In Recent Advances in Parsing Technology

GLR* – An Efficient Noise-skipping Parsing Algorithm For Context Free Grammars

Alon Lavie and Masaru Tomita

School of Computer Science, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213
email: lavie@cs.cmu.edu

Abstract

This chapter describes GLR*, a parser that can parse *any* input sentence by ignoring unrecognizable parts of the sentence. Using an efficient algorithm, the parser is capable of finding and parsing a maximal subset of the original input that is parsable, and therefore return the parse with fewest skipped words. The parser returns some parse(s) for any input sentence, unless no part of the sentence can be recognized at all.

Formally, the problem can be defined in the following way: Given a context-free grammar G and a sentence S, find and parse S' - the largest subset of words of S, such that $S' \in L(G)$.

The algorithm described in this chapter is a modification of the Generalized LR (Tomita) parsing algorithm (Tomita, (1986)). The parser accommodates the skipping of words by allowing shift operations to be performed from inactive state nodes of the Graph Structured Stack. A heuristic similar to beam search makes the algorithm computationally tractable.

The modified parser, GLR*, has been implemented and integrated with the latest version of the Generalized LR Parser/Compiler (Tomita *et al.*, (1988), Tomita, (1990)).

We discuss an application of the GLR* parser to spontaneous speech understanding and present some preliminary tests on the utility of the GLR* parser in such settings.

1. Introduction

In this chapter, we introduce a technique for substantially increasing the robustness of parsers to two particular types of extra-grammaticality: noise in the input, and limited grammar coverage. Both phenomena cause a common situation, where the input contains words or fragments that are unparsable. The distinction between these two types of extra-grammaticality is based to a large extent upon whether or not the unparsable fragment, in its context, can be considered grammatical by a linguistic judgment. This distinction may indeed be vague at times, and of limited practical importance.

Our approach to the problem is to enable the parser to overcome these forms of extra-grammaticality by ignoring the unparsable words and fragments and focusing on the maximal subset of the input that is covered by the grammar. Although presented and implemented as an enhancement to the Generalized LR parsing (GLR) paradigm, our technique should be applicable to most phrase-structured based parsing formalisms. However, the efficiency of our parser is due in part to several particular properties of GLR parsing, to which there may not be direct equivalents in other parsing formalisms.

The problem can be formalized in the following way: Given a context-free grammar G and a sentence S, find and parse S' - the largest subset of words of S, such that $S' \in L(G)$.

A naive approach to this problem is to exhaustively list and attempt to parse all possible subsets of the input string. The largest subset can then be selected from among the subsets that are found to be parsable. Such an algorithm is clearly computationally infeasible, since the number of subsets is exponential in the length of the input string. We thus devise an efficient method for accomplishing the same task, and pair it with an efficient search approximation heuristic that maintains runtime feasibility.

The algorithm described in this chapter, which we have named GLR*, is a modification of the Generalized LR (Tomita) parsing algorithm. It has been implemented in Lucid Common Lisp as an updated version of the GLR Parser/Compiler (Tomita *et al.*, (1988), Tomita, (1990)).

There have been several other approaches to robust parsing, most of which have been special purpose algorithms. Some of these approaches have abandoned syntax as a major tool in handling extra-grammaticalities and have focused on domain dependent semantic methods (Carbonell and Hayes, (1984), Ward, (1991)). Other systems have constructed grammar and domain dependent fall-back components to handle extra-grammatical input that causes the main parser to fail (Stallard and Bobrow, (1992), Seneff, (1992)).

Our approach can be viewed as an attempt to extract from the input the maximal grammatical structure that is possible, within a domain independent setting. Because the GLR* parsing algorithm is an enhancement to the standard GLR context-free parsing algorithm, all of the techniques and grammars developed for the standard parser can be applied as they are. In particular, the standard LR parsing tables are compiled in advance from the grammar and used "as is" by the parser in runtime. The GLR* parser inherits the benefits of the original parser in terms of ease of grammar development, and, to a large extent, efficiency properties of the parser itself. In the case that the input sentence is by itself grammatical, GLR* returns the exact same parse as the standard GLR parser.

The remaining sections of the chapter are organized in the following way: Section 2 presents an outline of the basic GLR* algorithm itself, followed by a detailed example of the operation of the parser on a simple input string. In section 3 we discuss the search heuristic that is added to the basic GLR* algorithm, in order to ensure its runtime feasibility. We discuss an application of the GLR* algorithm to spontaneous speech understanding, and present some preliminary test results in section 4. Finally, our conclusions and further research directions are presented in section 5.

2. The GLR* Parsing Algorithm

In this section, we provide an informal outline of the basic GLR* parsing algorithm. This algorithm is designed to find and parse all possible grammatical substrings of a given input. As an extension of the GLR parsing algorithm, GLR* uses the same data structures. Particularly, GLR* uses the same parsing tables that are pre-compiled from the grammar. It is only the run-time parsing part of the algorithm that is different.

The key difference between GLR and GLR* is in their use of the Graph Structured Stack (GSS). GLR uses the GSS as an extended version of a stack, where only the active nodes at the top of the stacks are accessible. GLR*, on the other hand, views the GSS as a chart that records partial parses and the state of the parser when they were created. GLR* thus maintains a structure representing the complete GSS throughout the parsing process, and

allows access to all nodes in this structure, not merely the active ones.

GLR* pursues the parses of all possible subsets of the input by allowing the parser to skip over words of the input in a controlled way. With an LR style parser that uses lookaheads into the input, allowing the parser to skip over words has implications on the lookaheads used by the parser, which complicates the algorithm considerably. However, a number of studies have demonstrated that with practical natural language grammars, using a GLR parser with extended lookaheads can in fact damage the parser's efficiency (Billot and Lang, (1989)). For these reasons, GLR* was designed to work with no lookaheads. It therefore uses LR(0) parsing tables.

GLR* accommodates skipping words of the input string by allowing shift operations to be performed from inactive state nodes in the GSS. Shifting an input symbol from an inactive state is equivalent to skipping the words of the input that were encountered after the parser reached the inactive state and prior to the current word being shifted. Since the parser is LR(0), reduce operations need not be repeated for skipped words (the reductions do not depend on any lookahead). Information about skipped words is maintained in the symbol nodes that represent parse sub-trees.

2.1. INFORMAL OUTLINE OF THE BASIC GLR* ALGORITHM

We now present an informal high-level outline of the basic GLR* parsing algorithm. Similar to the GLR algorithm, GLR* parses the input on a single left-to-right scan of the input, processing one word at a time. The processing of each input word is called *a stage*. Each stage consists of two main *phases*, a *reduce phase* and a *shift phase*. The reduce phase always precedes the shift phase. The outline of each stage of the algorithm is shown in Figure 1. `RACT(st)` denotes the set of reduce actions defined in the parsing table for state *st*. Similarly, `SACT(st,x)` denotes the shift actions defined for state *st* and symbol x, and `AACT(st,x)` denotes the accept action.

The last stage of the algorithm, in which the end-of-input symbol "$" is processed, is somewhat different. The `READ`, `DISTRIBUTE-REDUCE` and `REDUCE` steps are identical to those of previous stages. However, a `DISTRIBUTE-ACCEPT` replaces the `DISTRIBUTE-SHIFT` step. In the `DISTRIBUTE-ACCEPT` step, accept actions are distributed to all state nodes *st* in the GSS (active and inactive), for which `AACT(st,$)` is true. Pointers to these state nodes are then collected into a list of final state nodes. If this list is not empty, GLR* accepts the input, otherwise, the input is rejected. Finally, a `GET-PARSES` step creates and returns a list of all symbol nodes that are direct descendents of the final state nodes. These symbol nodes are the roots of the parse forest.

2.2. AN EXAMPLE

To clarify how the GLR* algorithm actually works, we present a step by step runtime example. Figure 2 contains a simple natural language grammar that we use for this example. The terminal symbols of the grammar are depicted in lower-case, while the non-terminals are in upper-case. The grammar is compiled into an SLR(0) parsing table, which is displayed in Table 1. Note that since the table is SLR(0), the reduce actions are independent of any lookahead. The actions on states 10 and 11 include both a shift and a reduce.

To understand the operation of the parser, we now follow some steps of the GLR* parsing algorithm on the input `x = det n v n det p n`. This input is ungrammatical due to the second "`det`" token. The maximal parsable subset of the input in this case is the string that

```
(1) READ:
    Read the next input token x.

(2) DISTRIBUTE-REDUCE:
    For each active state node st, get the set of reduce actions RACT(st) in
    the parsing table, and attach it to the active state node.

(3) REDUCE:
    Perform all reduce actions attached to active state nodes. Recursively
    perform reductions after distributing reduce actions to new active state
    nodes that result from previous reductions.

(4) DISTRIBUTE-SHIFT:
    For each state node st in the GSS (active and inactive), get SACT(st,x)
    from the parsing table. If the action is defined attach it to the state
    node.

(5) SHIFT:
    Perform all shift actions attached to state nodes of the GSS.

(6) MERGE:
    Merge active state nodes of identical states into a single active state
    node.
```

Figure 1: Outline of a Stage of the Basic GLR* Parsing Algorithm

includes all words other than this second "`det`".

In the figures ahead, we graphically display the GSS constructed by the parser in various stages of the parsing process. The following notation is used in these figures:

- An *active* state node is represented by a black circle with the state number indicated above it. Actions attached to the node are marked on the right.
- An *inactive* state node is represented by a clear circle. The state number is indicated above the node and attached actions are indicated above the state number.
- Symbol nodes are represented as squares. The symbol label is marked above the node.
- *Reduce* actions are denoted by **r***n*, where *n* is the index number of the grammar rule.
- *Shift* actions are denoted by **sh***n*, where *n* is the new state into which the parser moves after the shift action.

We follow the GSS of the parser prior to the reduce and shift phases of each stage of the algorithm, while processing the input string. The initial GSS contains the single active state node of state 0. Since there are no reduce actions from state 0, the first reduce phase is empty. With the first input token being "`det`", the `DISTRIBUTE-SHIFT` step attaches the action "sh3" to state node 0. Figure 3 shows the GSS following this `DISTRIBUTE-SHIFT` step and just prior to the `SHIFT` step of stage 1.

In the following `SHIFT` step, the "`det`" is shifted, a symbol node is created and the parser moves into a new active state node of state 3. The algorithm then proceeds to the next stage to process the next input token "`n`". Since there are no reduce actions from state 3, the

(1) S → NP VP
(2) NP → det n
(3) NP → n
(4) NP → NP PP
(5) VP → v NP
(6) PP → p NP

Figure 2: A Simple Natural Language Grammar

	Reduce	Shift					Goto			
State		det	n	v	p	$	NP	VP	PP	S
0		sh3	sh4				2			1
1						acc				
2				sh7	sh8			5	6	
3			sh9							
4	r3									
5	r1									
6	r4									
7		sh3	sh4				10			
8		sh3	sh4				11			
9	r2									
10	r5				sh8				6	
11	r6				sh8				6	

Table 1: SLR(0) Parsing Table for Grammar in Figure 1

reduce phase of this stage is empty. In the following `DISTRIBUTE-SHIFT` step, shift actions are distributed by the algorithm to both the active node of state 3 and the inactive node of state 0. Figure 4 shows the GSS after this step, just prior to the `SHIFT` step of stage 2.

In the following `SHIFT` step, the input token "`n`" is shifted from both state nodes, creating new active state nodes of states 9 and 4. The shifting of the input token "`n`" from state 0 corresponds to a parse in which the first input token "`det`" is skipped. Stage 3 begins to process the next input token "`v`". First, in the `DISTRIBUTE-REDUCE` step, reduce actions are distributed to the existing active nodes. Figure 5 shows the GSS at this point in time, just prior to the `REDUCE` step of stage 3.

The following `REDUCE` step reduces both branches into noun phrases. The two "`NP`"s are packed together by a local ambiguity packing procedure. In the following `DISTRIBUTE-SHIFT` step, shift actions for the input token "`v`" are distributed to all the state nodes. However, in this case, only state 2 allows a shift of "`v`" (into state 7). The resulting GSS, prior to the `SHIFT` step of stage 3, is displayed in Figure 6.

The following `SHIFT` step creates an active state node of state 7. The next stage begins by reading the next input token "`n`". The state 7 node is the only active node at this point. Since no reduce actions are specified for this state, the following reduce phase is empty. Shift actions are then distributed in the `DISTRIBUTE-SHIFT` step. The resulting GSS is shown in Figure 7.

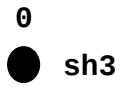

Figure 3: GSS Prior to SHIFT Step of Stage 1

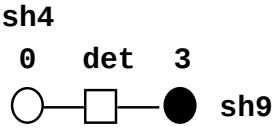

Figure 4: GSS Prior to SHIFT Step of Stage 2

The subsequent `SHIFT` step then shifts the input token "`n`" from the three nodes that were marked with shift actions. The next input token is "`det`". Figure 8 shows the GSS just prior to the next `REDUCE` step and Figure 9 after the `REDUCE` and `DISTRIBUTE-SHIFT` steps, just before the `SHIFT` step. Note that the current input token "`det`" cannot be shifted from either of the two active state nodes at this point. A GLR parser would have thus failed at this point. However, the GLR* algorithm succeeds in distributing the shift actions to two inactive state nodes in this case. The token "`det`" is then shifted from these two nodes.

The next stage then begins, and the input token "`p`" is read. Since no reduce actions can be distributed to the active state node 3, the reduce phase is empty. `DISTRIBUTE-SHIFT` then distributes shift actions to all state nodes. The GSS at this point is shown in Figure 10. Note that once again, shift actions could not be distributed to the active state node, since the grammar does not allow "`p`" to follow "`det`". However, a shift action was distributed to the node of state 10. This branch skips over the previous "`det`", and eventually leads to the desired maximal parse.

For the sake of brevity we do not continue to further follow the parsing step by step. The final GSS, following the `DISTRIBUTE-ACCEPT` step, is displayed in Figure 11. Several different parses, corresponding to different subsets of skipped words are actually packed into the "`S`" symbol node seen at the bottom of the figure. The parse that corresponds to the maximal subset of the input is one in which the second "`det`" is the only word that was skipped.

2.3. EFFICIENCY OF THE PARSER

Efficiency of the parser is achieved by a number of different techniques. The most important of these is a sophisticated process of local ambiguity packing and pruning. A local ambiguity is a part of the input sentence that corresponds to a phrase (thus, reducible to some non-terminal symbol of the grammar), and is parsable in more than one way. The process of skipping words creates a large number of local ambiguities. For example, the grammar in Figure 2 allows both determined and undetermined noun phrases (rules 2 and 3). As seen in the example presented earlier, this results in the creation of two different noun phrase symbol nodes for the initial fragment "`det n`". The first node is created for the full phrase after a reduction according to the first rule. A second symbol node is created when the determiner is skipped and a reduction by the second rule takes place.

Locally ambiguous symbol nodes are detected as nodes that are surrounded by common state nodes in the GSS. The original GLR parser detects such local ambiguities and packs them into a single symbol node. This procedure was extended in the GLR* parser. Locally

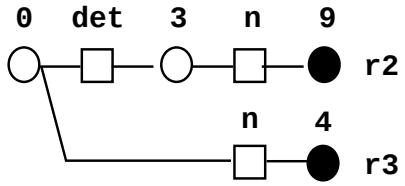

Figure 5: GSS Prior to REDUCE Step of Stage 3

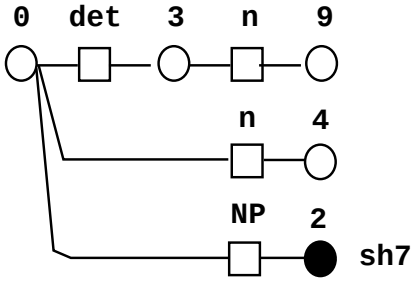

Figure 6: GSS Prior to SHIFT Step of Stage 3

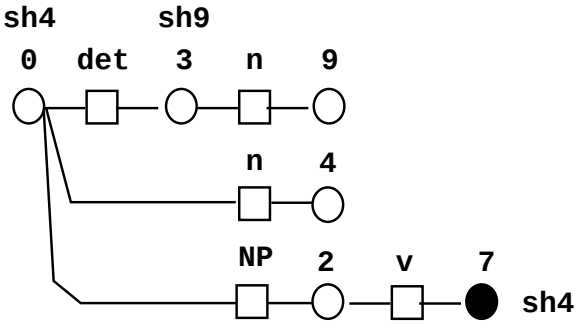

Figure 7: GSS Prior to SHIFT Step of Stage 4

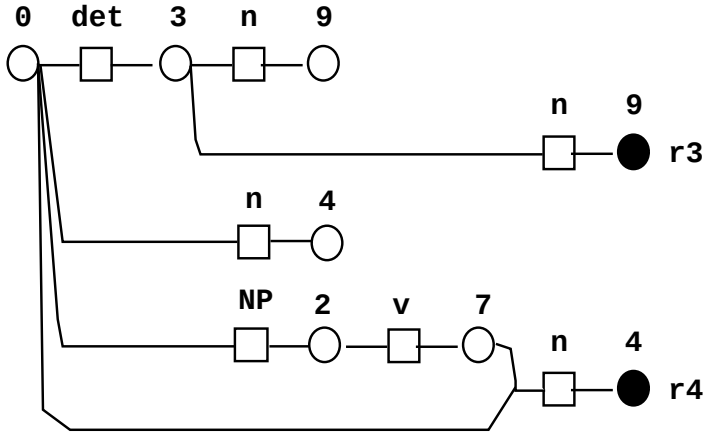

Figure 8: GSS Prior to REDUCE Step of Stage 5

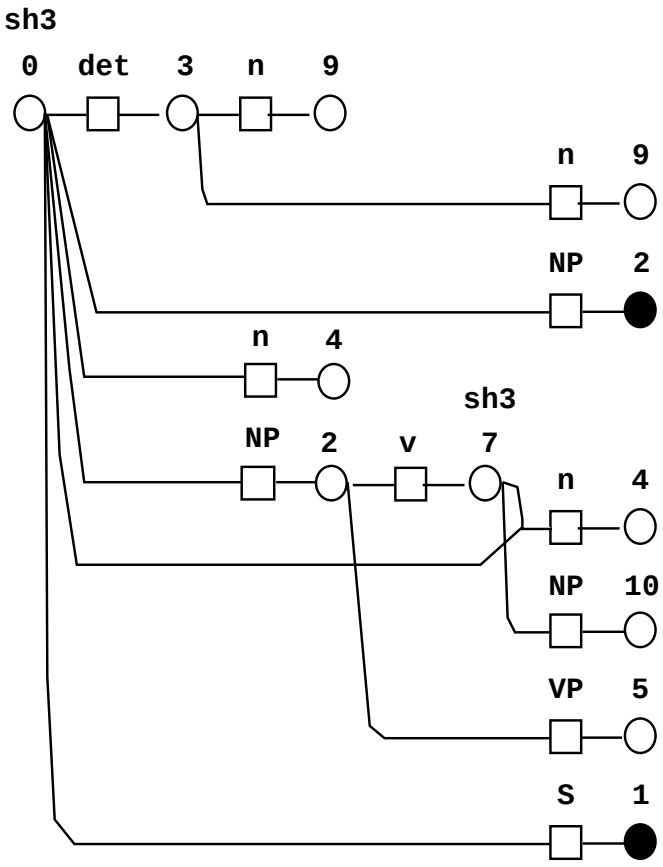

Figure 9: GSS Prior to SHIFT Step of Stage 5

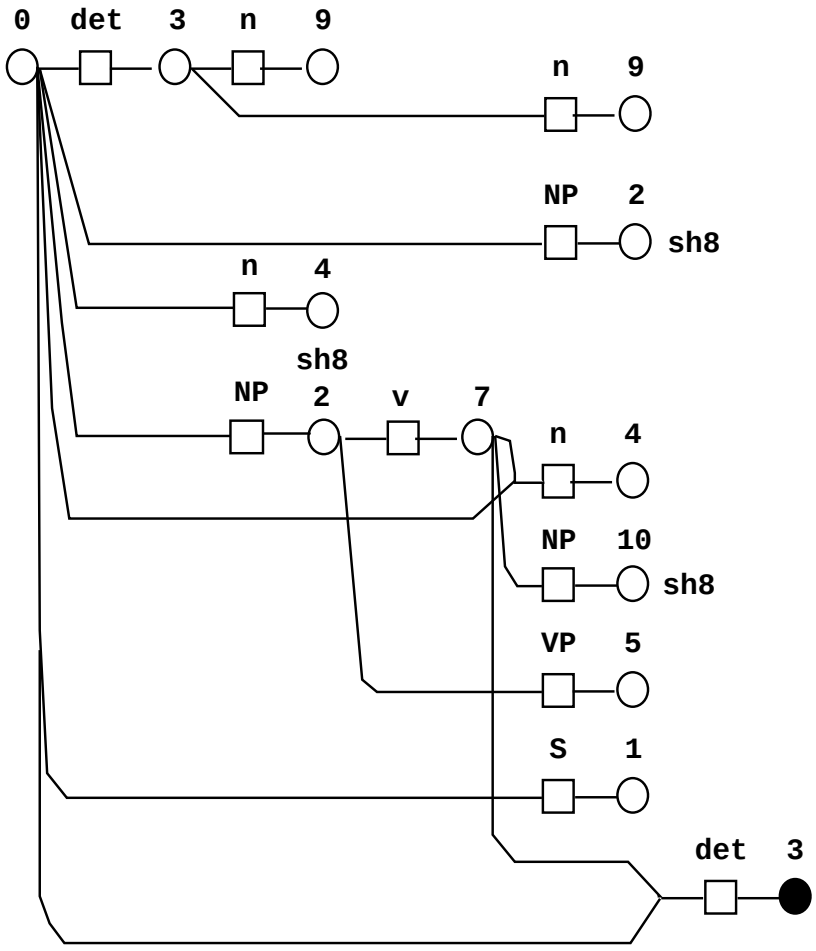

Figure 10: GSS Prior to SHIFT Step of Stage 6

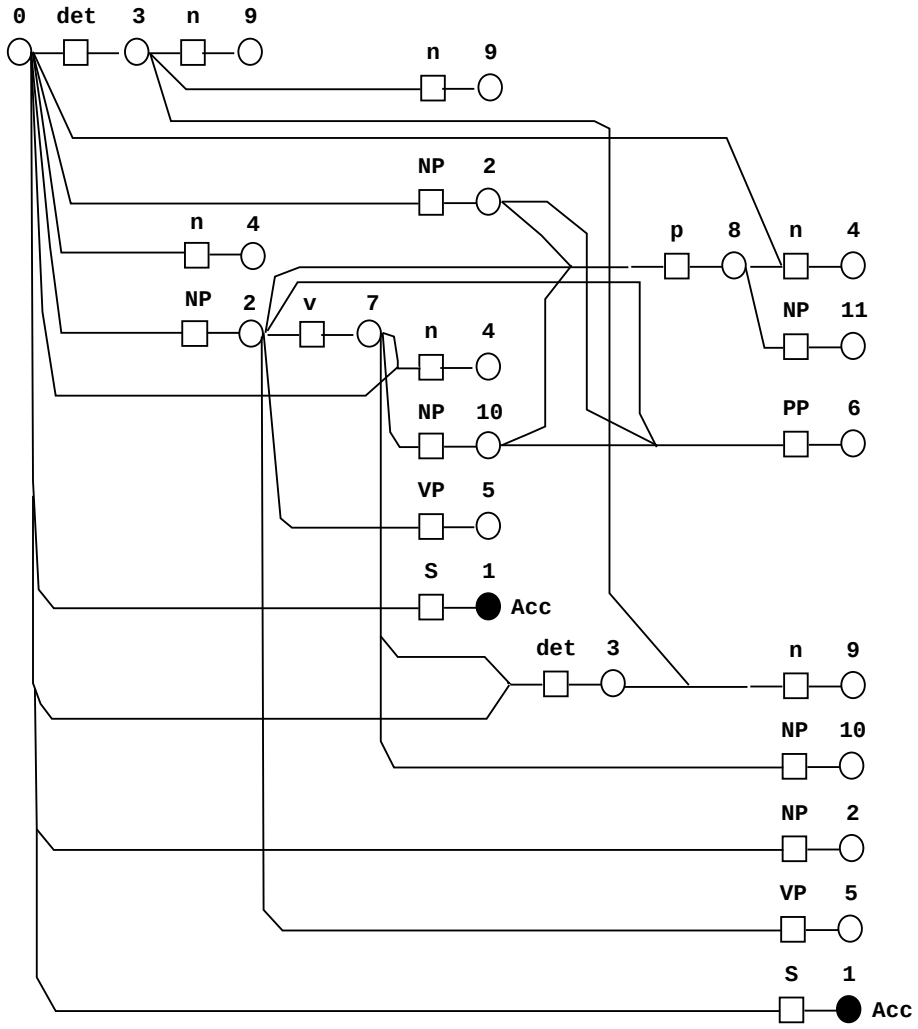

Figure 11: Final GSS after DISTRIBUTE-ACCEPT Step

ambiguous symbol nodes are compared in terms of the words skipped within them. In cases such as the example described above, where one phrase has more skipped words than the other, the phrase with more skipped words is discarded in favor of the more complete parsed phrase. This operation drastically reduces the number of parses being pursued by the parser.

2.4. SELECTING THE BEST MAXIMAL PARSE

The basic GLR* parsing algorithm constructs parses for all parsable subsets of a given input sentence. In principle, we are only interested in finding the maximal parsable subset of the input string (and its parse). However, in many cases there are several distinct maximal parses, each consisting of a different subset of words of the original sentence. Additionally, there are cases where a parse that is not maximal in terms of the number of words skipped may be deemed preferable.

To select the "best" parse from the set of parses returned by the parser, we use a scoring procedure that ranks each of the parses found. We then select the parse that was ranked best [1]. Presently, our scoring procedure is rather simple. It takes into account the number of words skipped and the fragmentation of the parse. Both measures are weighed equally. Thus a parse that skipped one word but parsed the remaining input as a single sentence is preferred over a parse that fragments the input into three sentences, without skipping any input word.

We are in the process of enhancing this simple scoring mechanism. We plan on adding to our scoring function several additional heuristic measures that will reflect various syntactic and semantic properties of the parse tree. We will measure the effectiveness of our enhanced scoring function in ranking the parse results by their desirability.

3. The Beam Search Heuristic

Although implemented efficiently, the basic GLR* parser is still not guaranteed to have a feasible running time. The basic GLR* algorithm described earlier computes parses of all parsable subsets of the original input string, the number of which is potentially exponential in the length of the input string. We are interested in finding parses of maximal subsets of the input string (or almost maximal subsets). We have therefore developed and added to the parser a heuristic that prunes parsing options that are not likely to produce a maximal parse. This process has been traditionally called "beam search".

A direct way of adding a beam search to the parser would be to limit the number of active state nodes pursued by the parser at each stage, and continue processing only active nodes that are most promising in terms of the number of skipped words associated with them. However, the structure of the GSS makes it difficult to associate information on skipped words directly with the state nodes [2]. We have therefore opted to implement a somewhat different heuristic that has a similar effect.

Since the skipping of words is the result of performing shift operations from inactive state nodes of the GSS, our heuristic limits the number of inactive state nodes from which a input symbol is shifted. At each shift stage, shift actions are first distributed to the active

[1] The system will display the n best parses found, where the parameter n is controlled by the user at runtime. By default, we set n to one, and the highest ranking parse is displayed.

[2] This is due to the fact that state nodes are merged. Therefore, a state node may be common to several different sub-parses, with different skipped words associated with each sub-parse.

state nodes of the GSS. This corresponds to not skipping any words at this stage. If the number of state nodes that allow a shift operation at this point is less than a predetermined constant threshold (the "beam-limit"), then shift operations from inactive state nodes are also considered. Inactive states are processed in an ordered fashion, so that shifting from a more recent state node that will result in fewer skipped words is considered first. Shift operations are distributed to inactive state nodes in this way until the number of shifts distributed reaches the threshold.

Using the beam search heuristic reduces the runtime of the GLR* parser to within a constant factor of the original GLR parser. Although it is not guaranteed to find the desired maximal parsable subset of the input string, our preliminary tests have shown that it works well in practice.

The threshold (beam-limit) itself is a parameter that can be dynamically set to any constant value at runtime. Setting the beam-limit to a value of 0 disallows shifting from inactive states all together, which is equivalent to the original GLR parser. In preliminary experiments that we have conducted (see next section) we have achieved good results with a setting of the beam-limit to values in the range of 5 to 10. There exists a direct tradeoff between the value of the beam-limit and the runtime of the GLR* parser. With a set value of 5, our tests have indicated a runtime that is within a factor of 2-3 times that of the original GLR parser, which amounts to a parse time of only several seconds on sentences that are up to 30 words long.

4. Parsing of Spontaneous Speech Using GLR*

4.1. THE PROBLEM OF PARSING SPONTANEOUS SPEECH

As a form of input, spontaneous speech is full of noise and irrelevances that surround the meaningful words of the utterance. Some types of noise can be detected and filtered out by speech recognizers that process the speech signal. A parser that is designed to successfully process speech recognized input must however be robust to various forms of noise, and be able to weed out the meaningful words from the rest of the utterance.

When parsing spontaneous spoken input that was recognized by a speech recognition system, the parser must deal with three major types of extra-grammaticality:

- Noise due to the spontaneity of the speaker, such as repeated words, false beginnings, stuttering, and filled pauses (i.e. "ah", "um", etc.).
- Ungrammaticality that is due to the language of the speaker, or to the coverage of the grammar.
- Errors of the speech recognizer.

We have conducted two preliminary experiments to evaluate the GLR* parser's ability to overcome the first two types of extra-grammaticality. We are in the process of experimenting with the GLR* parser on actual speech recognized output, in order to test its capabilities in handling errors produced by the speech recognizer.

4.2. PARSING OF NOISY SPONTANEOUS SPEECH

The first test we conducted was intended to evaluate the performance of the GLR* parser on noisy sentences typical of spontaneous speech. The parser was tested on a set of 100 sentences of transcribed spontaneous speech dialogues in a conference registration domain. The input is hand-coded transcribed text, not processed through any speech recognizer. The

	Robust Parser
Parsable	99
Unparsable	1
Good/Close Parses	77
Bad Parses	22

Table 2: Performance of the GLR* Parser on Spontaneous Speech

grammar used was an upgraded version of a grammar for the conference registration task, developed and used by the JANUS speech-to-speech translation project at CMU [Waibel et al. 1991]. Since the test sentences were drawn from actual speech transcriptions, they were not guaranteed to be covered by the grammar. However, since the test was meant to focus on spontaneous noise, sentences that included verbs and nouns that were beyond the vocabulary of the system were avoided. Also pruned out of the test set were short opening and closing sentences (such as "hello" and "goodbye"). The transcriptions include a multitude of noise in the input. The following example is one of the sentences from this test set:

```
"fckn2_10 /ls/ /h#/ um okay {comma}
then yeah I am disappointed {comma}
*pause* but uh that is okay {period}"
```

The performance results are presented in Table 2. Note that due to the noise contaminating the input, the original parser is unable to parse a single one of the sentences in this test set. The GLR* parser succeeded to return some parse result in all but one of the test sentences. However, since returning a parse result does not by itself guarantee an analysis that adequately reflects the meaning of the original utterance, we reviewed the parse results by hand, and classified them into the categories of "good/close" and "bad" parses. The results of this classification are included in the table.

4.3. GRAMMAR COVERAGE

We conducted a second experiment aimed exclusively on evaluating the ability of the GLR* parser to overcome limited grammar coverage. In this experiment, we compared the results of the GLR* parser with those of the original GLR parser on a common set of sentences using the same grammar. We used the grammar from the spontaneous speech experiment for this test as well. The common test set was a set of 117 sentences from the conference registration task of the JANUS project. These sentences are simple synthesized text sentences. They contain no spontaneous speech noise, and are not the result of any speech recognition processing. Once again, to evaluate the quality of the parse results returned by the parser, we classified the parse results of both parsers by hand into two categories: "good/close parses" and "bad parses". The results of the experiment are presented in Table 3.

The results indicate that using the GLR* parser results in a significant improvement in performance. The percentage of sentences, for which the parser returned good or close parses increased from 52% to 70%, an increase of 18%. Fully 97% of the test sentences (all but 3) are parsable by the GLR* parser, an increase of 36% over the original parser. However, this includes a significant increase (from 9% to 27%) in the number of bad parses found. Thus, fully half of the additional parsable sentences of the set return with parses that may be deemed bad.

	Original Parser		Robust Parser	
	number	percent	number	percent
Parsable	71	61%	114	97%
Unparsable	46	39%	3	3%
Good/Close Parses	61	52%	82	70%
Bad Parses	10	9%	32	27%

Table 3: Performance of the GLR* Parser vs. the Original Parser

The results of the two experiments clearly point to the following problem: Compared with the GLR* parser, the original GLR parser, although fragile, returned results of relatively good quality, when it succeeded in parsing the input. The GLR* parser, on the other hand, will succeed in parsing almost any input, but this parse result may be of little or no value in a significant portion of cases. This indicates a strong need for the development of methods for discriminating between good and bad parse results. We intend to try and develop some effective heuristics to deal with this problem. The problem is also due in part to the ineffectiveness of the simple heuristics currently employed for selecting the best parse result from among the large set of parses returned by the parser. As mentioned earlier, we are currently working on developing more sophisticated and effective heuristics for selecting the best parse.

5. Conclusions and Future Research Directions

Motivated by the difficulties that standard syntactic parses have in dealing with extra-grammaticalities, we have developed GLR*, an enhanced version of the standard Generalized LR parser, that can effectively handle two particular problems that are typical of parsing spontaneous speech: noise contamination and limited grammar coverage.

Given a grammar G and and input string S, GLR* finds and parses S', the maximal subset of words of S, such that S' is in the language $L(G)$. The parsing algorithm accommodates the skipping of words and fragments of the input string by allowing shift operations to be performed from inactive states of the GSS (as well as from the active states, as is done by the standard parser). The algorithm is coupled with a beam-search-like heuristic, that controls the process of shifting from inactive states to a limited beam, and maintains computational tractability.

Most other approaches to robust parsing have suffered to some extent from a lack of generality and from being domain dependent. Our approach, although limited to handling only certain types of extra-grammaticality, is general and domain independent. It attempts to maximize the robustness of the parser within a purely grammatical setting. Because the GLR* parsing algorithm is a modification of the standard GLR context-free parsing algorithm, all of the techniques and grammars developed for the standard parser can be applied as they are. In the case that the input sentence is by itself grammatical, GLR* returns the same parse as the standard GLR parser. The techniques used in the enhancement of the standard GLR parser into the robust GLR* parser are in principle applicable to other phrase-structure based parsers.

Our preliminary experiments on the effectiveness of the GLR* parser in handling noise contamination and limited grammar coverage have produced encouraging results. However, they have also pointed out a need for developing effective heuristics that can select the best parse result from a potentially large set of possibilities produced by the parser. Since the

GLR* parser is likely to succeed in producing some parse in practically all cases, successful parsing by itself can no longer be an indicator to the value and quality of the parse result. We are therefore developing additional heuristics for evaluating the quality of the selected best parse.

We are in the process of conducting extensive experiments with speech recognized input to evaluate our system and guide its further development. We are also investigating the potential of the GLR* parser in several other application areas and domains.

References

Billot, S. and Lang, B., 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, June 1989.

Carbonell, J. G. and Hayes, P. J., Recovery Strategies for Parsing Extragrammatical Language. *Technical Report CMU-CS-84-107*, 1984.

Seneff, S., A relaxation method for understanding spontaneous speech utterances. In *Proceedings of DARPA Speech and Natural Language Workshop*, pages 299–304, February 1992.

Stallard, D. and Bobrow, R., Fragment processing in the DELPHI system. In *Proceedings of DARPA Speech and Natural Language Workshop*, pages 305–310, February 1992.

Tomita, M., Mitamura, T., Musha, H., and Kee, M., The Generalized LR Parser/Compiler - Version 8.1: User's Guide. *Technical Report CMU-CMT-88-MEMO*, 1988.

Tomita, M., *Efficient Parsing for Natural Language*. Kluwer Academic Publishers, Hingham, Ma., 1986.

Tomita, M., The Generalized LR Parser/Compiler - Version 8.4. In *Proceedings of International Conference on Computational Linguistics (COLING-90)*, pages 59–63, Helsinki, Finland, 1990.

Ward, W., Understanding spontaneous speech: The Phoenix system. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 365–367, April 1991.