A Practical Comparison of Parsing Strategies

Jonathan Slocum
Siemens Corporation

INTRODUCTION

Although the literature dealing with formal and natural languages abounds with theoretical arguments of worst-case performance by various parsing strategies [e.g., Griffiths & Petrick, 1965; Aho & Ullman, 1972; Graham, Harrison & Ruzzo, 1980], there is little discussion of comparative performance based on actual practice in understanding natural language. Yet important practical considerations do arise when writing programs to understand one aspect or another of natural language utterances. Where, for example, a theorist will characterize a parsing strategy according to its space and/or time requirements in attempting to analyze the worst possible input according to an arbitrary grammar strictly limited in expressive power, the researcher studying Natural Language Processing can be justified in concerning himself more with issues of practical performance in parsing sentences encountered in language as humans actually use it using a grammar expressed in a form convenient to the human linguist who is writing it. Moreover, very occasional poor performance may be quite acceptable, particularly if real-time considerations are not involved, e.g., if a human querant is not waiting for the answer to his question), provided the overall average performance is superior. One example of such a situation is off-line Machine Translation.

This paper has two purposes. One is to report an evaluation of the performance of several parsing strategies in a real-world setting, pointing out practical problems in making the attempt, indicating which of the strategies is superior to the others in which situations, and most of all determining the reasons why the best strategy outclasses its competition in order to stimulate and direct the design of improvements. The other, more important purpose is to assist in establishing such evaluation as a meaningful and valuable enterprise that contributes to the evolution of Natural Language Processing from an art form into an empirical science. That is, our concern for parsing efficiency transcends the issue of mere practicality. At slow-to-average parsing rates, the cost of verifying linguistic theories on a large, general sample of natural language can still be prohibitive. The author's experience in MT has demonstrated the enormous impetus to linguistic theory formulation and refinement that a suitably fast parser will impart: when a linguist can formalize and encode a theory, then within an hour test it on a few thousand words of natural text, he will be able to reject inadequate ideas at a fairly high rate. This argument may even be applied to the production of the semantic theory we all hope for: it is not likely that its early formulations will be adequate, and unless they can be explored inexpensively on significant language samples they may hardly be explored at all, perhaps to the extent that the theory's qualities remain undiscovered. The search for an optimal natural language parsing technique, then, can be seen as the search for an instrument to assist in extending the theoretical frontiers of the science of Natural Language Processing.

Following an outline below of some of the historical circumstances that led the author to design and conduct the parsing experiments, we will detail our experimental setting and approach, present the results, discuss the implications of those results, and conclude with some remarks on what has been learned.

The SRI Connection

At SRI International the author was responsible for the development of the English front-end for the LADDER system [Hendrix et al., 1978]. LADDER was developed as a prototype system for understanding questions posed in English about a naval domain; it translated each English question into one or more relational database queries, prosecuted the queries on a remote computer, and responded with the requested information in a readable format tailored to the characteristics of the answer. The basis for the development of the NLP component of the LADDER system was the LIFER parser, which interpreted sentences according to a 'semantic grammar' [Burton, 1976] whose rules were carefully ordered to produce the most plausible interpretation first.

After more than two years of intensive development, the human costs of extending the coverage began to mount significantly. The semantic grammar interpreted by LIFER had become large and unwieldy. Any change, however small, had the potential to produce "ripple effects" which eroded the integrity of the system. A more linguistically motivated grammar was required. The question arose, "Is LIFER as suited to more traditional grammars as it is to semantic grammars?" At the time, there were available at SRI three production-quality parsers: LIFER; DIAMOND, an implementation of the Cocke-Kasami-Younger parsing algorithm programmed by William Paxton of SRI; and CKY, an implementation of the identical algorithm programmed initially by Prof. Daniel Chester at the University of Texas. In this environment, experiments comparing various aspects of performance were inevitable.

The LRC Connection

In 1979 the author began research in Machine Translation at the Linguistics Research Center of the University of Texas. The LRC environment stimulated the design of a new strategy variation, though in retrospect it is obviously applicable to any parser supporting a facility for testing right-hand-side rule constituents. It also stimulated the production of another parser. (These will be defined and discussed later.) To test the effects of various strategies on the two LRC parsers, an experiment was designed to determine whether they interact with the different parsers and/or each other, whether any gains are offset by introduced overhead, and whether the source and precise effects of any overhead could be identified and explained.

THE SRI EXPERIMENTS

In this section we report the experiments conducted at SRI. First, the parsers and their strategy variations are described and intuitively compared; second, the grammars are described in terms of their purpose and their coverage; third, the sentences employed in the comparisons are discussed with regard to their source and presumed generality; next, the methods of comparing performance are detailed; then the results of the major experiment are presented. Finally, three small follow-up experiments are reported as anecdotal evidence.

The Parsers and Strategies

One of the parsers employed in the SRI experiments was LIFER: a top-down, depth-first parser with automatic back-up [Hendrix, 1977]. LIFER employs special "look down" logic based on the current word in the sentence to eliminate obviously fruitless downward expansion when the current word cannot be accepted as the leftmose element in any expansion of the currently proposed syntactic category [Griffiths and Petrick, 1965] and a "well-formed substring table" [Woods, 1975] to eliminate redundant pursuit of paths after back-up. LIFER supports a traditional style of rule writing where phrase-structure rules are augmented by (LISP) procedures which can reject the application of the rule when proposed by the parser, and which construct an interpretation of the phrase when the rule's application is acceptable. The special user-definable routine responsible for evaluating the S-level rule-body procedures was modified to collect certain statistics but reject an otherwise acceptable interpretation; this forced LIFER into its back-up mode where it sought out an alternate interpretation, which was recorded and rejected in the same fashion. In this way LIFER proceeded to derive all possible interpretations of each sentence according to the grammar. This rejection behavior was not entirely unusual, in that LIFER specifically provides for such an eventuality, and because the grammars themselves were already making use of this facility to reject faulty interpretations. By forcing LIFER to compute all interpretations in this natural manner, it could meaningfully be compared with the other parsers.

The second parser employed in the SRI experiments was DIAMOND: an all-paths bottom-up parser [Paxton, 1977] developed at SRI as an outgrowth of the SRI Speech Understanding Project [Walker, 1978]. The basis of the implementation was the Cocke-Kasami-Younger algorithm [Aho and Ullman, 1972], augmented by an "oracle" [Pratt, 1975] to restrict the number of syntax rules considered. DIAMOND is used during the primarily syntactic, bottom-up phase of analysis; subsequent analysis phases work top-down through the parse tree, computing more detailed semantic information, but these do not involve DIAMOND per se. DIAMOND also supports a style of rules wherein the grammar is augmented by LISP procedures to either reject rule application, or compute an interpretation of the phrase.

The third parser used in the SRI experiments is dubbed CKY. It too is an implementation of the Cocke-Kasami-Younger algorithm. Shortly after the main experiment it was augmented by "top-down filtering," and some small-scale tests were conducted. Like Pratt's oracle, top-down filtering rejects the application of certain rules discovered up by the bottom-up parser specifically, those that a top-down parser would not discover. For example, assuming a grammar for English in a traditional style, and the sentence, "The old man ate fish," an ordinary bottom-up parser will propose three S phrases, one each for: "man ate fish," "old man ate fish," and "The old man ate fish." In isolation each is a possible sentence. But a top-down parser will normally propose only the last string as a sentence, since the left contexts "The old" and "The" prohibit the sentence reading for the remaining strings. Top-down filtering, then, is like running a top-down parser in parallel with a bottom-up parser. The bottom-up parser (being faster at discovering potential rules) proposes the rules, and the top-down parser (being more sensitive to context) passes judgement. Rejects are discarded immediately; those that pass muster are considered further, for example being submitted for feature checking and/or semantic interpretation.

An intuitive prediction of practical performance is a somewhat difficult matter. LIFER, while not originally intended to produce all interpretations, does support a reasonably natural mechanism for forcing that style of analysis. A large amount of effort was invested in making LIFER more and more efficient as the LADDER linguistic component grew and began to consume more space and time. In CPU time its speed was increased by a factor of at least twenty with respect to its original, and rather efficient, implementation. One might therefore expect LIFER to compare favorably with the other parsers, particularly when interpreting the LADDER grammar written with LIFER, and only LIFER, in mind. DIAMOND, while implementeing the very efficient Cocke-Kasami-Younger algorithm and being augmented with an oracle and special programming tricks (e.g., assembly code) intended to enhance its performance, is a rather massive program and might be considered suspect for that reason alone; on the other hand, its predecessor was developed for the purpose of speech understanding, where efficiency issues predominate, and this strongly argues for good performance expectations. Chester's implementation of the Cocke-Kasami-Younger algorithm represents the opposite extreme of startling simplicity. His central algorithm is expressed in a dozen lines of LISP code and requires little else in a basic implementation. Expectations here might be bi-modal: it should either perform well due to its concise nature, or poorly due to the lack of any efficiency aids. There is one further consideration of merit: that of inter-programmer variability. Both LIFER and Chester's parser were rewritten for increased efficiency by the author; DIAMOND was used without modification. Thus differences between DIAMOND and the others might be due to different programming styles -- indeed, between DIAMOND and CKY this represents the only difference aside from the oracle --while differences between LIFER and CKY should reflect real performance distinctions because the same programmer (re)implemented them both.

The Grammars

The "semantic grammar" employed in the SRI experiments had been developed for the specific purpose of answering questions posed in English about the domain of ships at sea [Sacerdoti, 1977]. There was no pretense of its being a general grammar of English; nor was it adept at interpreting questions posed by users unfamiliar with the naval domain. That is, the grammar was attuned to questions posed by knowledgeable users, answerable from the available database. The syntactic categories were labelled with semantically meaningful names like <SHIP>, <ARRIVE>, <PORT>, and the like, and the words and phrases encompassed by such categories were restricted in the obvious fashion. Its adequacy of coverage is suggested by the success of LADDER as a demonstration vehicle for natural language access to databases [Hendrix et al., 1978].

The linguistic grammar employed in the SRI experiments came from an entirely different project concerned with discourse understanding [Grosz, 1978]. In the project scenario a human apprentice technician consults with a computer which is expert at the disassembly, repair, and reassembly of mechanical devices such as a pump. The computer guides the apprentice through the task, issuing instructions and explanations at whatever levels of detail are required; it may answer questions, describe appropriate tools for specific tasks, etc. The grammar used to interpret these interactions was strongly linguistically motivated [Robinson, 1980]. Developed in a domain primarily composed of declarative and imperative sentences, its generality is suggested by the short time (a few weeks) required to extend its coverage to the wide range of questions encountered in the LADDER domain.
In order to prime the various parsers with the different frammars, four programs were written to transform each grammar into the formalism expected by the two parsers for which it was not originally writtten. Specifically, the linguistic grammar had to be reformatted for input to LIFER and CKY; the semantic grammar, for input to CKY and DIAMOND. Once each of six systems was loaded with one parser and one grammar, the stage would be set for the experiment.

The Sentences

Since LADDER's semantic grammar had been written for sentences in a limited domain, and was not intended for general English, it was not possible to test that grammar on any corpus outside of its domain. Therefore, all sentences in the experiment were drawn from the LADDER benchmark: the broad collection of queries designed to verify the overall integrity of the LADDER system after extensions had been incorporated. These sentences, almost all of them questions, had been carefully selected to exercise most of LADDER's linguistic and database capabilities. Each of the six systems, then, was to be applied to the analysis of the same 249 benchmark sentences; these ranged in length from 2 to 23 words and averaged 7.82 words.

Methods of Comparison

Software instrumentation was used to measure the following: the CPU time; the number of phrases (instantiations of grammar rules) proposed by the parser; the number of these rejected by the rule-body procedures in the usual fashion; and the storage requirements (number of CONSes) of the analysis attempt. Each of these was recorded separately for sentences which were parsed vs. not parsed, and in the former case the number of interpretations was recorded as well. For the experiment, the database access code was short-circuited; thus only analysis, not question answering, was performed. The collected data was categorized by sentence length and treatment (parser and grammar) for analysis purposes.

Summary of the First Experiment

The first experiment involved the production of six different instrumented systems -- three parsers, each with two grammars -- and six test runs on the identical set of 249 entences comprising the LADDER benchmark. The benchmark, established quite independently of the experiment, had as its raison d'etre the vigorous exercise of the LADDER system for the purpose of validationg its integrity. The sentences contained therein were intended to constitute a representative sample of what might be expected in that domain. The experiment was conducted on a DEC KL-10; the systems were run separately, during low-load conditions in order to minimize competition with other programs which could confound the results.

The Experimental Results

As it turned out, the large internal grammar storage overhead of the DIAMOND parser prohibited its being loaded with the LADDER semantic grammar: the available memory space was exhausted before the grammar could be fully defined. Although eventually a method was worked out whereby the semantic grammar could be loaded into DIAMOND, the resulting system was not tested due to its non-standard mode of operation, and because the working space left over for parsing was minimal. Therefore, the results and discussion will include data for only five combinations of parser and grammar.

Linguistic Grammar

In terms of the number of grammar rules found applicable by the parsers, DIAMOND instantiated the fewest (averaging 58 phrases per sentence); CKY, the most (121); and LIFER fell in between (107). LIFER makes copious use of CONS cells for internal processing purposes, and thus required the most storage (averaging 5294 CONSes per parsed sentence); DIAMOND required the least (1107); CKY fell in between (1628). But in terms of parse time, CKY was by far the best (averaging .386 seconds per sentence, exclusive of garbage collection); DIAMOND was next best (.976); and LIFER was worst (2.22). The total run time on the SRI-KL machine for the batch jobs interpreting the linguistic grammar (i.e., 'pure' parse time plus all overhead charges such as garbage collection, I/O, swapping and paging) was 12 minutes, 50 seconds for LIFER, 7 minutes, 13 seconds for DIAMOND, and 3 minutes 15 seconds for CKY. The surprising indication here is that, even though CKY proposed more phrases than its competition, and used more storage than DIAMOND (though less than LIFER), it is the fastest parser. This is true whether considering successful or unsuccessful analysis attempts, using the linguistic grammar.

Semantic Grammar

We will now consider the corresponding data for CKY vs. LIFER using the semantic grammar (remembering that DIAMOND was not testable in this configuration). In terms of the number of phrases per parsed sentence, CKY averaged five times as many as LIFER (151 compared to 29). In terms of storage requirements CKY was better (averaging 1552 CONSes per sentence) but LIFER was only slightly worse (1498). But in CPU time, discounting garbage collection, CKY was again significantly faster than LIFER (averaging .286 seconds per sentence compared to .635). The total run time on the SRI-KL machine for the batch jobs interpreting the semantic grammar (i.e., "pure" parse time plus all overhead charges such as garbage collections, I/O, swapping and paging) was 5 minutes, 10 seconds for LIFER, and 2 minutes, 56 seconds for CKY. As with the linguistic grammar, CKY was significantly more efficient, whether considering successful or unsuccessful analysis attempts, while using the same grammar and analyzing the same sentences.

Three Follow-up Experiments

Three follow-up mini-experiments were conducted. The number of sentences was relatively small (a few dozen), and the results were not permanently recorded, thus they are reported here as anecdotal evidence. In the first, CKY and LIFER were compared in their natural modes of operation -- that is, with CKY finding all interpretations and LIFER finding the first -- using both grammars but just a few sentences. This was in response to the hypothesis that forcing LIFER to derive all interpretations is necessarily unfair. The results showed that CKY derived all interpretations of the sentences in slightly less time than LIFER found its first.
The discovery that DIAMOND appeared to be considerably less efficient than CKY was quite surprising. Implementing the same algorithm, but augmented with the phrase-limiting "oracle" and special assembly code for efficiency, one might expect DIAMOND to be faster than CKY. A second mini-experiment was conducted to test the most likely explanation -- that the overhead of DIAMOND's oracle might be greater than the savings it produced. The results clearly indicated that DIAMOND was yet slower without its oracle.
The question then arose as to whether CKY might be yet faster if it too were similarly augmented. A top-down filter modification was soon implemented and another small experiment was conducted. Paradoxically, the effect of filtering in this instance was to degrade performance. The overhead incurred was greater than the observed savings. This remained a puzzlement, and eventually helped to inspire the LRC experiment.

THE LRC EXPERIMENT

In this section we discuss the experiment conducted at the Linguistics Research Center. First, the parsers and their strategy variations are described and intuitively compared; second, the grammar is described in terms of its purpose and its coverage; third, the sentences employed in the comparisons are discussed with regard to their source and presumed generality; next, the methods of comparing performance are discussed; finally, the results are presented.

The Parsers and Strategies

One of the parsers employed in the LRC experiment was the CKY parser. The other parser employed in the LRC experiment is a left-corner parser, inspired again by Chester [1980] but programmed from scratch by the author. Unlike a Cocke-Kasami-Younger parser, which indexes a syntax rule by its right-most constituent, a left-corner parser indexes a syntax rule by the left-most constituent in its right-hand side. Once the parser has found an instance of the left-corner constituent, the remainder of the rule can be used to predict what may come next. When augmented by top-down filtering, this parser strongly resembles the Earley algorithm [Earley, 1970].

Since the small-scale experiments with top-down filtering at SRI had revealed conflicting results with respect to DIAMOND and CKY, and since the author's intuition continued to argue for increased efficiency in conjunction with this strategy despite the empirical evidence to the contrary, it was decided to compare the performance of both parsers with and without top-down filtering in a larger, more carefully controlled experiment. Another strategy variation was engendered during the course of work at the LRC, based on the style of grammar rules written by the linguistic staff. This strategy, called "early constituent tests," is intended to take advantage of the extent of testing of individual constituents in the right-hand-sides of the rules. Normally a parser searches its chart for contiguous phrases in order as specified by the right-hand-side of a rule, then evaluates the rule-body procedures which might reject the application due to a deficiency in one of the r-h-s constituent phrases; the early constituent test strategy calls for the parser to evaluate that portion of the rule-body procedure which tests the first constituent, as soon as it is discovered, to determine if it is acceptable; if so, the parser may proceed to search for the next constituent and similarly evaluate its test. In addition to the potential savings due to earlier rule rejection, another potential benefit arises from ATN-style sharing of individual constituent tests among such rules as pose the same requirements on the same initial sequence of r-h-s constituents. Thus one test could reject many apparently applicable rules at once, early in the search -- a large potential savings when compared with the alternative of discovering all constituents of each rule and separately applying the rule-body procedures, each of which might reject (the same constituent) for the same reason. On the other hand, the overhead of invoking the extra constituent tests and saving the results for eventual passage to the remainder of the rule-body procedure will to some extent offset the gains.

It is commonly considered that the Cocke-Kasami-Younger algorithm is generally superior to the left-corner algorithm in practical application; it is also thought that top-filtering is beneficial. But in addition to intuitions about the performance of the parsers and strategy variations individually, there is the issue of possible interactions between them. Since a significant portion of the sentence analysis effort may be invested in evaluating the rule-body procedures, the author's intuition argued that the best combination could be the left-corner parser augmented by early constituent tests and top-down filtering -- which would seem to maximally reduce the number of such procedures evaluated.

The Grammar

The grammar employed during the LRC experiment was the German analysis grammar being developed at the LRC for use in Machine Translation [Lehmann et al., 1981]. Under development for about two years up to the time of the experiment, it had been tested on several moderately large technical corpora [Slocum, 1980] totalling about 23,000 words. Although by no means a complete grammar, it was able to account for between 60 and 90 percent of the sentences in the various texts, depending on the incidence of problems such as highly unusual constructs, outright errors, the degree of complexity in syntax and semantics, and on whether the tests were conducted with or without prior experience with the text. The broad range of linguistic phenomena represented by this material far outstrips that encountered in most NLP systems to date. Given the amount of text described by the LRC German grammar, it may be presumed to operate in a fashion reasonably representative of the general grammar for German yet to be written.

The Sentences

The sentences employed in the LRC experiment were extracted from three different technical texts on which the LRC MT system had been previously tested. Certain grammar and dictionary extensions based on those tests, however, had not yet been incorporated; thus it was known in advance that a significant portion of the sentences might not be analyzed. Three sentences of each length were randomly extracted from each text, where possible; not all sentence lengths were sufficiently represented to allow this in all cases. The 262 sentences ranged in length from 1 to 39 words, averaging 15.6 words each -- twice as long as the sentences employed in the SRI experiments.

Methods of Comparison

The LRC experiment was intended to reveal more of the underlying reasons for differential parser performance, including strategy interactions; thus it was necessary to instrument the systems much more thoroughly. Data was gathered for 35 variables measuring various aspects of behavior, including general information (13 variables), search space (8 variables), processing time (7 variables), and memory requirements (7 variables). One of the simpler methods measured the amount of time devoted to storage management (garbage collection in INTERLISP) in order to determine a "fair" measure of CPU time by pro-rating the storage management time according to storage used (CONSes executed); simply crediting garbage collect time to the analysis of the sentence immediately at hand, or alternately neglecting it entirely, would not represent a fair distribution of costs. More difficult was the problem of measuring search space. It was not felt that an average branching factor computed for the static grammar would be representative of the search space encountered during the dynamic analysis of sentences. An effort was therefore made to measure the search space actually encountered by the parsers, differentiated into grammar vs. chart search; in the former instance, a further differentiation was based on whether the grammar space was being considered from the bottom-up (discovery) vs. top-down (filter) perspective. Moreover, the time and space involved in analyzing words and idioms and operating the rule-body procedures was separately measured in order to determine the computational effort expended by the parser proper. For the experiment, the translation process was short-circuited; thus only analysis, not transfer and synthesis, was performed.

Summary of the LRC Experiment

The LRC experiment involved the production of eight different instrumented systems -- two parsers (left-corner and Cocke-Kasami-Younger), each with all four combinations of two independent strategy variations (top-down filtering and early constituent tests)-- and eight test runs on the identical set of 262 sentences selected pseudo-randomly from three technical texts supplied by the MT project sponsor. The sentences contained therein may reasonably be expected to constitute a nearly-representative sample of text in that domain, and presumably constitute a somewhat less-representative (but by no means trivial) sample of the types of syntactic structures encountered in more general German text.

The usual (i.e., complete) analysis procedures for the purpose of subsequent translation were in effect, which includes production of a full syntactic and semantic analysis via phrase-structure rules, feature tests and operations, transformations, and case frames. It was known in advance that not all constructions would be handled by the grammar; further, that for some sentences some or all of the parsers would exhaust the available space before achieving an analysis. The latter problem in particular would indicate differential performance characteristics when working with limited memory. One of the parsers, the version of the CKY parser lacking both top-down filtering and early constituent tests, is essentially identical to the CKY parser employed in the SRI experiments. The experiment was conducted on a DEC 2060; the systems were run separately, late at night in order to minimize competition with other programs which could confound the results.

The Experimental Results

The various parser and strategy combinations were slightly unequal in their ability to analyze (or, alternately, demonstrate the ungrammaticality of) sentences within the available space. Of the three strategy choices (parser, filtering, constituent tests), filtering constituted the most effective discriminant: the four systems with top-down filtering were 4% more likely to find an interpretation than the four without; but most of this difference occurred within the systems employing the left-corner parser, where the likelihood was 10% greater. The likelihood of deriving an interpretation at all is a matter that must be considered when contemplating application on machines with relatively limited address space. The summaries below, however, have been balanced to reflect a situation in which all systems have sufficient space to conclude the analysis effort, so that the comparisons may be drawn on an equal basis. Not surprisingly, the data reveal differences between single strategies and between joint strategies, but the differences are sometimes much larger than one might suppose. Top-down filtering overall reduced the number of phrases by 35%, but when combined with CKY without early constituent tests the difference increased to 46%. In the latter case, top-down filtering increased the overall search space by a factor of 46-- to well over 300,000 nodes per sentence. For the Left-Corner Parser without early constituent tests, the growth rate is much milder -- an increase in search space of less than a factor of 6 for a 42% reduction in the number of phrases -- but the original (unfiltered)search space was over 3 times as large as that of CKY. CKY overall required 84% fewer CONSes than did LCP (considering the parsers alone); for one matched pair of joint strategies, pure LCP required over twice as much storage as pure CKY.

Evaluating the parsers and strategies via CPU time is a tricky business, for one must define and justify what is to be included. A common practice is to exclude almost everything (e.g., the time spent in storage management, paging, evaluating rule-body procedures, building parse trees, etc.). One commonly employed ideal metric is to count the number of trips through the main parser loops. We argue that such practices are indefensible. For instance, the "pure parse times" measured in this experiment differ by a factor of 3.45 in the worst case, but overall run times vary by 46% at most. But the important point is that if one chose the "best" parser on the basis of pure parse time measured in this experiment, one would have the fourth-best overall system; to choose the best overall system, one must settle for the "sixth-best" parser! Employing the loop-counter metric, we can indeed get a perfect prediction of rank-order via pure parse time based on the inner-loop counters; what is more, a formula can be worked out to predict the observed pure parse times given the three loop counters. But such predictions have already been shown to be useless (or worse) in predicting total program runtime. Thus in measuring performance we prefer to include everything one actually pays for in the real computing world: Paging, storage management, building interpretations, etc., as well as parse time.

In terms of overall performance, then, top-down filtering in general reduced analysis times by 17% (though it increased pure parse times by 58%); LCP was 7% less time-consuming than CKY; and early constituent tests lost by 15% compared to not performing the tests early. As one would expect, the joint strategy LCP with top-down filtering [ON] and Late (i.e. not Early) Constituent Tests [LCT] ranked first among the eight systems. However, due to beneficial interactions the joint strategy [LCP ON ECT] (which on intuitive grounds we predicted would be most efficient) came in a close second; [CKY ON LCT] came in third. The remainder ranked as follows: [CKY OFF LCT], [LCP OFF LCT], [CKY ON ECT], [CKY OFF ECT], [LCP OFF ECT]. Thus we see that beneficial interaction with ECT is restricted to [LCP ON].

Two interesting findings are related to sentence length. One, average parse times (however measured) do not exhibit cubic or even polynomial behavior, but instead appear linear. Two, the benefits of top-down filtering are dependent on sentence length; in fact, filtering is detrimental for shorter sentences. Averaging over all other strategies, the break-even point for top-down filtering occurs at about 7 words. (Filtering always increases pure parse time, PPT, because the parser sees it as pure overhead. The benefits are only observable in overall system performance, due primarily to a significant reduction in the time/space spent evaluating rule-body procedures.) With respect to particular strategy combinations, the break-even point comes at about 10 words for [LCP LCT], 6 words for [CKY ECT], 6 words for [LCP LCT], and 7 words for [LCP ECT]. The reason for this length dependency becomes rather obvious in retrospect, and suggests why top-down filtering in the SRI follow-up experiment was detrimental: the test sentences were probably too short.

DISCUSSION

The immediate practical purpose of the SRI experiments was not to stimulate a parser-writing contest, but to determine the comparative merits of parsers in actual use with the particular aim of extablishing a rational basis for choosing one to become the core of a future NLP system. The aim of the LRC experiment was to discover which implementation details are responsible for the observed performance with an eye toward both suggesting and directing future improvements.

The SRI Parsers

The question of relative efficiency was answered decisively. It would seem that the CKY parser performs better than LIFER due to its much greater speed at finding applicable rules, with either the semantic or the linguistic grammar. CKY certainly performs better than DIAMOND for this reason, presumably due to programmar differences since the algorithms are the same. The question of efficiency gains due to top-down filtering remained open since it enhanced one implementation but degraded another. Unfortunately, there is nothing in the data which gets at the underlying reasons for the efficiency of the CKY parser.

The LRC Parsers

Predictions of performance with respect to all eight systems are identical, if based on their theoretically equivalent search space. The data, however, display some rather dramatic practical differences in search space. LCP's chart search space, for example, is some 25 times that of CKY; CKY's filter search space is almost 45% greater than that of LCP. Top-down filtering increases search space, hence compute time, in idealized models which bother to take it into account. Even in this experiment, the observed slight reduction in chart and grammar search space due to top-down filtering is offset by its enormous search space overhead of over 100,000 nodes for LCP, and over 300,000 nodes for [CKY LCT], for the average sentence. But the overhead is more than made up in practice by the advantages of greater storage efficiency and particularly the reduced rule-body procedure "overhead." The filter search space with late column tests is three times that with early column tests, but again other factors combine to reverse the advantage.

The overhead for filtering in LCP is less than that in CKY. This situation is due to the fact that LCP maintains a natural left-right ordering of the rule constituents in its internal representation, whereas CKY does not and must therefore compute it at run time. (The actual truth is slightly more complicated because CKY stores the grammar in both forms, but this caricature illustrates the effect of the differences.) This is balanced somewhat by LCP's greatly increased chart search space; by way of caricature again, LCP is doing some things with its chart that CKY does with its filter. (That is, LCP performs some "filtering" as a natural consequence of its algorithm.) The large variations in the search space data would lead one to expect large differences in performance. This turns out not to be the case, at least not in overall performance.

CONCLUSIONS

We have seen that theoretical arguments can be quite inaccurate in their predictions when one makes the transition from a worst-case model to an actual, real-world situation. "Order n-cubed" performance does not appear to be realized in practice; what is more, the oft-neglected constants of theoretical calculations seem to exert a dominating effect in practical situations. Arguments about relative efficiencies of parsing methods based on idealized models such as inner-loop counters similarly fail to account for relative efficiencies observed in practice. In order to meaningfully describe performance, one must take into account the complete operational context of the Natural Language Processing system, particularly the expenses encountered in storage management and applying rule-body procedures.

BIBLIOGRAPHY

Aho, A. V., and J. D. Ullman. The Theory of Parsing, Translation, and Compiling, Vol. I. Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

Burton, R. R., "Semantic Grammar: An engineering technique for constructing natural language understanding systems," BBN Report 3453, Bolt, Beranek, and Newman, Inc., Cambridge, Mass., Dec. 1976.

Chester, D., "A Parsing Algorithm that Extends Phrases," AJCL 6 (2), April-June 1980, pp.87-96.

Earley, J., "An Efficient Context-free Parsing Algorithm," CACM 13 (2), Feb. 1970, pp. 94-102.

Graham, S. L., M. A. Harrison, and W. L. Ruzzo, "An Improved Context-Free Recognizer," ACM Transactions on Programming Languages and Systems, 2 (3), July 1980, pp. 415-462.

Griffiths, T. V., and S. R. Petrick, "On the Relative Efficiencies of Context-free Grammar Recognizers," CACM 8 (5), May 1965, pp. 289-300.

Grosz, B. J., "Focusing in Dialog," Proceedings of Theoretical Issues in Natural Language Processing-2: An Interdisciplinary Workshop, University of Illinois at Urbana-Champaign, 25-27 July 1978.

Hendrix, G. G., "Human Engineering for Applied Natural Language Processing," Proceedings of the 5th International Conference on Artificial Intelligence, Cambridge, Mass., Aug. 1977.

Hendrix, G. G., E. D. Sacerdoti, D. Sagalowicz, and J. Slocum, "Developing a Natural Language Interface to Complex Data," ACM Transactions on Database Systems, 3 (2), June 1978, pp. 105-147.

Lehmann, W. P., W. S. Bennett, J. Slocum, et al., "The METAL System," Final Technical Report RADC-TR-80-374. Rome Air Development Center, Griffiss AFB, New York, Jan. 1981. Available from NTIS.

Paxton, W. H., "A Framework for Speech Understanding," Tech. Note 142, AI Center, SRI International, Menlo Park, Calif., June 1977.

Pratt, V. R., "LINGOL: A progress report," Proceedings of the Fourth International Joint Conference on Artificial Intelligence, Tbilisi, Georgia, USSR, 3-8 Sept. 1975, pp. 422-428.

Robinson, J J., "DIAGRAM: A grammar for dialogues," Tech. Note 205, AI Center, SRI International, Menlo Park, Calif., Feb. 1980.

Sacerdoti, E. D., "Language Access to Distributed Data with Error Recovery," Proceedings of the Fifth International Joint Conference on Artificial Intelligience, Cambridge, Mass., Aug. 1977.

Slocum, J., An Experiment in Machine Translation," Proceedings of the 18th Annual Meeting of the Association for Computational Linguistics, Philadelphia, 19-22 June 1980, pp. 163-167.

Walker, D. E. (ed.). Understanding Spoken Language. North-Holland, New York, 1978.

Woods, W. A., "Syntax, Semantics, and Speech," BBN Report 3067, Bolt, Beranek, and Newman, Inc., Cambridge, Mass., Apr. 1975.