Left-corner Unification-based Natural Language Processing

Steven L. Lytinen and **Noriko Tomuro**
DePaul University
School of Computer Science, Telecommunications and Information Systems
243 S. Wabash Ave.
Chicago IL 60604
lytinen@cs.depaul.edu

Abstract

In this paper, we present an efficient algorithm for parsing natural language using unification grammars. The algorithm is an extension of *left-corner parsing*, a bottom-up algorithm which utilizes top-down expectations. The extension exploits unification grammar's uniform representation of syntactic, semantic, and domain knowledge, by incorporating all types of grammatical knowledge into parser expectations. In particular, we extend the notion of the *reachability table*, which provides information as to whether or not a top-down expectation can be realized by a potential subconstituent, by including all types of grammatical information in table entries, rather than just phrase structure information. While our algorithm's worst-case computational complexity is no better than that of many other algorithms, we present empirical testing in which average-case linear time performance is achieved. Our testing indicates this to be much improved average-case performance over previous left-corner techniques.

Introduction

A family of *unification-based* grammars has been developed over the last ten years, in which the trend has been to represent syntactic and semantic information more uniformly than in previous grammatical formalisms. In these grammars, many different types of linguistic information, including at least some kinds of syntactic and semantic constraints, are encoded as *feature structures*. In the most extreme versions, such as HPSG (Pollard and Sag, 1994), and our own previous work (Lytinen, 1992), feature structures are used to encode all syntactic and semantic information in a completely uniform fashion.

Standard approaches to unification-based parsing do not reflect this uniformity of knowledge representation. Often a unification-based parser is implemented using an extension of context-free parsing techniques, such as chart parsing or left corner parsing. The context-free (phrase structure) component of the grammar is used to drive the selection of rules to apply, and the additional feature equations of a grammar rule are applied afterward. The result remains a syntax-driven approach, in which in some sense semantic interpretation (and even the application of many syntactic constraints) is performed on the tree generated by the context-free component of the unification grammar.

This standard approach to unification-based parsing is not efficient. Worst-case complexity must be as bad as context-free parsing ($O(n^3)$) and perhaps worse, due to the additional work of performing unifications. Empirical examinations of unification-based parsers have indicated nonlinear average case performance as well (Shann, 1991; Carroll, 1994). Other popular parsing algorithms, such as Tomita's algorithm (Tomita, 1986), also fail to achieve average-case linear performance, even without the inclusion of semantic interpretation.

Our hypothesis is that a uniform approach to processing will result in a more efficient parsing algorithm. To test this hypothesis, we have developed a further extension of left-corner parsing for unification grammars. The extension exploits unification grammar's uniform representation, by incorporating all types of grammatical knowledge into parser expectations. In particular, we have extended the notion of the *reachability table*, which provides information as to whether or not a top-down expectation can be realized by a potential subconstituent, by including all types of grammatical information in table entries, rather than just phrase structure information.[1] We have implemented the extended left-corner parsing algorithm within our unification-based NLP system, called LINK (Lytinen, 1992).

To evaluate the efficiency of our algorithm, we have tested LINK on a corpus of example sentences, taken from the Fifth Message Understanding Competition (MUC-5) (Sundheim, 1993). This corpus consists of a set of newspaper articles describing new developments in the field of microelectronics. Since we competed in MUC-5 using a previous version of LINK, we were able to test our left-corner algorithm using a knowledge base that was developed independent of the algo-

[1] The extended reachability table will be referred to as a *reachability net*, since the additional complexity of table entries requires it to be implemented as a discrimination network.

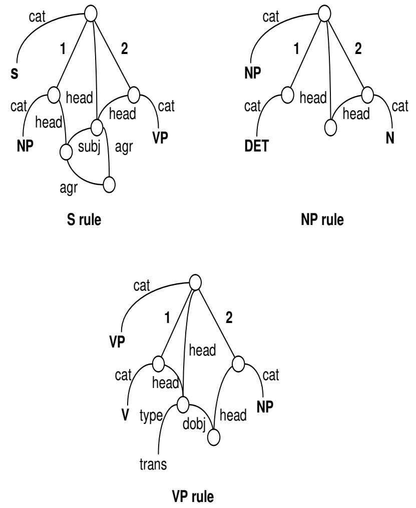

Figure 1: Example LINK grammar rules

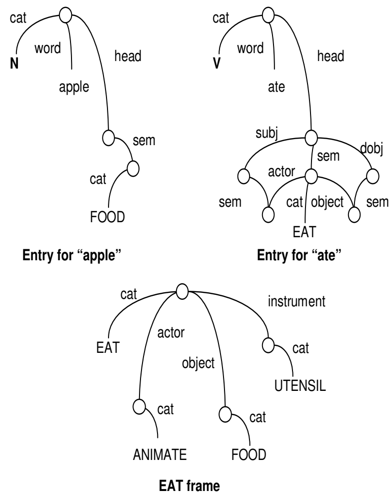

Figure 2: Example LINK lexical entries and frames

rithm, and compare its performance on this corpus directly to the performance of more standard approaches to unification-based parsing. A regression analysis of the data indicates that our algorithm has achieved linear average-case performance on the MUC-5 corpus, a substantial improvement over other unification-based parsing algorithms.

This paper is organized as follows: first we present the uniform knowledge representation used in LINK to represent syntactic, semantic, and domain knowledge. We then discuss LINK's parsing algorithm. Finally, we present results of empirical testing, and discuss its implications.

LINK's Knowledge Representation

All knowledge is encoded in LINK's unification grammar in the form of *feature structures*. A feature consists of a *name* and a *value*. Values may either be atomic or may themselves be feature structures. A feature structure may also have an atomic *label*. Thus, each rule in LINK's knowledge base can be thought of as a directed acyclic graph (DAG), whose edges corresponds to feature names, and whose nodes correspond to feature values.

Figure 1 shows a few simple LINK rules. The **S** rule encodes information about one possible structure of a complete sentence. The **cat** feature of the root indicates that this rule is about the syntactic category **S**. The numbered arcs lead to subconstituents, whose syntactic categories are **NP** and **VP** respectively. Implicit in the numbering of these features is the order in which the subconstituents appear in text. In addition, this rule indicates that the **VP** functions as the **head** of the sentence, that the **NP** is assigned as the **subj** of the sentence, and that the **NP** and **VP** share the same **agr** feature (which encodes the number and person features which must agree between a verb and its subject). Each of the other two rules displayed in figure 1 describes one possible structure for a **NP** and a **VP**, respectively. Other rules exist for the other possible structures of these constituents.

The purpose of the **head** feature is to bundle a group of other features together. This makes it easier for a constituent to inherit a group of features from one of its subconstituents, or vice versa. In this case, the **agr** feature is passed up from the noun and verb to the **NP** and **VP** constituents, to be checked for compatibility in the **S** rule. In the other direction, the **subj** feature is passed down to the verb, where its semantics is checked for compatibility with the semantics of the verb (see figure 2).

Other rules in LINK's knowledge base encode lexical and domain information, such as those in figure 2. Lexical rules typically provide many of the feature values which are checked for compatibility in the grammar rules. For example, the entry for **ate** indicates that this verb is transitive, and thus may be used with the **VP** rule in figure 1. Lexical items also provide semantic information, under the **sem** feature. Thus, "ate" refers to a frame called **EAT**, and "apple" refers to a **FOOD**.

The operation responsible for checking compatibility of features is *unification*, which can be thought of as the combining of information from two DAGs. The result of unifying two DAGs is a DAG with all features from both of the original DAGs. Two DAGs fail to unify if they share a feature with incompatible values.

Domain knowledge is encoded in frame definition rules, such as the **EAT** frame. A node whose **cat** feature has a frame definition must unify with the definition. As a result, semantic type-checking is per-

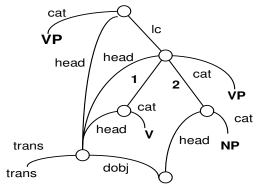

Figure 3: LINK reachability net entry

formed during parsing, resulting in the construction of a semantic interpretation. In these example rules, since the lexical entry for "ate" unifies the **subj** of the verb with the **actor** of its semantic representation, this means the subject of **ate** must be **HUMAN**.

Note that LINK's knowledge base is completely uniform. All rules, including grammar rules, lexical entries, and frame definitions, are represented as DAGs. Moreover, within a DAG there is no structural distinction between syntactic and semantic information. While certain naming conventions are used in the rules for different kinds of features, such as using the **cat** feature for the syntactic category of a constituent and the **sem** feature for its semantic representation, these conventions are only for mnemonic purposes, and play no special role in parsing.

Parsing

The Reachability Net

Context-free left-corner parsers generate top-down expectations in order to filter the possible constituents that are constructed via bottom-up rule application. In order to connect top-down and bottom-up information, a *reachability table* is used to encode what contituents can possibly realize a top-down expectation. The table is constructed by pre-analyzing the grammar in order to enumerate the possible left corner constituents of a particular syntactic category. For example, possible left corners of a NP (noun phrase) might include **DET**, **ADJ**, and **N** (noun), but not **PREP**. In most left-corner unification-based parsers (e.g., see Carroll, 1994), the reachability table is the same: only the syntactic labels of an expectation and a potential subconstituent are used as indices into the table, which then provides information as to which rules may lead to the satisfaction of the expectation.

In LINK, an extended *reachability net* is used, in which entire DAGs, rather than just syntactic labels, serve both as indices and entries. During grammar precompilation in LINK, net entries are constructed by connecting each possible expectation (represented as a DAG) with possible constituents that could be found in a sentence to realize the expectation (also DAGs). A net entry is generated for each possible consituent, which is placed in the **lc** (left corner) arc of the expectation. For example, figure 3 shows the entry for the situation in which a **VP** is expected and a transitive verb is encountered.

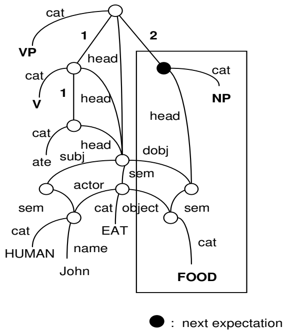

Figure 4: A portion of the parse of the sentence fragment "John ate..."

The use of the reachability net sometimes enables LINK to prune incorrect parses earlier than they otherwise would be. For example, consider the sentence "John slept." After the word "John," the expectation is for a **VP** to follow. Upon encountering "slept", a standard reachability table would indicate that two possible rules could apply: the **VP** rule for transitive verbs pictured in figure 1, and a similar rule for intransitive verbs. Application of the transitive rule would result in a unification failure, assuming that "slept" is marked as intransitive, while the intransitive rule would succeed. In LINK, because net entries contain more than just syntactic category information, only the intransitive verb rule is retrieved in this situation, because the marking of "slept" as intransitive is part of the DAG which is used as an index into the net. Thus, the unification failure is avoided.

Because all features are utilized in retrieval of net entries, semantics can also come into play in the selection of rules. For example, figure 4 shows the **VP** constituent from the parse of the sentence fragment "John ate ...". At this point, the expectation is for an **NP** which means **FOOD**. This semantic information may be used in lexical disambiguation, in the case of an ambiguous noun. For instance, the word "apple" at this point would be immediately disambiguated to mean **FOOD** (as opposed to **COMPUTER**) by this expectation. Structural ambiguities may also be immediately resolved as a result of the semantic information in expectations. For example, consider the sentence fragment "The course taught...". Upon encountering "taught", a standard left-corner parser would attempt to apply at least two grammar rules: the **VP** rule for transitive verbs (see figure 1), and another rule for reduced relative subclauses. In LINK, assuming the existence of a **TEACH** frame whose **ACTOR** should be a **HUMAN**, the transitive **VP** rule would not be retrieved from the reachability net, since the semantics

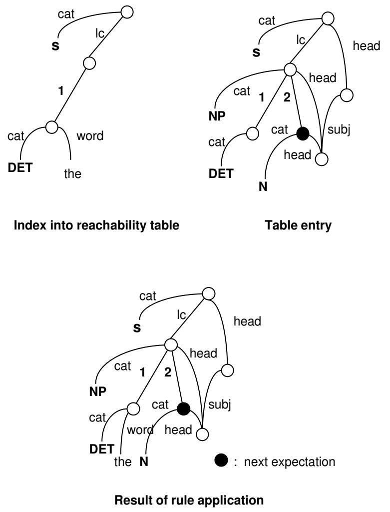

Figure 5: Net entries and DAGs constructed while parsing the word "the"

of "The course taught" do not agree with the **ACTOR** constraint of **TEACH**.

The Parsing Algorithm

At the beginning of the parse of a sentence, LINK constructs an expectation for an **S** (a complete sentence). As the parse proceeds left-to-right, LINK constructs all possible interpretations that are consistent with top-down expectations at each point in the sentence. A rule is applied as soon as its left corner is found in the sentence, assuming the reachability net sanctions the application of that rule given the current expectations. A single-word lookahead is also used to further constrain the application of rules.

LINK's parsing algorithm extends the standard left-corner parsing in the way top-down constraints are propagated down to the lower subconstituents. When a subconstituent is completed (often called a *complete edge* in chart parsing), it is connected to the current expectation through the **lc 1** path. Then, that expectation is used to retrieve the possible rule(s) to apply from the net. If the unification succeeds (creating an active edge with the dot just after the first constituent), the algorithm first checks to see if an expectation for the next word is generated (i.e., there are more constituents to be found after the dot). If there is a new expectation, the iteration stops. Otherwise, the DAG under **lc** arc is complete. That DAG is demoted to **lc 1** path, and the process is repeated. This way, the gap between the expectation and the input word is incrementally filled in a bottom-up fashion, while the top-down constraints are fully intact at each level. Thus, the top-down constraints are applied at the earliest possible time.

Some simple examples will illustrate the key aspects of the algorithm. At the beginning of a sentence, the first DAG in figure 5 is constructed if the word "the" is the first word of a sentence. This DAG is matched against entries in the reachability net, retrieving the entry shown. This entry indicates that the **NP** rule should be applied, resulting in the third DAG shown in figure 5. At this point, the algorithm identifies **N** at the end of **lc 2** path as the expectation for the the next word.

In LINK, a constituent under the **lc** arc is only implicitly connected to the expectation (i.e., the expectation is not completed yet). After all the subconstituents under **lc** arc are found, if the root DAG and the DAG under its **lc** arc unify, it means that the expectation has been fully realized. One possible action at this point is to replace the root with its **lc** arc and continue. This action corresponds to the decision that a constitutent is complete.

Empirical Results

To test the performance of our parsing algorithm, we selected a random set of sentences from the MUC-5 corpus, and parsed them using two different versions of LINK. One version used the extended reachability net as described above; the second version used a standard reachability table, in which only phrase structure information was utilized.

Both versions of LINK were run using a pre-existing knowledge base, developed for the MUC-5 competition. [2] Thus, both versions successfully parsed the same set of 131 sentences from the random sample. These 131 sentences formed the basis of the performance analysis.

Performance was analyzed in terms of several factors. First, a left-corner parser can be thought of as performing several "primitive" actions: rule instantiation and subsequent "dot" advancing, indicating the status of a partially matched grammar rule (i.e., how many of the right-hand side constituents of the rule have matched constituents in the sentence). These two actions involve different operations in the implementation.[3] A rule is instantiated when a constituent in the text (either a lexical item or a completed edge) matches with its left-corner child on the right-hand side. This action involves retrieving the rules from the reachability net and unifying the two constituents. On the other hand, when the dot is advanced, the subconstituent only needs to trace the pointer back to the (partially filled) parent DAG which predicted that constituent at the position right after the dot. Also, since all the expected features were already propagated down when the prediction was made, the subconstituent can be simply replaced into the rule.

[2] In order to improve the coverage of the domain, we added to LINK's original MUC-5 knowledge base for this test.

[3] Both of these actions correspond to the construction of an edge in chart parsing.

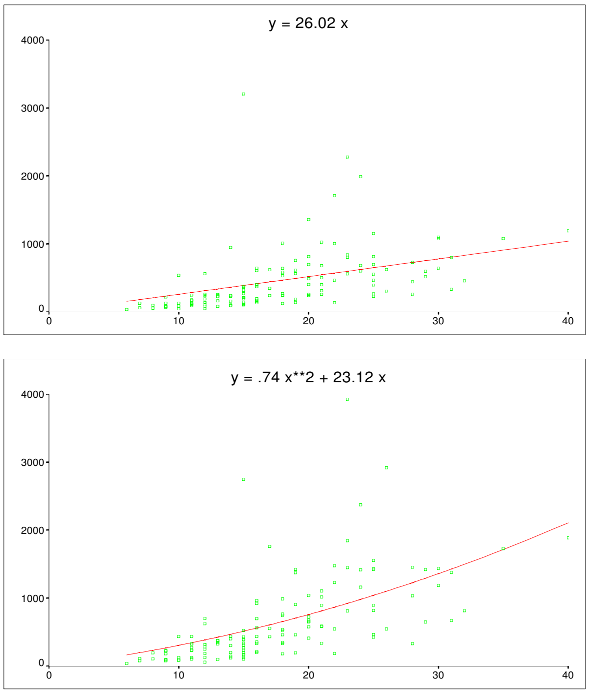

Figure 6: Actions vs. sentence length in LINK using extended and standard reachability tables

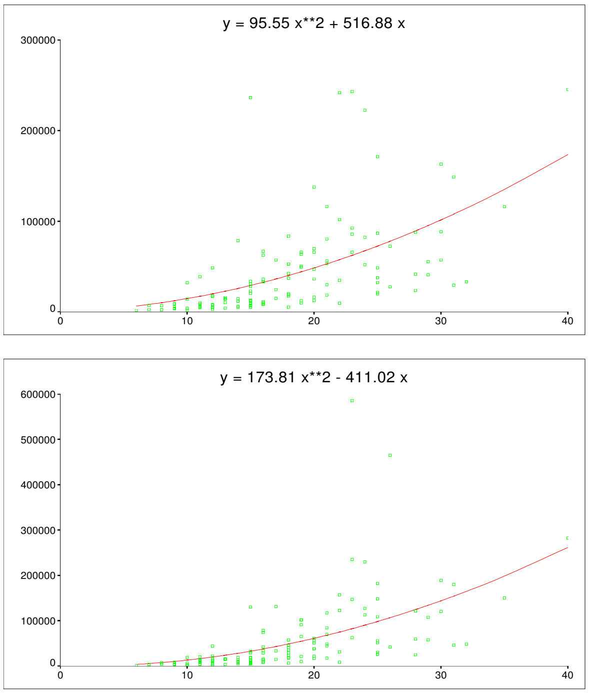

Figure 7: CPU time vs. sentence length in LINK using extended and standard reachability tables

For performance monitoring purpose, those two actions are recorded separately.

Figure 6 shows plots of the number of actions executed during a parse vs. sentence length for LINK using the standard and extended reachability nets. The number of actions also includes failures; i.e., rule instantiations or dot advances which were attempted but in which unification failure occurred (see discussion of rule failures in **Parsing** section). A best regression model analysis[4], using the adjusted R^2 metric, indicates that when using the extended reachability net, LINK achieved linear performance in this respect (R^2 = .599).[5] 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. When using the standard reachability table, a best regression analysis indicates a small quadratic component to the best-fitting curve (adjusted $R^2 = .682$ vs. .673 for the best linear model). When comparing best-fit linear models, on average LINK performed 40% more actions using the standard reachability table than when using the extended reachability net.

Figure 7 shows plots of CPU time used vs. sentence length for the two versions of LINK. The best regression model in both cases for this variable is quadratic.[6] Thus, the number of primitive actions taken by the parser is not linearly proportional to processing time, as it would be for a context-free parser. Average CPU time is 20% longer with the standard reachability table than with the extended reachability net. Thus, this analysis indicates that we have achieved a considerable speed-up in performance over the standard left-corner technique.[7]

Further analysis indicated that a potential source of nonlinear performance in our system is the need to copy DAGs when multiple interpretations are produced. If the reachability net indicates that more than one rule can be applied at some point in the parse, it

[4] In all analyses, best-fitting curves were restricted to those with no constant coefficient (i.e., only curves which pass through the origin). Intuitively, this makes sense when analyzing actions vs. sentence length, since parsing a sentence containing 0 words requires no actions.

[5] Although not shown, performance was also analyzed for a no-lookahead version of this algorithm. The action vs. sentence length result was also linear, but with a much steeper slope.

[6] Intuitively, the best model of CPU time vs. sentence length may contain a constant coefficient, since the algorithm may include some constant-time components; however, when allowing for a constant coefficient, the best regression model results in a negative value for the constant. Thus, we did not allow for constant coefficients in the best models.

[7] We speculate that the difference of the reduction ratio between the number of actions and CPU time comes from the processing overhead by other parts of the system, such as the added complexity of looking up entries in the reachability net.

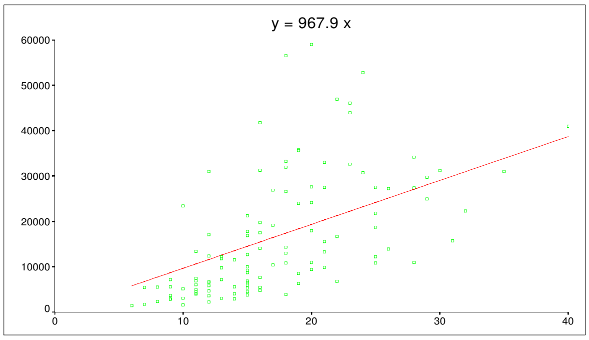

Figure 8: CPU time vs. sentence length in LINK using extended reachability net and improved DAG copying

is necessary to copy the DAG representing the parse so far, so that the alternate interpretations can be constructed without interference from each other. Indeed, a regression analysis of the number of DAGs generated during a parse vs. sentence length using the extended reachability net indicates that a quadratic model is the best for this variable ($R^2 = .637$).

To remedy this problem, we re-implemented the version of LINK using the extended reachability net, this time using a more efficient algorithm for copying DAGs. Our approach is similar to the *lazy unification* algorithm presented in (Godden, 1983). Space constraints prohibit us from describing the copying algorithm in detail. The same set of 131 test sentences was parsed again, and the results were analyzed in a similar fashion. The modified copying algorithm did not affect the number of actions vs. sentence length, since copying had no effect on which rules could or could not be applied. However, it did have a marked effect on the CPU time performance of the system. Figure 8 shows the plot of CPU time vs. sentence length for the lazy version of LINK. On average, the lazy copying algorithm achieved an additional 43% reduction in average CPU time per parse, and an average total speedup of 54% when compared to the version of LINK which used the standard reachability table. In addition, a regression analysis indicates a linear relationship between CPU time and sentence length for the lazy version of LINK (adjusted R^2=.726, vs. an adjusted R^2 of .724 for a quadratic model [8]).

Related Work

Efficient Parsing Algorithms

Many previous efforts have been focused on the construction of efficient parsing algorithms. Some deterministic algorithms such as Marcus' (1980) parser and Register Vector Grammar (Blank, 1989) achieve linear time complexity. However, because linear time is achieved due to the restrictions imposed by determinism, those algorithms consequently limit the generative capacity of the grammar. Our approach, on the other hand, does not limit the generative capacity of our system's unification grammar.

Some nondeterministic algorithms have been developed which utilize efficient encoding techniques. Chart-parsing algorithm uses a *chart* (or table) to record the partial constituents in order to eliminate redundant search. Earley's algorithm (Earley, 1970), a variant of chart-parsing, is proven to run in time $O(n^3)$ for general context-free grammars. Tomita's Generalized LR parsing algorithm (GLR) (Tomita, 1986, 1991) uses a precompiled table, an extension of LR parse table, to guide the search at any given point in the parse. GLR also employs other efficient encoding techniques such as graph-structured stack and *packed* shared forest. However, the worst case complexity of GLR is proven to be no better than Earley's algorithm (Johnson, 1991).

In (Shann, 1991), the performance of several variations of chart-parsing algorithms is empirically tested and compared. In this report, left-corner parsing (LC) with a top-down filtering strategy ranked the highest, and scored even or better in timing than Tomita's GLR. In particular, top-down filtering seemed to make a significant contribution to reducing the parse time. The timing results of this report, however, shows that neither LC nor GLR achieved linear performance in average case.

Parsing Algorithms for Unification Grammars

In (Shieber, 1992), a generalized grammar formalism is developed for the class of unification grammars, and an abstract parsing algorithm is defined. This abstract algorithm involves three components: *prediction*, in which grammar rules are used to predict subsequent constituents that should be found in a sentence; *scanning*, in which predictions are matched against the input text; and *completion*, in which predictions are matched against fully realized subconstituents. Shieber leaves the prediction component intentially vague; depending on the specificity of the predictions generated,[9] the algorithm behaves as a bottom-up parser, a top-down parser, or some combination thereof. On one extreme, if no information is used, the predictor does not propagate any expectations; hence, the algorithm is in essence equivalent to bottom-up parsing. If the predictor limits itself to only the phrase structure information in unification rules, then the algorithm is analogous to traditional (syntax-driven) left-corner parsing. Our algorithm can

[8] While the adjusted R^2 figures for the linear and quadratic models are very close, statistical analysis indicates that the quadratic coefficient in the latter model is not significantly different from 0.

[9] A prediction is created after the filtering function ρ is applied by the predictor.

be characterized as a version of this abstract algorithm in which the most extreme prediction component is used, one in which all possible information is included in the predictions.

Top-down Filtering

Shieber (1985) shows how Earley's algorithm can be extended to unification-based grammars, and the extended algorithm in effect gives a greater power in performing top-down filtering. He proposes *restriction*, a function which selects a set of features by which top-down prediction is propagated. By defining the restriction to select more features (eg. subcategorization, gap or verb form feature) than just phrase structure category, those features are used to prune unsuccessful rule application at the earliest time. Although with a very small example, a substantial effect on parsing efficiency by the use of of restriction is reported.

Another approach taken in (Maxwell and Kaplan, 1994) encodes some (functional) features directly in the context-free symbols (which requires the grammar modification), thereby allowing those features to be propagated down by the *predictor* operation of the Earley's algorithm. Not only does this strategy enable the early detection of parse failure, it can also help exploit the efficiency of the context-free chart-parsing ($O(n^3)$) in unification-based systems. In their report, despite the increased number of rules, the modified grammar showed an improved efficiency.

Early detection of failure is accomplished in LINK in a more pricipled way, by simply including all information in reachability net entries rather than deciding in an *ad hoc* fashion which constraints to encode through subcategorization and which to encode as features.

Conclusion and Future Work

We have presented a unification-based parser which achieves a significant improvement in performance over previous unification-based systems. After incorporating an improved version of DAG copying into the parser, our extended left-corner algorithm achieved average-case linear-time performance on a random sample of sentences from the MUC-5 corpus. This is a significant improvement over standard left-corner parsing techniques used with unification grammars, both in terms of average-case complexity and overall average speed. The improvement is indicated by our own comparative analysis, as well as by comparing our results with empirical testing done by others on standard left-corner parsers and other algorithms such as Tomita's algorithm (e.g., Shann, 1991).

Linear time performance was not achieved without the addition of an improved DAG copying algorithm. Further analysis is required to determine more precisely how much of the improvement in performance is due to the extended reachability net and how much is due to the improved DAG copying. However, our testing indicates that, even without improved copying, the extended reachability net achieves significant improvements in performance as compared to the use of a standard reachability table.

Acknowledgement

The authors would like to thank Joseph Morgan for very useful comments and help on the statistical analysis of the experiment data.

References

Blank, G. (1989). A Finite and Real-Time Processor for Natural Language. *Communications of the ACM, 32*(10) p. 1174-1189.

Carroll, J. (1994). Relating complexity to practical performance in parsing with wide-coverage unification grammars. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics.*

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2).

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

Johnson, M. (1991). The computational complexity of GLR parsing. In Tomita, 1991, pp. 35-42.

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

Marcus, M. (1980). *A theory of syntactic recognition for natural language*, Cambridge, MA: MIT Press.

Maxwell, J. and Kaplan, R. (1994). The interface between phrasal and functional constraints, *Computational Linguistics*, **19** (4).

Pollard, C. and Sag, I. (1994). *Head-driven Phrase Structure Grammar.* Stanford, CA: Center for the Study of Language and Information. The University of Chicago Press.

Shann, P. (1991). Experiments with GLR and chart parsing. In Tomita, 1991, p. 17-34.

Shieber, S. (1985). Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, Chicago, IL, pp. 145-152.

Shieber, S. (1992). *Constraint-based Grammar Formalisms.* Cambridge, MA: MIT Press.

Sundheim, B. (1993). *Proceedings of the Fifth Message Understanding Conference (MUC-5).* San Francisco: Morgan Kaufmann Publishers.

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

Tomita, M. (1991). *Generalized LR Parsing.* Boston: Kluwer Academic Publishers.