

## Chapter 15

# OPTIMAL AMBIGUITY PACKING IN CONTEXT-FREE PARSERS WITH INTERLEAVED UNIFICATION

Alon Lavie

*Language Technologies Institute, Carnegie Mellon University*

*5000 Forbes Avenue, Pittsburgh, PA 15213*

alavie@cs.cmu.edu

Carolyn Penstein Rosé

*Learning Research and Development Center, University of Pittsburgh*

*3939 O'Hare Street, Pittsburgh, PA 15260*

rosecp@pitt.edu

**Abstract** Ambiguity packing is a well known technique for enhancing the efficiency of context-free parsers. However, in the case of unification-augmented context-free parsers where parsing is interleaved with feature unification, the propagation of feature structures imposes difficulties on the ability of the parser to effectively perform ambiguity packing. We demonstrate that a clever heuristic for prioritizing the execution order of grammar rules and parsing actions can achieve a high level of ambiguity packing that is provably optimal. We present empirical evaluations of the proposed technique, performed with both a Generalized LR parser and a chart parser, that demonstrate its effectiveness.

## 1. Introduction

Efficient parsing algorithms for purely context-free grammars have long been known. Most natural language applications, however, require a far more linguistically detailed level of analysis that cannot be supported in a natural way by pure context-free grammars. Unification-based grammar formalisms (such as HPSG), on the other hand, are linguistically well founded, but are

difficult to parse efficiently. Unification-augmented context-free grammar formalisms have thus become popular approaches where parsing can be based on an efficient context-free algorithm enhanced to produce linguistically rich representations. Whereas in some formalisms, such as the ANLT grammar formalism (Briscoe and Carroll, 1993; Carroll, 1993) a context-free “backbone” is compiled from a pure unification grammar, in other formalisms (Tomita, 1990) the grammar itself is written as a collection of context-free phrase structure rules augmented with unification constraints that apply to feature structures that are associated with the grammar constituents. When parsing with such grammar formalisms, the goal of the parser is to produce both a *c-structure* – a constituent phrase structure that satisfies the context-free grammar – and an *f-structure* – a feature structure that satisfies the functional unification constraints associated with phrasal constituents.

Regardless of the specific context-free parsing algorithm used, the most common computational approach to performing the dual task of deriving both *c-structure* and *f-structure* is an interleaved method, where the unification functional constraints associated with a context-free rule are applied whenever the parser completes a constituent according to the rule. If a “bottom-up” parsing approach is used, this process usually involves applying the unification constraints that augment a grammar rule at the point where the right-hand side (RHS) constituents of the rule have all been identified and are then reduced to the left-hand side (LHS) constituent in the rule. The functional unification constraints are applied to the already existing feature structures of the RHS constituents, resulting in a new feature structure that is associated with the LHS constituent. In the case that any of the specified functional constraints cannot be satisfied, unification fails, and the completed left-hand side (LHS) constituent of the rule is pruned out from further consideration.

The computational cost involved in the construction and manipulation of *f-structures* during parsing can be substantial. In fact, in some unification-augmented formalisms, parsing is no longer guaranteed to be polynomial time, and in certain cases may even not terminate (when infinitely many *f-structures* can be associated by the grammar with a given substring of the input (Shieber, 1986). Additionally, as pointed out by Maxwell and Kaplan (1993), interleaved pruning is not the only possible computational strategy for applying the two types of constraints. However, not only is interleaved pruning the most common approach used, but certain grammars, in which the context-free backbone in itself is cyclic, actually rely on interleaved *f-structure* computation to “break” the cycle and ensure termination. Thus, in this paper, we only consider parsers that use the interleaved method of computation.

Ambiguity packing is a well known technique for enhancing the efficiency of context-free parsers. However, as noted by Briscoe and Carroll (Briscoe and Carroll, 1993; Carroll, 1993), when parsing with unification-augmented CFGs,

the propagation of feature structures imposes difficulties on the ability of the parser to effectively perform ambiguity packing. In this paper, we demonstrate that a clever heuristic for prioritizing the execution order of grammar rules and parser actions can significantly reduce the problem and achieve high levels of ambiguity packing. We prove that our ordering strategy is in fact optimal in the sense that no other ordering strategy that is based only on the context-free backbone of the grammar can achieve better ambiguity packing. We then present empirical evaluations of the proposed technique, implemented for both a chart parser and a generalized LR parser, which demonstrate the effectiveness of our rule prioritization technique. Finally, our conclusions and future directions are discussed in Section 5.

## 2. Ambiguity Packing in Context Free Parsing

### 2.1 Ambiguity Packing

Many grammars developed for parsing natural language are highly ambiguous. It is often the case that the number of different parses allowed by a grammar increases rapidly as a function of the length (number of words) of the input, in some cases even exponentially. In order to maintain feasible runtime complexity (usually cubic in the length of the input), many context-free parsing algorithms use a shared parse node representation and rely on performing *Local Ambiguity Packing*. A local ambiguity is a case in which a portion of the input sentence can be parsed and reduced to a particular grammar category (non-terminal) in multiple ways. Local ambiguity packing allows storing these multiple sub-parses in a single common data structure, indexed by a *single* pointer. Any constituents further up in the parse tree can then refer to the set of sub-parses via this single pointer, instead of referring to each of the sub-analyses individually. As a result, an exponential number of parse trees can be succinctly represented, and parsing can still be performed in polynomial time.

Obviously, in order to achieve optimal parsing efficiency, the parsing algorithm must identify all possible local ambiguities and pack them together. Certain context-free parsing algorithms are inherently better at this task than others. Tabular parsing algorithms such as CKY (Younger, 1967) by design synchronize processing in a way that supports easy identification of local ambiguities. On the other hand, as has been pointed out by Billot and Lang (1989), the Generalized LR (GLR) Parsing algorithm (Tomita, 1987) is not capable of performing full ambiguity packing, due to the fact that stacks that end in different states must be kept distinct. There has been some debate in the parsing community regarding the relative efficiency of GLR and chart parsers, with conflicting evidence in both directions (Schabes, 1991; van Noord, 1997; Briscoe and Carroll, 1993; Nederhof and Satta, 1996). The relative effective-

ness of performing ambiguity packing has not received full attention in this debate, and may in fact account for some of the conflicting evidence<sup>1</sup>.

## 2.2 The Problem: Ambiguity Packing in CFG Parsing with Interleaved Unification

Most context-free parsing algorithms, including GLR and chart parsing, are under-specified with respect to the order in which various parsing actions must be performed. In particular, when pursuing multiple ambiguous analyses, the parsing actions for different analyses may be arbitrarily interleaved. In a chart parser, for example, this non-determinism is manifested via the choice of which key (or inactive edge) among the set of keys waiting in the agenda should next be used for extending active edges. In the case of a GLR parser the non-determinism appears in the choice of which among a set of possible rule reductions should be picked next.

The particular order in which parsing actions are performed determines when exactly alternative analyses of local ambiguities are created and detected. With certain orderings, a new local ambiguity for a constituent may be detected after the constituent has already been further processed (and incorporated into constituents higher up in the parse tree). When parsing with a pure CFG, this does not affect the ability of the parser to perform local ambiguity packing. When a new local ambiguity is detected, it can simply be packed into the previously created node. Any pointers to the packed node will now also point to the new sub-analysis packed with the node. However, when parsing with unification-augmented CFGs, this is often not the case. Since feature structures corresponding to the various analyses are propagated up the parse tree to parent nodes, a new local ambiguity requires the creation of a new feature structure that must then be propagated up to any existing parent nodes in the parse tree. In effect, this requires re-executing any parsing actions in which the packed node was involved. With certain “pure” unification grammar formalisms such as GPSG (Gazdar *et al.*, 1985; Alshawi, 1992; Carroll, 1993), feature structure subsumption provides a solution to this problem. Other unification-augmented CFG grammar formalisms, however, require a full encoding of the ambiguous feature structures. In many implementations – for example, the Generalized LR parser/compiler (Tomita, 1990) and the CLE, (Alshawi, 1992) – to avoid the rather complex task of re-calculating the feature structures, ambiguity packing is not performed if it is determined that the previous parse node has already been further processed. Instead, the parser creates a new parse node for the newly discovered local ambiguity and processes the new node separately. As a result, the parser’s overall local ambiguity packing is less than optimal, with a significant effect on the performance of the parser.

It should be noted that effective local ambiguity packing when parsing with unification-augmented CFGs requires not only a compact representation for the packed constituent (c-structure) nodes, but also an effective representation of the associated feature structures (f-structures). The issue of how to effectively pack ambiguous f-structures has received quite a bit of attention in the literature over the last decade (Eisele and Dorre, 1998; Maxwell and Kaplan, 1991; Maxwell and Kaplan, 1993; Miyao, 1999; Kiefer *et al.*, 1999). Particularly noteworthy is Placeway's recent work (Placeway, 2002) on how to improve overall parser efficiency using a two-pass feature structure computation method. While the issues of c-structure and f-structure packing are related, we focus here on how to achieve optimal packing of c-structures, so that unification operations do not have to be re-executed due to the later detection of an additional local ambiguity. While there is much to be gained from improved packing on the f-structure level, one should note that effective ambiguity packing on the c-structure level is a necessary pre-condition for efficient parsing, regardless of how well f-structures are packed.

### 3. The Rule Prioritization Heuristic

In order to ensure effective ambiguity packing in unification-augmented context-free parsers, the situation in which a new local ambiguity is detected after the previous ambiguity has already been further processed must be avoided as much as possible. Ideally, we would like to further process a constituent only after all local ambiguities for the constituent have been detected and packed together into the appropriate node. Our goal was thus to find a computationally inexpensive heuristic that can determine the optimal ordering, or at least closely approximate it. The heuristic that we describe in this section uses only the context-free backbone of the grammar and the spans of constituents in determining what grammar rule (or parsing action) should be "fired" next. Later in the section we prove that this heuristic does in fact achieve the best amount of packing possible, given the information available. In practice, it is easy and fast to compute, and it results in very substantial parser efficiency improvements, as demonstrated by the evaluations presented in the following section. The heuristic was initially developed for the GLR parser, parsing with a unification-augmented CFG. It was then modified to handle the corresponding ordering problem in a chart parser.

#### 3.1 The GLR Ordering Heuristic

In the context of the GLR parser, the heuristic is activated at the point where a decision has to be made between multiple applicable grammar rule reductions. The following example demonstrates the problem and how we would like to address it. Let us assume we have just executed a rule reduction by

[rule0: A --> B C] that created a parse node for a constituent A that spans words 4 – 7 of the input. Assume we now have to choose between the following possible rule reductions:

- 1 Reduce by [rule1: D --> A] that would create a constituent D spanning words 4 – 7 (using the previously just created constituent A).
- 2 Reduce by [rule2: A --> E F] that would create a constituent A spanning words 4 – 7.
- 3 Reduce by [rule3: G --> B A] that would create a constituent G spanning words 3 – 7 (using the previously just created constituent A).

Which rule reduction should be picked next? Since the last rule reduction created a constituent A spanning 4 – 7, by picking [rule2] we can create another constituent A spanning 4 – 7, that can then potentially be packed with the previous A. If we pick [rule1] on the other hand, we would further process the previously created constituent A. When [rule2] is then executed later, the new A can no longer be packed with the previous A. Thus, it is best to choose to perform the [rule2] reduction first.

The first criterion we can use in a heuristic aimed at selecting the desired rule is the span of the constituent that would result from applying the rule. Obviously, we wish to delay applying rules such as [rule3] which process the first A until all other possible A's of the same span have been found and packed. Ignoring for the moment the complications arising from unary rules (such as [rule1: D --> A] above) and epsilon rules (which consume no input), selecting a rule that is “rightmost” - creating a constituent with the greatest possible starting position, will in fact achieve this goal. This is due to the fact that in the absence of unary and epsilon rules, every constituent must cover some substring of the input, and thus every RHS constituent in a grammar rule must start at a different input position. Thus, the “rightmost” criterion supports the goal of delaying the construction of constituents that extend “further to the left” until all ambiguities of the previous span have been processed, and all potential local ambiguities have been packed.

In the presence of unary grammar rules, however, the “rightmost” criterion is insufficient. Looking again at our example above, both [rule1] and [rule2] are “rightmost”, since both create a constituent that starts in position 4. However, [rule1] would further process the existing A constituent before [rule2] detects a local ambiguity. The problem here is that the application of unary rules does not consume an additional input token. We therefore require a more sophisticated measure that can model the dependencies between constituents in the presence of unary rules. To do this, we use the context-free backbone of the grammar, and define the following partial-order relation “ $\geq$ ” between constituent non-terminals:

Input: A set of applicable grammar rule reductions.

Output: A selected grammar rule reduction to be performed next.

Selection Heuristic:

- (1) For each potential grammar rule reduction, determine the span (start and end positions) and category of the constituent that would result from the rule reduction.
- (2) Select the rule reduction that is rightmost - has the largest start position.
- (3) If there is more than one rightmost rule reduction, pick a rule reduction that results in a category that is "least" according to the " $>^*$ " partial-order.

Figure 15.1. Rule Reduction Selection Heuristic for GLR Parser

- For every unary rule  $A \rightarrow B$  in the grammar  $G$ ,  $A \geq B$ .
- We compute " $\geq^*$ " - the transitive closure of " $\geq$ "

We can now extend our heuristic to use the partial-order information. The idea is not to perform a unary reduction resulting in a constituent  $B$  when there is an alternative "rightmost" reduction that produces a constituent  $A$  where  $B \geq^* A$ . The resulting GLR parser heuristic can be seen in Figure 15.1.

The partial-order " $\geq^*$ " can easily be pre-compiled from the grammar. Note that it is theoretically possible that for particular non-terminals  $A$  and  $B$ , both  $A \geq^* B$  and  $B \geq^* A$ . This implies that the context-free backbone of the grammar contains a cycle. With unification-augmented CFGs, this is indeed possible, since the unification equations augmenting the grammar rules may resolve the cycle. In such cases, the above heuristic may encounter situations where there is no unique rule that is minimal according to the partial-order. If this occurs, we pick one of the "least" categories randomly. This may result in sub-optimal packing, but only full execution of the unification operations can in fact correctly decide which rule should be picked in such cases. In practice, the computational cost of such a test would most likely far exceed its benefits.

### 3.2 Handling Epsilon Rules

The case that the grammar contains epsilon rules requires some additional attention. In this case, the premise that every RHS constituent of a grammar rule starts at a different position (i.e. consumes input) no longer holds. Consequently, the "rightmost" criterion no longer ensures that a rule that includes a constituent  $A$  on the RHS will not fire until all possible local ambiguities of  $A$  have been processed and packed. It is useful to note, however, that the problem is in fact similar to that of unary rules, and can be treated quite similarly as well. We first find all "nullable" non-terminals in the grammar  $G$  - the set of all non-terminals that can derive the empty string. A well known algorithm for detecting nullable non-terminals can be found in Hopcroft and Ullman (1979). We denote by  $A \in EP(G)$  that  $A$  is nullable. We now modify the partial-order

relation defined above to handle the case of nullable categories. We define “ $\geq_\epsilon^*$ ” as follows:

- 1 if  $A \geq^* B$  then  $A \geq_\epsilon^* B$
- 2 for every rule  $A \rightarrow B_1 B_2 \cdots B_k$  in  $G$ , check if  $B_i \in EP(G)$  (for  $1 \leq i \leq k$ ). If all  $B_i \in EP(G)$ , then for all  $1 \leq i \leq k$ ,  $A \geq_\epsilon^* B_i$ . Otherwise, if the rule is such that only *one*  $B_i \notin EP(G)$ , then  $A \geq_\epsilon^* B_i$ .
- 3 we compute the transitive closure of  $\geq_\epsilon^*$

We then replace the “ $\geq^*$ ” relation in our ordering heuristic with the extended “ $\geq_\epsilon^*$ ” relation. Intuitively, the idea is to identify grammar rules  $A \rightarrow B_1 B_2 \cdots B_k$  that are not unary, but where when “fired”, the LHS constituent  $A$  may have a span identical to that of one (or more) of the  $B_i$  RHS constituents. This is the case whenever all the RHS constituents (and thus also the LHS constituent) derive the empty string, or else all but one of the RHS constituents ( $B_i$ ) derive the empty string. In the first case, we want to ensure that all empty derivations of the  $B_i$  are selected before the rule reducing to  $A$  is applied. In the second case, we want to ensure that all derivations of the non-nullable  $B_i$  are selected before the rule reducing to  $A$ . We thus extend the partial-order so that the appropriate RHS constituents in such rules are “lesser” in the order than the LHS constituent. This ensures that such rules will not be selected until all local ambiguities of the RHS constituents have been processed and packed.

### 3.3 Proof of Optimality

We now argue that the complete ordering heuristic that uses the “rightmost” and “least” criteria as defined above is in fact optimal in the sense that it achieves the maximal ambiguity packing possible given only the context-free backbone of the grammar as available information. Let us assume to the contrary that the heuristic does not result in optimal packing. This implies that a constituent  $A_1$  of span  $(i, j)$  was created, that a rule in which  $A_1$  serves as a RHS constituent was then selected by the heuristic, executed, and created a LHS constituent  $B_1$ , and that subsequently, another rule with a LHS of  $A$  was executed, creating a new  $A_2$  of span  $(i, j)$  that can no longer be packed with the previous  $A_1$ .  $B_1$  must also be of span  $(i, j)$ , since otherwise it could not have been chosen due to the “rightmost” criterion. At the time the rule creating  $B_1$  was selected by the heuristic,  $B$  must have satisfied the “least” criterion. If the  $A_2$  is created as a result of processing  $A_1$  via a series of rule applications, then the grammar is in fact cyclic. As mentioned earlier, while our heuristic is not guaranteed to be optimal in such cases, any better heuristic would require using f-structure unification information. So we assume that  $A_2$  was created

via a series of rule applications that did not involve  $A_1$ . We look at the sequence of constituents  $X_1, X_2, \dots, A$  created in the series of rule applications that resulted in  $A_2$  after  $A_1$  had already been created. All must have span  $(i, j)$ , since otherwise  $A_2$  cannot have a span of  $(i, j)$ . At least one of these rule applications must have been a possible candidate at the point in time that the rule using  $A_1$  and creating  $B_1$  was chosen and executed. However, by the definition of  $\geq_\epsilon^*$ , for each of the  $X_i$ ,  $A \geq_\epsilon^* X_i$ , and since  $B_1$  is created by applying a rule that uses  $A_1$ , we also have that  $B \geq_\epsilon^* A$ . By the transitive closure property of  $\geq_\epsilon^*$  we thus have that for all of the  $X_i$ ,  $B \geq_\epsilon^* X_i$ . Thus, at the point in time where the rule creating  $B_1$  was fired, either  $B$  was *not* minimal according to the “least” criterion, and we derive a contradiction, or else both  $B$  and some  $X_i$  were minimal, requiring a random choice between the two. In the latter case, as mentioned earlier, the grammar contains a cycle, and only full execution of the unification operations could correctly decide which of the two rules should be picked first for execution. This case excluded, we can thus conclude that the heuristic is in fact optimal.

### 3.4 The Chart Parser Ordering Heuristic

In order to achieve effective ambiguity packing in the case of the chart parser, we require that the parser process the input strictly left-to-right, in order to synchronize the creation and processing of constituents with similar spans. While chart parsers in general do not require such ordered processing, many chart-based parsers are in fact left-to-right, and this is usually not a burdensome restriction. Our rule reordering heuristic can then be added to the parsing algorithm in a straightforward way. In the case of the chart parser, rather than directly reordering reductions, the reordering heuristic modifies the order in which active edges are extended. The idea is that the most efficient way to extend active edges is to ensure that active edges like  $[A \rightarrow \dots \bullet C \dots]$  are not extended over inactive edges of category  $C$  until all possible inactive edges of category  $C$  starting in the appropriate position in the chart have been inserted and packed. The same criteria of “rightmost” and “least” are applied in deciding which active edge should be selected next for extension.

## 4. Empirical Evaluations and Discussion

To evaluate the effectiveness of our rule selection heuristic we conducted empirical test runs with both a GLR parser and a chart parser. Both parsers are designed to support the same LFG style unification grammar framework. The two parsers and the grammar formalism are briefly described below. Both parsers also have robust versions that are designed to overcome input that is not completely grammatical due to disfluencies or lack of grammar coverage. The robust parsers can skip over words or segments of the input that cannot

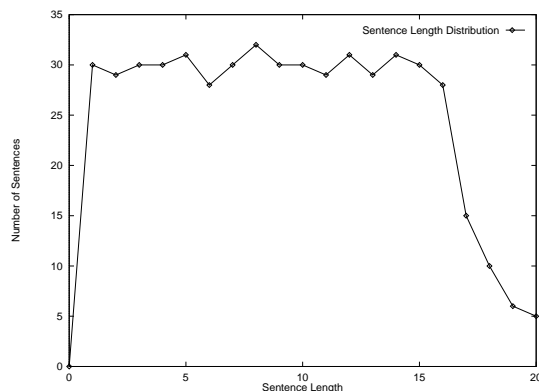


Figure 15.2. Distribution of evaluation set sentence lengths

be incorporated into a grammatical structure. When operated in robust mode, the skipping of words and segments introduces a significantly greater level of local ambiguity. In many cases, a portion of the input sentence may be reduced to a non-terminal symbol in many different ways, when considering different subsets of the input that may be skipped. Thus, efficient runtime performance of the robust versions of the parsers is even more dependent on effective local ambiguity packing. We therefore conducted evaluations with the two parsers in both robust and non-robust modes, in order to quantify the effect of the rule selection heuristic under both scenarios.

All of the described experiments were conducted on a common test set of 520 sentences from the JANUS English Scheduling Task (Lavie et al., 1996), using a general English syntactic grammar developed at Carnegie Mellon University. The grammar has 412 rules and 71 non-terminals, and produces a full predicate-argument structure analysis in the form of a feature structure. For the GLR parser, the grammar compiles into an SLR parsing table with 628 states and 8822 parsing actions. Figure 15.2 shows the distribution of sentence lengths in the evaluation set.

#### 4.1 The GLR Parser

The Generalized LR Parser/Compiler (Tomita, 1990) is a unification based practical natural language analysis system that was designed around the GLR parsing algorithm at the Center for Machine Translation at Carnegie Mellon University. The system supports grammatical specification in an LFG framework, that consists of context-free grammar rules augmented with feature bundles that are associated with the non-terminals of the rules. Feature structure computation is, for the most part, specified and implemented via unification operations. This allows the grammar to constrain the applicability of

context-free rules. A reduction by a context-free rule succeeds only if the associated feature structure unification is successful as well. The Generalized LR Parser/Compiler is implemented in Common Lisp, and has been used as the analysis component of several different projects at the Center for Machine Translation at CMU in the course of the last decade.

GLR\*rser\* (Lavie, 1994; 1996b; 1996a), the robust version of the parser, was constructed as an extended version of the unification-based Generalized LR Parser/Compiler. The parser skips parts of the utterance that it cannot incorporate into a well-formed sentence structure. Thus it is well-suited to domains in which non-grammaticality is common. The parser conducts a search for the maximal subset of the original input that is covered by the grammar. This is done using a beam search heuristic that limits the combinations of skipped words considered by the parser, and ensures that it operates within feasible time and space bounds. The GLR\* parser also includes several tools designed to address the particular difficulties of parsing spontaneous speech. These include a statistical disambiguation module and a collection of parse evaluation measures which are combined into an integrated heuristic for evaluating and ranking the parses produced by the parser.

## 4.2 The LCFLEX Parser

The LCFLEX parser (Rosé and Lavie, 2001) is a recently developed robust left corner chart parser designed to incorporate the flexibility of GLR\* within a more efficient parsing architecture. Its left corner chart parsing algorithm is similar to that described in (Carroll, 1993). Thus, it makes use both of bottom-up predictions based on newly created inactive edges as well as top-down predictions based on active edges bordering on those edges. It utilizes the same grammar formalism used in GLR\*, described above. The context free backbone within the GLR grammar formalism allows for efficient left corner predictions using a pre-compiled left corner prediction table, such as that described by Van Noord (1997). It incorporates similar statistical disambiguation and scoring techniques to those used in GLR\*rser\*, as described by Lavie (1996a).

## 4.3 Evaluation Results

We first ran both the GLR and the LC parsers in non-robust mode (with no word skipping allowed), once without the rule reordering heuristic and once with the heuristic. Figure 15.3 (left) shows a plot of the average number of created parse nodes as a function of sentence length. For both parsers, the total number of nodes created when rule reordering is applied significantly decreases, especially with longer sentences. This is a direct result of the increased level of ambiguity packing performed by the parsers when rule reordering is

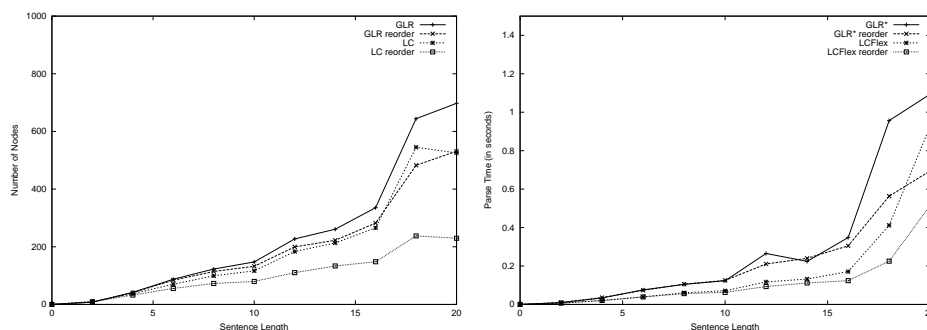


Figure 15.3. Non-robust parsers: number of parse nodes and parse time as a function of sentence length

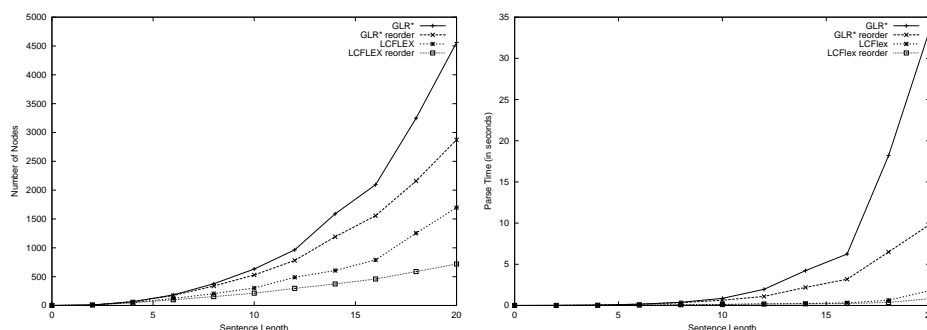


Figure 15.4. Robust parsers: number of parse nodes and parse time as a function of sentence length

applied. For the LC parser, the savings are quite dramatic. For example, for sentences of length 12, the average relative reduction in number of parse nodes with reordering was about 12% for the GLR parser and about 40% for the LC parser. As can be expected, this relative reduction rate appears to grow as a function of sentence length, due to increasing levels of ambiguity. Figure 15.3 (right) shows a plot of the average parse times as a function of sentence length. As can be seen, the LC parser is for the most part faster than the GLR parser, regardless of whether or not rule reordering is applied. However, for both parsers, the improved ambiguity packing with rule reordering results in a significant reduction in parsing time which increases with sentence length, due to increasing levels of ambiguity. For sentences of length 12, the time savings were about 21% for both parsers.

We then re-ran the experiment with the robust versions of the two parsers. The GLR\* parser was run with a search beam of 30, which was determined in various previous experiments to be a reasonable setting for achieving good

parsing results within feasible parsing times. The LCFLEX parser was then run in a way that simulated the same skipping behavior of GLR\* (i.e. the exact same word skipping possibilities were considered by the parser). Figure 15.4 (left) shows a plot of the average number of created parse nodes as a function of sentence length. Once again, the total number of nodes created when rule reordering is applied significantly decreased. For GLR\*rser\*, the reduction in number of nodes was more significant than the non-robust case, while for LCFLEX it appears to be about the same. For example, for sentences of length 12, the average relative reduction in number of parse nodes with reordering was about 19% for the GLR\* parser and about 39% for the LCFLEX parser. Figure 15.4 (right) shows a plot of the average parse times as a function of sentence length. As can be seen, in robust mode, the LCFLEX parser is much faster than GLR\*. As expected, runtimes when rule reordering is applied significantly decreased. For GLR\*, the decrease is more pronounced than in the non-robust experiment, while for LCFLEX, it was about the same. For sentences of length 12, the time savings were about 44% for GLR\* and about 21% for LCFLEX.

## 5. Conclusions and Future Directions

The efficiency of context-free parsers for parsing natural languages crucially depends on effective ambiguity packing. As demonstrated by our experimental results presented in this paper, when parsing with CFGs with interleaved unification, it is vital to execute parsing actions in an order that allows detecting local ambiguities in a synchronous way, before their constituents are further processed. Our grammar rule prioritization heuristic orders the parsing actions in a way that achieves the best possible ambiguity packing using only the context-free backbone of the grammar and the constituent spans as input to the heuristic. Our evaluations demonstrated that this indeed results in a very substantial improvement in parser efficiency, particularly so when incorporated into robust versions of the parsers, where local ambiguities abound.

The focus of most of our current and planned future research is on efficient parsing within the context of the robust LCFLEX parser. In particular, we are interested in investigating the interdependence between the parser's robustness capabilities, its search strategy, and its disambiguation and parse selection heuristics. Effective ambiguity packing will continue to be an important factor in this investigation. We plan to further explore strategies other than interleaved pruning for efficiently applying both phrasal and functional constraints, while using the existing grammar framework and broad-coverage grammars that are at our disposal.

## Notes

1. For example, van Noord (van Noord, 1997) compares left corner and chart parsers that do apply ambiguity packing with a GLR parser without ambiguity packing.

## References

- Alshawi, H., editor (1992). *The Core Language Engine*. ACL-MIT Series in Natural Language Processing. MIT Press, Cambridge, MA.
- Billot, S. and Lang, B. (1989). The Structure of Shared Forests in Ambiguous Parsing. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics (ACL'89)*, Vancouver, BC, Canada.
- Briscoe, T. and Carroll, J. (1993). Generalized Probabilistic LR Parsing of Natural Language (Corpora) with Unification-Based Grammars. *Computational Linguistics*, 19(1):25–59.
- Carroll, J. (1993). *Practical Unification-Based Parsing of Natural Language*. PhD thesis, University of Cambridge, Cambridge, UK. Computer Laboratory Technical Report 314.
- Eisele, A. and Dorre, J. (1988). Unification of Disjunctive Feature Descriptions. In *Proceedings of the 26th Annual Meeting of the ACL (ACL-88)*, Buffalo, NY.
- Gazdar, G., Klein, E., Pullum, G., and Sag, I. (1985). *Generalized Phrase Structure Grammar*. Harvard University Press, Cambridge, MA.
- Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages and Computation*, chapter 4.4, page 90. Addison-Wesley.
- Kiefer, B., Krieger, H.-U., Carroll, J., and R., M. (1999). A Bag of Useful Techniques for Efficient and Robust Parsing. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL'99)*, pages 473–480, College Park, MD.
- Lavie, A. (1994). An Integrated Heuristic Scheme for Partial Parse Evaluation. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics (ACL-94)*, pages 316–318, Las Cruces, New Mexico.
- Lavie, A. (1996a). *GLR\*: A Robust Grammar-Focused parser for Spontaneously Spoken Language*. PhD thesis, School of Computer Science, Carnegie Mellon University. Technical Report CMU-CS-96-126.
- Lavie, A. (1996b). GLR\*: A Robust Parser for Spontaneously Spoken Language. In *Proceedings of ESSLLI-96 Workshop on Robust Parsing*.
- Lavie, A., Gates, D., Gavalda, M., Mayfield, L., Waibel, A., and Levin, L. (1996). Multi-lingual Translation of Spontaneously Spoken Language in a Limited Domain. In *Proceedings of International Conference on Computational Linguistics (COLING'96)*, pages 442–447, Copenhagen, Denmark.

- Maxwell, J. T. and Kaplan, R. M. (1991). A Method for Disjunctive Constraint Satisfaction. In Tomita, M., editor, *Current Issues in Parsing Technology*. Kluwer Academic Press.
- Maxwell, J. T. and Kaplan, R. M. (1993). The Interface between Phrasal and Functional Constraints. *Computational Linguistics*, 19(4):571–590.
- Miyao, Y. (1999). Packing of Feature Structures for Efficient Unification of Disjunctive Feature Structures. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL'99)*, pages 579–584, College Park, MD.
- Nederhof, M.-J. and Satta, G. (1996). Efficient Tabular LR Parsing. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL'96)*.
- Placeway, P. (2002). *High-Performance Multi-Pass Unification Parsing*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA. Technical Report CMU-LTI-02-172.
- Rosé, C. P. and Lavie, A. (2001). Balancing Robustness and Efficiency in Unification-augmented Context-Free Parsers for Large Practical Applications. In van Noord and Junqua, editors, *Robustness in Language and Speech Technology*, ELSNET. Kluwer Academic Press.
- Schabes, Y. (1991). Polynomial Time and Space Shift-Reduce Parsing of Arbitrary Context-free Grammars. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL'91)*.
- Shieber, S. M. (1986). *An Introduction to Unification-based Approaches to Grammar*. Number 4 in CSLI Lecture Notes. CSLI Stanford University, Stanford, CA.
- Tomita, M. (1987). An Efficient Augmented Context-free Parsing Algorithm. *Computational Linguistics*, 13(1-2):31–46.
- Tomita, M. (1990). The Generalized LR Parser/Compiler - Version 8.4. In *Proceedings of International Conference on Computational Linguistics (COLING'90)*, pages 59–63, Helsinki, Finland.
- van Noord, G. (1997). An Efficient Implementation of the Head-Corner Parser. *Computational Linguistics*, 23(3):425–456.
- Younger, D. (1967). Recognition and Parsing of Context-Free Languages in Time  $n^3$ . *Information and Control*, 10(2):189–208.