically and seem to result in the desired behavior.

A final factor to be considered after word verification is the alignment with the adjacent word. The word verification process returns beginning and ending boundaries as well as a score. Because of the structure of the word verification algorithms, there can be an overlap of the end of one word with the beginning of the next, which results, for example, from a shared segment. More seriously, there is also the possibility of a gap between the words containing a portion of the utterance not recognized as part of either. The system ignores small gaps up to 30 milliseconds in length and gives increasing penalties to larger gaps up to a maximum allowed gap of 200 milliseconds. Similarly, overlaps of up to 100 milliseconds are ignored, but penalties are assigned to larger overlaps up to a maximum of 300 milliseconds. Gaps or overlaps greater than the maximums cause the path to be permanently abandoned.

These procedures for penalizing gaps and overlaps are in the system only as temporary expedients. They will be replaced by phrase verification procedures explicitly designed to deal with interword coarticulation that will check whether the adjacent words really do fit together acoustically and give a reasoned justification for any apparent gap or overlap.

Interparse Cooperation

We have now discussed three of the four types of steps taken by the parser: syntactic, lexical, and word verification steps. The final type reflects a special mechanism for interparse cooperation. There are three ways in which competing parses cooperate in the current system; results of word verification are reused, successful parsing of a constituent leaves a map to guide later attempts, and the effort to find certain basic constituents is shared among all parses. The first two mechanisms have been described elsewhere,[1] so we will limit our discussion to the third.

It is not uncommon for different paths to reach the same point in the utterance looking for the same type of constituent. There is potentially a large amount of duplicated effort as essentially the same alternatives are considered for each of the paths. No duplication of effort will occur, however, if all the paths reaching a particular point in the utterance and looking for a particular constituent share the results of a single search effort rather than carrying out independent searches. An approach to parsing that emphasizes such cooperative searches has been described by Kaplan,[8] and we have implemented a similar scheme in our system.

The set of parses that have reached a certain point looking for a certain type of constituent will be called the consumers for that constituent at that point. For each set of consumers, there is a corresponding set of producers carrying out a search for instances of the constituent. The producers are parses following the different paths through the grammar for the constituent; they are assigned priorities just like other parses being considered by the system. When the first consumer arrives, a new family is created consisting of a single parse starting at the root of the constituent grammar and having a priority equal to that of the consumer. When a producer reaches completion, the resulting constituent is made available to all of the consumers. The constituent is also recorded as a product of the family and offered to any new consumer that arrives.

The calculation of priorities within a family of producers is complicated by the fact that, in general, different consumers have different syntactic restrictions and semantic preferences and these differences should be reflected in the priorities of the producers. Moreover, it must be possible to revise priorities when a new consumer arrives with new demands.

These goals are achieved in the following way. Each producer sets its priority to the maximum over the set of consumers of the priority of the producer with respect to the requirements of that particular consumer (Equation 1).

$$\text{producer priority} = \underset{\text{consumers}}{\text{MAXIMUM}} \left(\text{priority of producer wrt consumer} \right) \quad . \quad (1)$$

The priority with respect to a certain consumer is the priority of that consumer times the product over the steps taken by the producer of the step priority with respect to that consumer (Equation 2).

$$\text{priority of producer wrt consumer} = \text{consumer priority} * \underset{\text{steps}}{\text{PRODUCT}} \left(\text{step priority wrt consumer} \right) . \quad (2)$$

The step priority with respect to a certain consumer is calculated by using data regarding the consumer in making any context-dependent decisions. Since not all of the decisions entering into the calculation of the step priority depend on the consumer, the step priority with respect to a consumer can be viewed as the product of a factor that is the same for all consumers and a second factor that is consumer dependent (Equation 3).

$$\text{step priority wrt consumer} = \text{consumer-independent step priority} * \text{consumer-dependent step priority}$$

This factoring of the step priority in turn allows the overall priority of the producer to be reformulated as the produet of a consumer-independent factor and the maximum of a set of consumer-dependent factors (Equations 4a,b,c).

$$\text{producer priority} = \text{consumer-independent factor} * \underset{\text{consumers}}{\text{MAXIMUM}} \left(\text{consumer-dependent factor}\right). \quad (4a)$$

$$\text{consumer-independent factor} = \underset{\text{steps}}{\text{PRODUCT}} \left(\text{consumer-independent step priority}\right) \quad (4b)$$

$$\text{consumer-dependent factor} = \text{consumer priority} * \underset{\text{steps}}{\text{PRODUCT}} \left(\text{consumer-dependent step priority}\right). \quad (4c)$$

This formulation of the producer priority shows what must be donc when a new consumer arrives. The consumer-dependent factor for the new consumer is calculated, and if it is greater than that for any of the previous consumers, then the producer's priority is raised accordingly. In this way, the work to accommodate a new consumer is essentially redueed to the consumer-dependent computations; all that can be shared is.

An additional consequence of this method of computing producer priorities is that each producer is given the maximum priority its path would have received had the consumers actually carried out independent searches. This means that, although the searches for the various consumers have been merged, no step is taken in the combined search that would not have been taken in at least one of the independent searches. lnterparse cooperation has been achieved without sacrificing the guidance provided by context-dependent priorities.

A final issue is the assignment of priorities to the pairing of a particular consumer with a particular constituent. Such pairings are made in two situations: when a new constituent is produced, it is paired with eaeh consumer; when a new consumer arrives, it is paired with each constituent already produced. The priority for a pairing equals the product of the final consumer-independent factor for the producer of the constituent (Equation 4b) and the final consumer-dependent factor for the consumer of the constituent (Equation 4c). This is exactly the priority that would have resulted had the consumer independently created the constituent, and so the use of the producer-consumer mechanism has no effect on priorities outside the family of producers.

Currently, we use this mechanism only for simple noun phrases without following modifiers, and we only allow lexieal step priorities to he dependent on consumers. The simple noun phrase has relatively straightforward context dependencies, which simplifies the implementation, while at the same time it seems to he a good size unit for such a strategy. It is small enough to be shared often and large enough to offer a substantial savings when it is shared. Moreover, simple noun phrases are fundamental constituents of every utterance. The sentence is basically a collection of such phrases held together by a few verbs and prepositions.

Of course, not all utterances will benefit equally from this facility. For instance, in parsing the test utterance discussed in this paper, there is never more than one consumer per family; hence, no sharing takes place. However, in contrast to this, we have observed parses with a significant amount of sharing. As an example, the parse of the sentence, "Screw a big screw in the handle," has two consumers for each of its noun phrases. The two consumers correspond to the two senses of "screw" known to the system (that of screwing a connector, such as a bolt or screw, into a part or socket and that of screwing a part onto a fixture, such as a faucet). The two senses have different case frames, and thus they are two separate paths through the grammar. Because of this, the family producing the noun phrase "a big screw" has one consumer looking for a phrase that refers to a connector and anothcr consumsr looking for a phrase that refers to any part. Since screws are classified both as parts and as connectors, both consumers accept "a big screw" with a high priority. At this point in the parse, onc sense of the verb predicts the preposition "on" and the other sense predicts the preposition "in." In the actual utterance, the word "in" occurs in a reduced form that leads the word verification routines to give "in" and "on" identical scores and ending positions. As a result, the family of producers for the final noun phrase has two consumers: one looking for a phrase that refers to a part or a soeket. Since handles are parts but not fixtures, the second consumer is given a higher priority and the utterance is correctly understood as "Screw a big screw in the handle."

In this utterance, significant duplication of effort is avoided by use of the producer-consumer facility, whereas in certain other utteranees no savings are possible and use of the facility leads to added overhead. Preliminary results (reported in the Appendix) suggest that the overhead is small compared to the potential savings and that this facility, or something similar to it, will be useful in future systems as a way of combating combinatorial explosion. We are currently considering variations of this approach that will allow sharing with a wider variety of constituents and provide for consumer-dependent step priorities with syntactic and verification steps. The two basic goals remain the same: to make competing parses cooperate, and, at the same time, to exploit the contextual constraints in language as a powerful heuristic.

Experimental Results

We are now ready to examine the overall performance of the parser on the test utterance, "What little brass parts are in the box?" Seven measures of system performance were used. The total number of syntactic and lexical steps taken on all paths indicated the amount of search needed to find a parse. The maximum number of queued paths indicates the storage requirements of the

parse. The number of words predicted by the parser reflects how much "poking around" in the utterance occurred. The number of word verification attempts is the number of words predicted minus the number of times the results of word verification were already available because tbe word had previously been predicted in the same place. Tbis measure indicates both the actual amount of acoustic processing and the savings due to sharing. The number of words found with a score greater than zero gives an indication of the false paths explored. The two final measures are storage and processor resources consumed during the parse. These figures are inflated by the highly interpretive implementation of the parser currently in use, but the relative values are still of interest. The measures are total number of LISP list nodes used and total central processing unit (CPU) time,* excluding initial acoustic processing, such as digital filtering.

For comparison, these measures were also taken for otber configurations of the parser with the same input. First, the parser was given a map leading it directly down the correct path. This provides lower bounds for the various measures and shows how the system would perform if the priority functions were giving perfect guidance. As a second basis for comparison, we modified the system to a depth-first parse, to discard paths with a priority below a threshold but otherwise ignoring priorities. For realistic results, the threshold must be set high enough to eliminate a substantial number of paths but low enough not to block the desired path in more than a small percentage of utterances. In the experiment we tried two levels--a conservative low of 100 and a radical high of 500. The threshold of 500 is probably higher than would actually be employed if a depth-first strategy were used in practice since portions of the grammar are made inaccessible even if the words leading to them are given perfect scores. We feel that the results with this threshold are indicative of the best performance attainable with the depth-first method in combination with fixed threshold pruning. In both of the depth-first parses, all of the system mechanisms for interparse cooperation were operative. The same 52 word vocabulary was used in all the tests. Results of the experiment are given in Table 1.

Using the performance guided by a map as a standard, the best-first parse took roughly three times as long, the depth-first parse with a threshold of 500 took over eight times as long, and the depth-first parse with a threshold of 100 took over forty times as long. With respect to the amount of search (indicated by the number of syntactic and lcxical stcps), best-first used 2.6 times as much as the parse guided by a map, depth-first with a threshold of 500 used 5.4 times as much, and depth-first with a threshold of 100 used 26.6 times as much. The maximum number of queued paths for the depth-first parses is much lower than for best-first since the depth-first explores and eliminates paths rather than suspending them. This is not a significant advantage, however, since the best-first parse did not approach the storage limits. And if storage did become a problem, low priority parses could be pruned. All other measures consistently show the best-first substantially outperforming depth-first with a threshold of 500 and overwhelmingly outperforming depth-first with a threshold of 100.

The results also show the best-first parse doing surprisingly well compared with the parse with a map. This may be because of the tendency of the system to auspend work on a false path before a great deal of effort has been expended. A detailed analysis of the trace of the best-first parse of the test utterance shows the following false paths:

(1) "Are the" was suspended because of a gap penalty and was never resumed.

(2) "How" was suspended because of poor acoustical match and was never resumed.

(3) "What handle" with no further modifiers was killed when the alternative accepting it could not find a following verb. The alternative looking for furtber modifiers was given a lower priority and never resumed.

(4) "What one are" had its priority lowered because the test for number agreement shows that the leading noun phrase eannot be the subject of the sentence. Paths for a noun phrase after "are" were initialized but suspended and were never resumed.

(5) "What little box are the" was suspended because of gap penalty and was never resumed.

(6) "What little handle" was given low priority because of poor match for the last word, and was never resumed.

(7) "What little handles" was treated in the same way as (6) above.

(8) "What little wrench" was treated in the same way as (6) above.

(9) "What little faucet" was treated in the same way as (6) above.

(10) "What little brass wrench" was given a low score because of gap and was never resumed.

(11) "What little brass faucet" was treated in the same way as (10) above.

(12) "What little brass part" was treated in the same way as (3) above.

(13) "What little brass parts are there" was killed because too much of the utterance remained for this to be complete.

(14) "What little brass parts are there on" was given a lower priority because of gap and was never resumed.

*On a PDP-10 running under the TENEX timesharing system. The figures for CPU time are influenced by the amount of timesharing system activity concurrent with tbe parse, since overhead such as core management and scheduling are included. It has been our experience that, even under conditions of light timesharing systcm load, the values for CPU time varied by as much as 10% from one repetition of a parse to the next. For this reason, experimental results for CPU time should only be viewed as rough approximations, accurate to about ± 10%.

It is significant that the majority of the false paths were suspended and never resumed rather than being killed as a result of exploring.

In conclusion, we consider these results as evidence of the value of the best-first approach in parsing. They support our expectations that even rudimentary priority functions can have a significant effect on system performance, and they suggest that more sophisticated priority functions may be able to counteract the problems one would expect to be created by larger vocabularies.

References

1. W. H. Paxton and A. E. Robinson, "A Parser for a Speech Understanding System," Advance Papers of the Conference, Third International Joint Conference on Artificial Intelligence, pp. 216-222, 1973.

2. D. E. Walker, "The SRI Speech Understanding System," paper presented at IEEE Symposium on Speech Recognition, Carnegie-Mellon University, 15-19 April 1974.

3. R. W. Becker and F. Poza, "Acoustic Processing in the SRI Speech Understanding System, paper presented at IEEE Symposium on Speech Recognition, Carnegie-Mellon University, 15-19 April 1974.

4. J. J. Robinson, "Performance Grammars," IEEE Symposium on Speech Recognition, Carnegie-Mellon University, 15-19 April 1974.

5. B. G. Deutsch, "The Structure of Task-Oriented Dialogs," paper presented at IEEE Symposium on Speech Recognition, Carnegie-Mellon University, 15-19 April 1974.

6. R. M. Kaplan, "A Multi-Processing Approach to Natural Language," manuscript, Department of Psychology and Social Relations, Harvard University, Cambridge, Massachusetts, undated.

7. M. Celce-Murcia, "Paradigms for Sentence Recognition," Report AFHRL-TR-30, System Development Corporation, Santa Monica, California (1972).

Appendix: Producer-Consumer Experiment

The experiment reported in this appendix concerns the producer-consumer facility for cooperation among alternative parses that is already described. The utterance "Screw a big screw in the handle" was parsed with four slightly different versions of our parser. All four versions were successful, and all correctly understood the input. Two versions (1 and 3) employed the producer-consumer mechanism, while the other two (2 and 4) did not. Two versions (1 and 2) had the grammar written so that the correct sense of "screw" was considered first, while the other two (3 and 4) considered the "screw-on" sense first. (Version 3 is discussed in the section on interparse cooperation. The priorities for the two senses of the verb were identical; the only change was the order in which they occurred in the grammar. When the "screw-on" sense was considered first, the system followed a false path as far as "Screw a big screw on the" since "in" and "on" were indistinguishable in the utterance. This path went no further, however, because "handle" was given a low priority for semantic reasons. Instead, the system resumed work on the "screw-in" sense and successfully completed the parse. The two versions of the system considering the "screw-in" sense first did not have such problems. They went directly to correct parses without even predicting "on."

Performance measures (described above) for the four parses are given in Table A-1. Both versions considering "screw-in" first did better than either of the ones considering "screw-on" first. Of the two with "in" first, the version without the producer-consumer mechanism searches somewhat more (since there were some opportunities for sharing in the first noun phrase), but it is still marginally faster because of the overhead involved with the producer-consumer mechanism. Of the two with "on" first, the one with the producer-consumer mechanism is clearly superior. Benefiting from sharing in both noun phrases, it requires far fewer syntactic and lexical steps, takes less CPU time, and generally outperforms the parse not using the mechanism. Finally, combining results for the different orderings of the verb senses, the performance with the producer-consumer mechanism is better and has smaller variance than the performance without it. The margin of superiority would have been increased if the vocabulary had been larger and there had been more false paths. This suggests that the producer-consumer approach, or some related mechamism for cooperation among competing parses, will be of increasing value in systems with expanded vocabularies and enriched grammars.

NAME	STEP PRIORITY	CUMULATIVE PRIORITY
"INITSTRING"	.95	950.0
QUEST	.96	912.0
WHQUEST	.96	875.52
WHNG	.96	840.4992
QDET	.96	806.8792
WHAT	.97	782.6729
VRFY	1.4	1095.742
"NUM"	.95	1040.955
ADJNOUN	.95	988.9072
ADJSTRING	.9	890.0164
"ADV"	.95	845.5156
ADJ	.96	811.695
OADJ	.96	779.2272
LITTLE	.98	763.6426
VRFY	1.6	1221.828
ADJSTRING	.85	1038.554
"ADV"	.95	986.6263
ADJ	.96	947.1612
OADJ	.96	909.2748
BRASS	.98	891.0893
VRFY	1.6	1425.743
"ADJSTRING"	.95	1354.456
NOUN	.96	1300.277
PARTS	.95	1235.264
VRFY	1.6	1976.422
FPARSE	1.0	1976.422
"ENDINGS"	.95	1877.601
FULLFORM	.95	1783.721
AUX	.96	1712.372
BEAUX	.95	1626.753
ARE	.98	1594.218
VRFY	1.130357	1802.036
"VG-MODIFIERS"	.95	1711.934
WHSUBJ	.98	1677.695
"THERE"	.95	1593.811
COP	.85	1354.739
PREPCOMPL	.96	1300.549
IN	.97	1261.533
VRFY	1.25	1576.916
ART	.96	1513.84
THE	.98	1483.563
VRFY	1.25	1854.454
"ORD"	.95	1761.731
"NUM"	.95	1673.644
"ADJSTRING"	.95	1589.962
NOUN	.96	1526.364
BOX	.98	1495.836
VRFY	1.6	2393.338
FPARSE	1.0	2393.338
"ENDINGS"	.95	2273.671

FIGURE 1. Path for "What little brass parts are in the box?"

Table 1

EXPERIMENTAL RESULTS

	With Map	Best-First	Depth-First Threshold=500	Depth-First Threshold=100
Syntactic and lexical steps	110*	281	590	2,921
Maximum queued	65	146	15	38
Words predicted	9	121	299	1,396
Words attempted	9	88	165	434
Words found	9†	26	42	92
LISP nodes	12,808	26,814	86,481	437,204
CPU time‡ (min:sec)	1:11	3:31	9:39	47:32

*The successful path includes 41 syntactic and lexical steps. The other 69 steps are the immediate syntactic alternatives along the path.

†The plural ending on "parts" is included as a separate item.

‡Measurements of CPU time are accurate only to about ± 10%.

Table A-1

RESULTS OF PRODUCER-CONSUMER EXPERIMENT

	"Screw-In" First		"Screw-On" First	
	With p-c*	Without p-c*	With p-c*	Without p-c*
Syntactic and lexical steps	132	156	162	249
Maximum queued	72	91	85	131
Words predicted	26	30	36	67
Words attempted	26	25	35	53
Words found	7	7	9	9
LISP nodes	22,613	21,781	24,748	29,498
CPU time (min:sec)	2:06	1:55	2:20	2:41

*Producer-consumer mechanism.