Efficient Bottom-Up Parsing

Robert Moore and John Dowding

SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

Abstract

This paper describes a series of experiments aimed at producing a bottom-up parser that will produce partial parses suitable for use in robust interpretation and still be reasonably efficient. In the course of these experiments, we improved parse times by a factor of 18 over our first attempt, ending with a system that was twice as fast as our previous parser, which relied on strong top-down constraints. The major algorithmic variations we tried are described along with the corresponding performance results.

Introduction

Elsewhere [1] we describe a change in our approach to NL processing to allow for more robust methods of interpretation. One consequence of this change is that it requires a different type of parsing algorithm from the one we have been using. In our previous SLS work, we have used a shift-reduce left-corner parser incorporating strong top-down constraints derived from the left context, to limit the structures built by the parser [2]. With this parser, no structure is built unless it can combine with structures already built to contribute to an analysis of the input as a single complete utterance. If we want to find grammatical fragments of the input that may be of use in robust interpretation, however, such strong use of top-down constraints is not appropriate.

To address this issue, we have built and measured the performance of a number of bottom-up parsers. These parsers use the same unification grammar as our shift-reduce parser, but they do not impose the strong top-down constraints of the original. These experimental parsers fall into two groups: purely bottom-up parsers and bottom-up parsers that use limited top-down constraints. The experiments were performed using a fixed grammar and lexicon for the Air Travel Information System (ATIS) domain, and an arbitrarily selected test corpus of 120 ATIS0 training sentences. The test grammar could produce complete parses for 79 of these 120 sentences.

Pure Bottom-Up Parsing

The first parser we implemented was a straightforward "naive" implementation of the CKY algorithm [3, 4] adapted to unification grammar. In this algorithm, a "chart" is maintained that contains records, or "edges," for each type of linguistic category that has been found between given start and end positions in a sentence. In context-free parsing, these categories are simply the nonterminal symbols of the grammar. In a unification grammar, they are complex structures that assign values to particular features of a more general category type.

Our naive algorithm simply seeds the chart with edges for each possible category for all the words in the sentence, and then works left to right constructing additional edges bottom-up. Each time an edge is added to the chart, the grammar is searched for rules whose last category on the right-hand side matches the edge just added to the chart, and the chart is scanned back to the left for a contiguous sequence of edges that match the remaining categories on the right-hand side of the rule. If these are found, then an edge for the category on the left-hand side of the rule is added to the chart, spanning the segment of the input covered by the sequence of edges that matched the right-hand side of the rule.

When measured with our test grammar and test corpus, our implementation of this algorithm is almost nine times slower than our original shift-reduce parser. We conjectured that one significant problem was the unconstrained hypothesization of empty categories or "gaps." Our grammar, like many others, allows certain linguistic phrase types to be realized as the empty string in order to simplify the overall structure of the grammar. For example, "What cities does American fly to from Boston?" is analyzed as having an empty noun phrase between "to" and "from," so that most of the analysis can be carried out using the same rules that are used to analyze such sentences as "Does American fly to Dallas from Boston?" Because empty categories are not directly indicated in the word string, our naive bottom-up parser must hypothesize every possible empty category at every point in the input.

To address this point, we applied a well-known transformation to the grammar to eliminate empty categories by adding additional rules. For each type of empty cat-

egory, we found every case where it would unify with a category on the right-hand side of a rule, performed the unification, and deleted the unified empty category from the rule. For example, if B can be an empty category then from $A \rightarrow BC$ we would derive the rule $A \rightarrow C$, taking into account the results of unification. When all such derived rules are added to the grammar, all the empty categories can be eliminated.

Performing this transformation both reduced the number of edges being generated and speeded up parsing, but only by about 20 percent in each case. We observed that the elimination of empty categories had resulted in a grammar with many more unit production rules than the original grammar; that is, rules of the form $A \rightarrow B$. This occurred because of the large number of cases like the one sketched above, where an empty category matches one of the categories on the right-hand side of a binary branching rule. We determined that the application of these unit production rules accounted for more than 60 percent of the edges constructed by the parser.

Our next thought, therefore, was to try to transform the grammar to eliminate unit productions as well, but this process turned out to be, in practical terms, intractable. Eliminating empty categories had increased the grammar size but only by about half. When we tried to eliminate unit productions, processing the first four (out of several hundred) grammar rules took a couple of hours of computation time and generated more than 1800 derived rules. We abandoned this approach, and instead we eliminated the unit productions from the grammar by compiling them into a "link table." The link table is basically the transitive closure of the unit productions, so it is, in effect, a specification of the unit derivations permitted by the grammar, omitting the intermediate nodes. This table is then used by the parser to find a path via unit productions between the edges in the chart and the categories that appear in the nonunit grammar rules. This is effectively the same as the CKY algorithm except that edges that would be produced by unit derivations are never explicitly created.

We also made some modifications to speed up selection of applicable grammar rules. We added a "skeletal" chart that keeps track of the sequences of general categories (ignoring features) that occur in the chart (or could be generated using the link table), with the restriction that the only sequences recorded are those that are initial segments of the sequence of general categories (ignoring features) on the right-hand side of some grammar rule. Each grammar rule is itself indexed by the sequence of general categories occuring on its right-hand side. For example, if there is some sort of verb spanning position x through position y in the input and some some sort of noun phrase spanning position y through position z, the skeletal chart would record that there is a sequence of type `v_np` ending at point z. Thus, when the parser searches for applicable rules to apply to generate new edges in the chart at a particular position, it only considers rules which are indexed by an entry in the skeletal chart for that position.

Eliminating unit productions by use of the link table and accessing the grammar rules through the skeletal chart made the parser substantially faster, but this parser is still almost three times slower than the shift reduce parser on our test corpus using our test grammar. At this point, we seemed to have reached a practical limit to how fast we could make the parser while still constructing essentially every possible edge bottom-up. This parser is in fact almost twice as fast as the shift-reduce parser in terms of time per edge constructed, but it constructs more than four times as many edges.

Making Limited Use of Context

Our limited success in constructing a purely bottom-up parser that would be efficient enough for practical use with our unification grammar led us to reconsider whether it is really necessary to compute every phrase that can be identified bottom-up in order to use the output of the parser in a robust interpretation scheme. We again focused our attention on syntactic gaps. Although we had dealt effectively with explicitly empty categories and with categories generated by the unit productions created by the elimination of empty categories, we knew that many of the additional edges the bottom-up parser was creating were for larger phrases that implicitly contain gaps (e.g., a transitive verb phrase with a missing object noun phrase), even when there is nothing in the preceding context to license such a phrase. We reasoned that there is little benefit to identifying such phrases, the vast majority of which would be spurious anyway, because unless we can determine the semantic filler of a gap, the phrase containing it is unlikely to be of any use in robust interpretation.

With this rationale, we have implemented several variants of a bottom-up parsing algorithm that allows us to use limited top-down constraints derived from the left-context to block the formation of just the phrases that implicitly contain gaps not licensed by the preceding context. For example, in the sentence we previously discussed, "What cities does American fly to from Boston?" the interrogative noun phrase "what cities" signals the possible presence of a noun phrase gap later in the sentence. This licenses

fly to
fly to from Boston
American fly to from Boston
does American fly to from Boston

all as being legitimate phrases that contain a noun phrase gap. Without that preceding context, we would not want to consider any of these word strings as legitimate phrases.

To implement this approach we partitioned the set of grammatical categories into context-independent and context-dependent subsets, with the context-dependent categories being those that implicitly contain gaps. Defining which categories those are is relatively easy in our grammar, because we have a uniform treatment of

"wh" gaps, usually called "gap-threading" [5], so that every category that implicitly or explicitly contains a gap has a feature **gapsin** whose value is something other than **null**. We have a similar treatment of the fronting of auxiliary verbs in yes/no questions, controlled by the feature **vstore**. Finally, an additional quirk of our grammar required us to treat all relative clauses as context dependent categories. So we defined the context-independent categories to be those that

- Have **null** as the value of **gapsin** or lack the feature **gapsin**, and
- Have **null** as the value of **vstore** or lack the feature **vstore**, and
- Are not relative clauses.

All other categories are context dependent.

These is, of course, simply one of any number of ways that categories could be divided between context-independent and context-dependent. Our ability to change these declarations gives us an interesting parameterization of our parser, such that it can be run as anything from a purely bottom-up parser, if all categories are declared context-independent, to one that uses maximum prediction based on left context like our shift-reduce parser, if all categories are declared context-dependent. It would also be possible to derive a candidate set of context-dependent categories automatically or semi-automatically from a corpus. The candidates for context-dependent categories would be those categories that most often fail to contribute to a complete parse when found bottom-up.[1]

The basic parsing algorithm remains the same as in the purely bottom-up parsers, with a few modifications. After each rule application the resulting category is checked to see whether it unifies with one of the context-independent categories. If so, the edge for it is added to the chart with no further checking. If not, a test is made to see whether the category is predicted by the preceding left context. If so, it is added to the chart; otherwise, it is rejected.

The main complexities of the algorithm are in the generation and testing of predictions. Whenever an edge is added to the chart, predictions are generated that are similar to "dotted rules" or "incomplete edges," except that predictions include only the remaining categories to be matched, since predictions are not used in a reduction step as they are in other algorithms. So, if we have a rule of the form $A \rightarrow BC$ and we add an edge for B to the chart, then we may add a prediction for C following B. Whether the prediction is made or not depends on a number of things, including whether the left-hand side of the rule is context-dependent or independent. In the current example, if A is a context-independent category, we proceed with the prediction; otherwise, we must check whether A itself is predicted. In addition, predictions can arise from matching part of a previous prediction. If we have predicted AB and we find A, then we can predict B.

In order to minimize the number of predictions made, we make two important checks. First we check that the prediction actually predicts some context-dependent category. Second, we do a "follow" check, to make sure that the predicted category might occur, given the next word in the input stream. There are a few other minor refinements to limit the number of predictions, but these are the most important ones. In order to check whether a context-dependent category is predicted by a certain prediction, we consult a "left-corner reachability table" that tells us whether the category we are testing is a possible left corner of the predicted category.

When we tested this algorithm, we found that it dramatically reduced the number of edges generated, and equally dramatically improved parse time. We noted above that our best purely bottom-up parser was about three times slower that the shift-reduce parser. This algorithm proved to be 20 percent faster than the shift-reduce parser on our test corpus and test grammar.

Examination of the number and type of edges produced by this weakly-predictive parser led us to question whether all the refinements that we had made to the purely bottom-up parsers, in order to deal with the enormous number of edges they produced, were still necessary. We have performed a number of experiments removing some of those refinements, with interesting results. The main effect we observed was that using the link table to avoid creating edges for categories produced by unit derivations is no longer productive. By using the link table to create explicit edges for those categories, so that we do not have to use the link table at the time we match the right-hand sides of rules against the chart, we got a parser that was twice as fast as the shift reduce parser. We also found that leaving empty categories in the grammar actually speeded-up this version of the parser very slightly (about 4 percent). More edges and predictions were generated for the empty categories, but this was apparently more than compensated for by the reduction in the number of grammar rules.

Conclusions

This paper is, in effect, a narrative of an exercise in algorithm design and software engineering. Unlike most algorithms papers, it contains a great deal of detail on what did not work, or at least what did not work as well as had been hoped. It is also notable because it talks about practical, rather than theoretical efficiency. Most papers on parsing algorithms focus on theoretical worst-case time bounds. Although we have not analyzed it, it seems likely that all the algorithms we tried have the same polynomial time bound, but the difference in the constants of proportionality involved makes all the difference between the algorithms being usable and not usable. Also, unlike most experimental results on parsing, ours are based on a real grammar, being developed

[1] This idea arose in response to a question posed by Mitch Marcus.

for a real application, not a toy grammar written only for the purposes of testing parsing algorithms. It is unlikely that the problems with gaps that are absolutely crucial in this exercise would arise in such a toy grammar.

In terms of concrete results, the relative performance of several of the parsers is summarized in the table below.

Parser	Time	# Edges	Time/Edge
shift-reduce	1.00	1.00	1.00
naive bottom-up	8.81	12.52	0.70
best bottom-up	2.95	4.62	0.63
best predictive	0.48	1.79	0.27

Notice that all the new parsers are significantly faster than the shift-reduce parser in terms time per edge generated. This is undoubtedly due to the high overhead of the prediction mechanism used in the shift-reduce parser. It is also interesting to note that among the new parsers, the faster the overall speed of the parser, the faster the time per edge, also. This may be somewhat surprising, because of all the additional mechanisms added to the last two parsers to reduce the number of edges, compared to the naive bottom-up parser. Evidently the benefits of having a smaller chart to search outweighed the costs of the additional mechanism, even on the basis of time per edge.

In summary, our first attempt to produce a bottom-up parser was nine times slower than our baseline system; our last attempt was twice as fast. Thus we achieved a speed up of a factor of 18 over the course of these experiments. We finished not only with a parser that produced the additional possible phrases that we wanted for robust interpretation, but did so much faster than the parser we started with. Furthermore, we have developed what seems to be an important new parsing method for grammars that allow gaps, and perhaps more generally for grammars with a set of categories that can be divided into those constrainted mainly internally and those with important external constraints.

Acknowledgments

This research was supported by the Defense Advanced Research Projects Agency under Contract N00014-90-C-0085 with the Office of Naval Research.

References

[1] E. Jackson, D. Appelt, J. Bear, R. Moore, and A. Podlozny, *A Template Matcher for Robust NL Interpretation*, Proceedings, Fourth DARPA Workshop on Speech and Natural Language (February 1991).

[2] R. Moore, D. Appelt, J. Bear, M. Dalrymple, and D. Moran, SRI's Experience with the ATIS Evaluation, Proceedings, DARPA Speech and Natural Language Workshop (June 1990).

[3] T. Kasami, "An Efficient Recognition and Syntax Algorithm for Context-Free Languages," Scientific Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, Massachusetts (1965).

[4] D. H. Younger, "Recognition and Parsing of Context-Free Languages in Time n^3," *Information and Control* Vol. 10, No. 2, pp. 189–208 (1967).

[5] L. Karttunnen, "D-PATR: A Development Environment for Unification-Based Grammars," Proceedings of the 11th International Conference on Computational Linguistics, Bonn, West Germany, pp. 74–80 (1986).