Steps Toward Real-time Natural Language Processing

Steven L. Lytinen and Noriko Tomuro
DePaul University
Department of Comuter Science and Information Systems
Chicago IL 60604
lytinen@cs.depaul.edu

Abstract

Understanding language is a seemingly effortless task for people. Not only can they understand the meaning of sentences with great accuracy, they do so quickly: in most cases, people understand language in linear time. In constrast, understanding language is not so easy for computers. Even ignoring problems of accuracy, natural language processing systems are much slower than people are. All current NLP systems that fully analyze both the syntactic structure and semantic meaning of text fall short of human performance in this respect.

In this paper, we present an attempt to develop a linear time algorithm for parsing natural language using unification grammars. While the computational complexity of the algorithm is, in the worst case, no better than that of many other algorithms, empirical testing indicates improved average-case performance. Although linear performance has not yet been achieved, we will discuss possible improvements that may result in an average-case linear time algorithm.

Introduction

Unification grammar has become a popular formalism to use in natural language processing (NLP) systems. Unfortunately, the formal power of unification grammar makes it difficult to implement an efficient unification-based parser. A common approach is to build a unification-based parser on top of a context-free chart parser. The result is an algorithm that is at least $O(n^3)$ in the worst case (since this is the worst-case complexity of context-free chart parsing), and perhaps worse, due to the additional work of performing unifications. Adding to the difficulties is the inclusion of semantic information in many unification-based systems, such as in HPSG (Pollard and Sag, 1988) and in our own previous work (Lytinen, 1992). This can greatly increase the size and complexity of a grammar, which also has an adverse effect on performance of a parsing algorithm. Thus, unification-based parsers for complex grammars capable of processing any significant subset of English or other natural languages, even for a limited-domain application, yield quite poor performance. Indeed, empirial examinations of unification-base parsers indicate that average case performance of these systems also falls short of linear (Shann, 1991). Other popular parsing algorithms, such as Tomita's algorithm (Tomita, 1986), also fail to achieve linear performance, even without the inclusion of semantic interpretation.

How efficient should we expect NLP systems to be? One way to answer this question is to observe human performance in understanding language. Although there are exceptional constructions (e.g., garden-path sentences) that can cause problems in comprehension, it seems that in general people process text in linear time. Thus, if we expect an NLP system to match human performance, a reasonable goal is to achieve linear performance in the average case, with perhaps significantly worse worst-case performance.

This paper describes an attempt to implement an average-case linear time unification-based parser. The algorithm that has been implemented is a mixture of top-down and bottom-up chart parsing. In the top-down component, semantic information encoded in the grammar is utilized as much as possible in the production of *active edges* (i.e., expectations for what is to come next in a sentence). The general philosophy is to use this information to produce as specific expectations as possible, thus limiting the possible alternative parses that need to be pursued. While the worst-case performance of this algorithm is no better than other unification-based parsers based on context-free chart parsing, the hypothesis is that the utilization of both syntactic and semantic constraints in top-down expectations will significantly improve average case performance.

The parsing algorithm has been implemented as part of the LINK system (Lytinen, 1992). Specifically, the top-down version of LINK (TDLINK) has been implemented as an alternative version of the system which we used in the Fifth Message Understanding Competition (MUC-5) (Lytinen *et al.*, 1993), so that its performance could be tested on a pre-existing corpus and grammar. MUC-5 systems processed newspaper articles describing new developments in the field of microelectronics. Our original MUC-5 system used a bottom-up chart parser, very similar to PATR-II (Shieber, 1986).

TDLINK was tested on a random sample of sentences from the MUC-5 corpus. The test results are reported and analyzed in this paper. While TDLINK does not appear to achieve average case linear processing time on the sample sentences, the number of edges entered into the chart for a sentence does appear to increase linearly with sentence length, on average. This is an encouraging result, since in context-free chart parsing, processing time is linearly proportional to the number of edges. Possible reasons for why edges and processing

time do not seem to be linearly proportional in TDLINK, and proposals for achieving linear processing time based on TDLINK, are discussed in section 5.

LINK's Unification Grammar

LINK's knowledge base can be thought of as consisting of three modules: a grammar, a lexicon, and a set of domain knowledge. All of these types of knowledge are encoded in unification *constraint rules*. These rules are very similar in form to other unification grammars, such as PATR-II. Figure 1 shows a small piece of a simple knowledge base.

Consider the S rule (eqs. 1-5) in the example grammar. Each equation in this rule specifies a property that any node labeled S (a complete sentence) must have. A property consists of a *path*, or a sequence of arcs with the appropriate labels starting from the node in question; and a *value*, which is another node to be found at the end of the path. Equations specify the values of properties in one of two ways. They may specify the label of the node to be found at the end of the path, as in equations 1 and 2 (i.e., the arc from an S node labeled 1 leads to a node labeled NP). Or, they may specify that two paths must lead to the identical node, as in equations 3-5. Identity here is defined by the *unification* operation; i.e, if two paths must lead to the identical node, then the nodes at the end of the two paths must unify. Unification merges the properties of two nodes; thus, two paths can unify if their values have no properties that explicitly contradict each other.

Functionally, the S rule encodes information about English sentences as follows. Equations 1 and 2 specify that a sentence is made up of two subconstituents: a NP and a VP. Ordering of these constituents is implicit in the numbering of the paths. Equation 3 assigns the HEAD of the sentence to be the VP, by unifying the VP's HEAD path with the HEAD path of the S. This will be discussed further shortly. Equation 4 specifies that the NP and the VP must agree in number and person. These syntactic properties are found under the AGR (agreement) feature of each constituent. Finally, equation 5 assigns the NP to be the subject of the sentence.

The HEAD property referred to in equations 3-5 is used to propagate information up and down the DAG. This is accomplished by unification of HEAD links, as in equation 3. Because of this equation, any information on the HEAD of the VP is accessible from the S node. Other rules assign the heads of other constituents, such as a verb group (VG) to be the HEAD of the VP (in the two VP rules; eqs. 7 and 11), and a verb (V) to be the HEAD of a VG (eqs. 15 and 18).

Lexical items typically provide the values that are propagated by HEAD links. They are encoded in the same form as grammar rules. Typical values provided by lexical rules include syntactic feature information, such as the AGR feature; as well as semantic information, which causes a semantic interpretation of the sentence to be constructed as parsing proceeds. For example, in the entry for "eats", eq. 20 specifies that "eats" is a transtive verb, and eqs. 21-22 specify the word's syntactic agree-

```
(define-class S
   (1) = NP                          <1>
   (2) = VP                          <2>
   (head) = (2 head)                 <3>
   (head agr) = (1 head agr)         <4>
   (head subj) = (1 head))           <5>

(define-class VP
   (1) = VG                          <6>
   (head) = (1 head)                 <7>
   (head type) = intrans)            <8>

(define-class VP
   (1) = VG                          <9>
   (2) = NP                          <10>
   (head) = (1 head)                 <11>
   (head type) = trans               <12>
   (head dobj) = (2 head))           <13>

(define-class VG
   (1) = V                           <14>
   (head) = (1 head))                <15>

(define-class VG
   (1) = AUXES                       <16>
   (2) = V                           <17>
   (head) = (2 head))                <18>

(define-class V
   (1) = eats                        <19>
   (head type) = trans               <20>
   (head agr number) = sing          <21>
   (head agr person) = 3rd           <22>
   (head rep) = EAT-FOOD             <23>
   (head subj rep) =
      (head rep actor)               <24>
   (head dobj rep) =
      (head rep object))             <25>

(define-class EAT-FOOD
   (actor) = ANIMATE                 <26>
   (object) = FOOD                   <27>
   (instrument) = UTENSIL)           <28>
```

Figure 1: A set of example LINK rules

ment features, found under the AGR property. Eq. 23 provides semantic information about the word, specifying that "eats" means EAT-FOOD. Eqs. 24-25 specify mappings from syntactic to semantic dependencies. 24 states that whatever constituent fills the SUBJECT role will also be assigned as the ACTOR of the EAT-FOOD. Similarly, 25 specifies that the syntactic direct object (DOBJ) is assigned as the semantic OBJECT.

The mapping equations are used in conjunction with the system's domain knowledge, to impose restrictions on the semantic properties (i.e., the values of the REP path) of the subject and direct object of "eats" (i.e., the ACTOR and OBJECT of the EAT-FOOD). Domain knowledge is also encoded in constraint rules, as exemplified by the EAT-FOOD rule (eqs. 26-28). Because of the mapping provided by "eats" between its subject and the ACTOR of the EAT-FOOD, the restriction that this constituent's representation must be ANIMATE is propagated up to the NP that fills the SUBJ role specified by equation 26. Similarly, the FOOD restriction on the object of an EAT-FOOD would propagate to the NP assigned as the direct object (DOBJ) of "eats."

The Parsing Algorithm

TDLINK is a bottom-up, breadth-first, left-to-right chart parser which uses top-down expectations (active edges) to eliminate the construction of edges which could not possibly fit into the overall parse of a sentence. TDLINK also filters edges based on a single-word lookahead.

Labels of edges in TDLINK are more complex than is the case in context-free chart parsers: the labels of both active and complete edges are directed acyclic graphs (DAGs), which encode the syntactic category of a constituent (the normal label of an edge in a context-free chart parser) as well as other syntactic and semantic features of the constituent. These features are specified in the definitions of lexical items, as well as in some of the grammar rules, as we saw in section 2.

Active edges are used in conjunction with a *reachability table* in order to find potential connections between an expected DAG and the next word in the sentence. This is done as follows: first, the word is looked up in the dictionary. The result is a list of one or more DAGs, each of which corresponds to a sense of the word. Next, the syntactic label of each word sense DAG is used, along with the syntactic labels of each expected DAG, to look up possible connections in the reachability table. The table lookup results in a list of grammar rules which, if applied, would connect the word to the expectation.

Let us illustrate with a simple example sentence, "Pat eats the sandwich." After processing "Pat", TDLINK has built an active edge labelled S, with an expectation to find a VP beginning at the word "eats" (since the S rule states that a VP should follow the NP). The reachability table provides the information that "eats" can be connected to the expectation in two ways: by applying the first VG rule (eqs. 14-15), followed by either VP rule (eqs. 6-8 or 9-13). In other words, the table provides the information that "eats" must start a verb phrase, and that the VP may be transitive (with a direct object) or intransitive (without an object).

After comparing all active expectations with the next word in the sentence, TDLINK applies the sequences of rules provided by the reachability table in a bottom-up fashion. After each possible sequence is finished, the resulting DAG is unified with the expectation, to make sure that all of their features are compatible. If so, the new edges are added to the chart; if not, they are discarded. In our example, since the TRANS feature of "eats" turns out to be incompatible with the INTRANS feature in eq. 8, one of the rule sequences applied is discarded, leaving only an active VP edge, with the expectation that an NP (the direct object of the verb) will follow.[1] In addition, the NP expectation also contains the information that this NP should refer to a type of FOOD.

While it is a syntactic feature in this example which disambiguates the parse and causes edges to be discarded, in general either syntactic or semantic features (or a combination) can resolve an ambiguity. For example, semantic information would eliminate the active/passive ambiguity in "The log cut by Pat was big" at the word "cut", assuming that semantics required a log to be the OBJECT, rather than the ACTOR, of the action CUT. The early resolution of ambiguity has the potential to dramatically improve average-case parsing performance.

Empirical Results

TDLINK was incorporated into our existing MUC-5 system (Lytinen *et al.*, 1993), replacing the existing bottom-up chart parser that had been used in MUC-5. The system was then run on 100 randomly chosen sentences from the MUC-5 corpus.[2] 27 of these sentences were sucessfully parsed by TDLINK using our MUC-5 grammar. These 27 sentences form the basis of the performance analysis.

First, TDLINK's performance was compared against the performance of the purely bottom-up version of LINK. TDLINK achieved a factor of 4 speedup (measured in CPU time), on average, compared with LINK's performance on the 27 sentences. Performance improvement also increased with sentence length.

Next, TDLINK's performance was analyzed on absolute terms. Figure 2 shows a plot of sentence length vs. the number of edges entered in the chart for the 27 sentences. A weighted R^2 analysis indicates that the best-fitting polynomial for this data is a straight line (R^2 = .86). This is an encouraging result, because parsing time in context-free chart parsing is linearly proportional to the number of edges entered in the chart.

[1] Even without the incompatible features, the intransitive VP edge in this case would be discarded at this point due to lookahead.

[2] Our MUC-5 system contained a filtering mechanism, which discarded all sentences that did not contain at least one word whose meaning was relevant to the domain. Those sentences judged to be irrelevant by the filter were not included in the random sample.

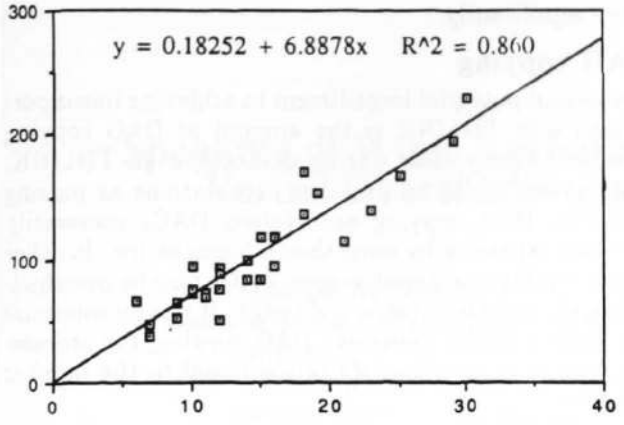

Figure 2: Edges vs. sentences length in TDLINK

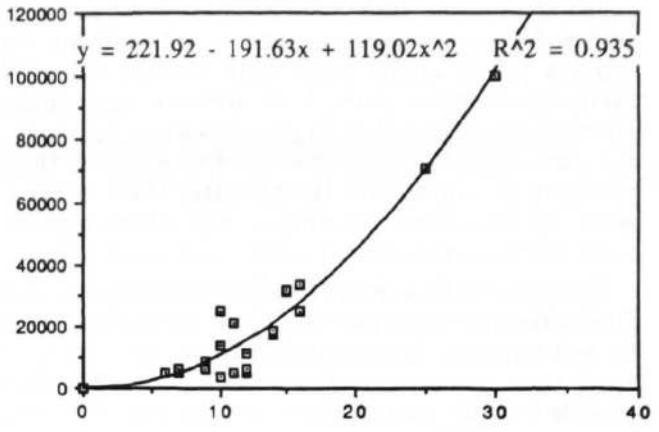

Figure 3: CPU time vs. sentences length in TDLINK

The number of edges constructed was also compared against the minimum number of edges that could possibly be constructed in a complete parse of each of the sentences. This is the number of edges that would be constructed by a chart parser which always parsed every sentence as unambiguous at all stages of the parse. The ratio of the number of edges constructed by TDLINK, divided by the minimum number of edges, remained virtually constant vs. word length, with a value of 2.5. This means that, on average, between 2 and 3 different possible interpretations of a sentence were active as a parse proceeded. The number of interpretations did not increase with sentence length, another encouraging sign.

Figure 3 shows a plot of CPU time used vs. sentence length. A weighted R^2 analysis indicates that the best-fitting polynomial for this data is a quadratic (R^2 = .953), indicating that TDLINK did not achieve average-case linear time performance on the test sentences. Thus, it appears that the number of edges in the chart is not linearly proportional to processing time in TDLINK.

Discussion

There are two possible sources for additional processing time in TDLINK. First, the system may unsuccessfully apply enough rules that the time to build discarded edges overshadows the time to build edges that are entered into the chart. Second, as the sentence progresses, the application of a rule becomes more expensive. This is because the DAG labels of edges further along in the chart become more and more complex. Since unification in LINK is a destructive operation, whenever more than one interpretation is being pursued by the parser, DAGs must be copied before unification is performed, so that additions made to one interpretation do not affect other interpreatations. The more complex the DAG, the more expensive copying is. It may be that the time required to copy DAGs is overshadowing the rest of the overhead required in constructing a new edge, meaning that edge construction is not a constant time operation. Both of these possible difficulties will be discussed below.

Discarded Edges

Sometimes a rule is applied which eventually results in a failed unification. The reason that this happens is due to the bottom-up application of rules found in the reachability table. Thus, all the information contained in expectations, except for an expectation's syntactic category, is not used until after a sequence of rules that connect a word to an expectation has already been applied. This may result in a lot of unnecessary work, if the newly constructed edges are discarded due to a failed unification at the end of the process. We saw an example of this scenario in section 3: TDLINK tried to construct two VP edges beginning with the word "eats", only to discard the edge corresponding to the VP rule for intransitive verbs.

The reachability table only utilizes the syntactic category of an expectation and a word. If additional information from expectations could be utilized during the bottom-up construction of edges which connect a word to an expectation, it might be possible to cut down on the number of discarded edges that are constructed by TDLINK.

We are currently investigating ways to do this. First, we wish to modify the reachability table entries to a form wherein additional information from expectations is directly accessible from a word. This way, rule sequences which will lead to unsuccessful unification at the end of bottom-up application can be eliminated from consideration immediately, and the number of discarded edges will decrease. Connection between an expectation and a word can be automatically constructed by pre-compiling the list of grammar rules in each table entry; one complete DAG which connects two categories will have the

expectation in its root node and the word in its left-corner descendant position, and it will indicate what features (of all kinds) must unify between them. By using this complete DAG, both syntactic and semantic information contained in an expectation at any point in the parse can be brought down to a word: when the expectation is unified with the complete DAG at the top node, the left-corner descendant will inherit some features which must be found in the next input word.

Complete DAGs can also bring down the features specified in the grammar rules to a word: some of the syntactic and semantic features tested during bottom-up rule application are propagated down and become directly accessible from the the left-corner descendant. Then, those features are used to cut down the number of discarded edges: if, during the parse, the input word does not unify with the left-corner descenddant of the complete DAG, then no edge will be created. From the previous example sentence "Pat eats the sandwich.", in processing "eats" from the input, the intransitive VP rule will be eliminated from consideration immediately after the unsuccessful unification of "eats" with this rule's expected verb. By using complete DAGs, since no processing time will be wasted by discarded edges, we expect CPU time to decrease.

We are presently implementing a version of TDLINK which uses this reachability table. In this version, complete DAGs are simplified into DAGs with the top-most category (expectation) and the left-corner descendant (word). Those two categories are directly connected by a special arc named LC; each DAG is associated with one or more rule sequences which compile into the DAG. Although no features from the intermediate rules are recorded in the DAG except for the ones propagated to an expectation or a word, bottom-up rule application is still required to test if the rule sequence will actually lead to successful unification. However, additinal information from expectations can still be utilized to disambiguate the parse and eliminate discarded edges.

Another modification we wish to make to the reachability table is the rule hierarchy. We like to organize the grammar rules which connect an expectation to a word into a discrimination network where each node in the network indicates a value to be found for a specific feature in the word and the grammar rules to select or eliminate. This discrimination network can be constructed automatically by using the complete DAGs described above. After complete DAGs are created for all rule sequences, features in the left-corner descendants are further analyzed to form a hierarchy of feature-value tests. In our example, under the table entry for connecting a VP expectation with a V, the TYPE feature would be tested first in the discrimination net. If a value other than INTRANSITIVE is found in the word, this eliminates the first VP rule (eqs. 16-18) from consideration, and if a value other than TRANSITIVE is found, this eliminates the second VP rule (eqs. 19-23). Since rules are selected by value lookup which requires constant time rather than DAG unification, by using this rule hierarchy, we expect the parsing performance to improve significantly.

DAG copying

The second potential impediment to achieving linear performance in TDLINK is the amount of DAG copying that is currently done during unification. In TDLINK, features are added to previous expectations as parsing proceeds; thus, copying expectation DAGs necessarily becomes expensive by more than a linear factor. In other words, the time required to copy DAGs may be overshadowing the time to construct an edge. If we can minimize the amount of this expensive DAG copying, the processing time may stay linearly proportional to the number of edges.

One way to minimize DAG copying is to modify unification in TDLINK to a non-destructive operation; a resulting new DAG is created only if two DAGs unify. However, this non-destructive unification must be no more expensive than the destructive one; depending on how it is implemented, processing time to decide if two DAGs unify may alone be as expensive as copying them. Attempts to build efficient non-destructive unification algorithms have shown promise (Wroblewski, 1990; Godden, 1990).

Non-destructive unification will be most effective when unification is applied to DAGs that are fairly big and incompatible. This situation can happen at two places in TDLINK: at the end of bottom-up rule application when the expectation is unified with the resulting DAG, and at the end of each grammar rule when all the children are unified to form the parent DAG. Significant reduction in the expense of copying these large DAGs may result in linear performance of our algorithm.

References

Godden, K. (1990). Lazy unification. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, Pittsburgh PA, pp. 180-187.

Lytinen, S. (1992). A unification-based, integrated natural language processing system. *Computers and Mathematics with Applications, 23*(6-9), pp. 403-418.

Lytinen, S., Burridge, R., Hastings, P., and Huyck, C. Description of the LINK system used for MUC-5. In *Proceedings of the Fifth Message Understanding Conference (MUC-5)*, Baltimore MD, August 1993, pp. 293-304.

Pollard, C., and Sag, I. (1987). *Information-based Syntax and Semantics.* Stanford, CA: Center for the Study of Language and Information.

Shann, P. (1991). Experiments with GLR and chart parsing. In Tomita, M. (ed), *Generalized LR Parsing.* Boston: Kluwer Academic Publishers, p. 17-34.

Shieber, S. (1986). *An Introduction to Unification-Based Approaches to Grammar.* Stanford, CA: Center for the Study of Language and Information.

Tomita, M. (1986). *Efficient Parsing for Natural Language.* Boston: Kluwer Academic Publishers.

Wroblewski, D. (1990). Efficient nondestructive unification. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston MA, pp. 491-496.