Bi-directional LR Parsing from an Anchor Word for Speech Recognition

Hiroaki Saito

Center for Machine Translation
Carnegie Mellon University
Pittsburgh, PA 15213, USA

Net Address: saito+@cs.cmu.edu

Abstract

This paper introduces a new technique of parsing sentences from an arbitrary word which is highly reliable or semantically important. This technique adopts an efficient LR parsing method and uses a *reverse LR table* constructed besides a standard LR table. This technique is particularly suitable in parsing a lattice of words hypothesized by a speech recognition module. If we choose anchor symbols in such a way that they are almost always acoustically reliable, the bi-directional LR parsing performs better against misrecognized words than the regular left-to-right LR parser, while most of the LR efficiency is preserved. A pilot implementation shows a 43 % reduction of the error rate against the left-to-right LR method in parsing the speech input.

1. Introduction

Parsing a word lattice produced by a speech recognition module requires much more search than conventional sentence parsing, and therefore an extremely efficient parsing algorithm is needed. A word lattice is a set of words hypothesized by a speech recognition system from an utterance. A typical word lattice consists of 30 - 200 words for a 10 word utterance, and each word has a score indicating probability of its having been actually uttered. Not only are there many junk words which were never uttered, some actually uttered words may not be present in the lattice (missing words).

An island growing parsing in ATN mechanism presented the serious maintenance and practical problems [10]. The first promising attempt to parse an incomplete word lattice was made by Hayes *et al.* [2], using semantic caseframes. This attempt revealed that, while the semantic caseframes can provide a reasonable degree of robustness, a very efficient algorithm is required to be practical. Good efforts were made by Poesio *et al.* [4] and Giachin *et al.* [1] to make the semantic caseframe approach more efficient and robust. Meanwhile, Tomita modified the generalized LR parsing algorithm (GLR) [8] to handle word lattices [9]. The GLR algorithm is a very efficient, table-driven, non-deterministic context-free parsing algorithm, and it has been applied in speech recognition projects with further modification of the algorithm to handle missing words [5]. It requires heavy search, however, especially when a word is missed in the beginning part of the utterance, since the parser guesses missing words only from its left context. Thus, the strict left-to-right-ness sometimes suffers inefficiency, and it is desired to parse occasionally backwards from an acoustically reliable word called an anchor word [10]. Bidirectionality also plays an important role in Head-Driven parsing and a method of bi-directional parsing was presented by Satta et al [7].

This paper describes a technique, called *bi-directional GLR parsing*, to parse a word lattice occasionally backwards without loss of the table-driven efficiency. A *reverse LR table* is constructed as well as a standard LR table. Section 2 reviews the generalized LR parsing algorithm. Section 3 then describes how to construct reverse LR tables and how to use them in word lattice parsing. Section 4 discusses the robustness of bi-directional GLR parsing, and finally concluding remarks are made in Section 5.

2. Background: Generalized LR Parsing

The LR parsing technique was originally developed for compilers of programming languages and has been extended for Natural Language Processing [8]. The LR parsing analyzes the input sequence from left to right with no backtracking by looking at the parsing table constructed from the context-free grammar rules in advance. An example grammar and its parsing table are shown in Figure 2-1 and Figure 2-2 respectively.

Entries "s *n*" in the action table (the left part of the table) indicate the action "shift one word from input buffer onto the stack and go to state *n*". Entries "r *n*" indicate the action "reduce constituents on the stack using rule *n*". The entry "acc" stands for the action "accept", and blank spaces represent "error". '$' in the action table is the end-of-input symbol. The goto table (the right part of the table) decides to which state the parser should go after a reduce action. The LR parsing table in Figure 2-2 is different from the regular LR tables utilized by compilers of programming

languages in that there are multiple entries, called *conflicts*, on the row of state 9. While the encountered entry has only one action, parsing proceeds exactly the same way as the normal LR parsing. In case there are multiple actions in one entry, it executes all the actions with the *graph-structured stack* [8]. The bi-directional GLR parsing method begins at an arbitrary spot of the input, while the conventional GLR parsing analyzes the input sequence only from left to right.

```
---------------------------
   (1)  S    -->  NP VP
   (2)  NP   -->  n
   (3)  NP   -->  NP PP
   (4)  VP   -->  v NP
   (5)  PP   -->  p NP
---------------------------
```

Figure 2-1: An Example Ambiguous Grammar

	Action Table				Goto Table			
	n	v	p	$	NP	VP	PP	S
1	s2				3			4
2		r2	r2	r2				
3		s6	s5			7	8	
4				acc				
5	s2				9			
6	s2				10			
7				r1				
8		r3	r3	r3				
9		r5	r5,s5	r5			8	
10			s5	r4			8	

Figure 2-2: Generalized LR Parsing Table

3. Bi-directional GLR parsing

In this section we describe the bi-directional GLR parsing algorithm and an example of parsing a word lattice.

3.1. Reverse LR table

Bi-directional GLR parsing uses a *reverse LR table* besides a standard LR table. The reverse LR table is constructed from the context-free grammar in which the order of right-hand-side symbols is reversed in each rule. For example, the grammar in Figure 3-1 is the set of reverse rules built from the example grammar in Figure 2-1. Its parsing table (Figure 3-2), which is a reverse LR table, is constructed from the reversed grammar in Figure 3-1.

```
---------------------------
   (1)  S    -->  VP NP
   (2)  NP   -->  n
   (3)  NP   -->  PP NP
   (4)  VP   -->  NP v
   (5)  PP   -->  NP p
---------------------------
```

Figure 3-1: Reversed Grammar

	Action Table				Goto Table			
	n	v	p	$	NP	VP	PP	S
1	s3				2	5	4	6
2		s7	s8					
3		r2	r2	r2				
4	s3				9		4	
5	s3				10		4	
6				acc				
7	r4							
8	r5							
9		r3	s8,r3	r3				
10			s8	r1				

Figure 3-2: Reverse LR Table for Right-to-left Parsing

3.2. Parsing from the Anchor Word in Both Directions

Here we describe the algorithm for parsing the lattice starting from an anchor symbol and expanding in both left and right directions.

Parsing Procedure:

1. Choose the anchor symbol A from the lattice.
2. Because A is a terminal symbol, the initial state(s) are determined from the action table. Note that only the states in which the shift action(s) are performed are valid. There are two kinds of starting states:
   - initial states for left-to-right parsing from the standard LR table
   - initial states for right-to-left parsing from the reverse LR table

   Start GLR parsing from the initial states in both directions independently until the reduce action is suspended due to the lack of the reduce constituents. (Since the parsing starts in the middle of the input, this could happen unless A is located on the edge of the lattice.) The standard LR table is used when the parsing proceeds from left to right and the reverse LR table is used when the parse proceeds in the opposite direction.
3. Perform the suspended reduce action when the same number reduce action from the other direction is ready.

Here we show how this procedure works in parsing the

lattice in Figure 3-3 using the grammars and the tables in Figures 2-1, 2-2, 3-1 and 3-2. In parsing a lattice, the juncture verifier JUNCT(W_i, W_j) should be prepared which returns TRUE if W_i and W_j can abut.[1]

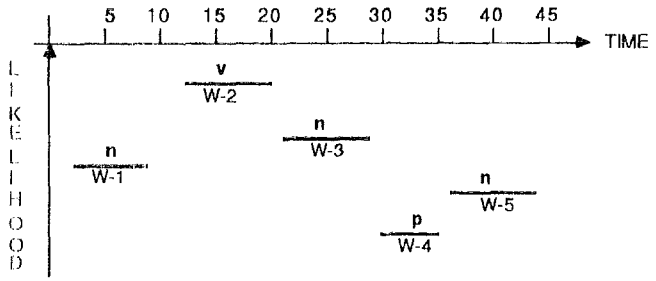

Figure 3-3: Word Lattice

First we choose the most probable word from the lattice, i.e. W-2 (v). The standard LR table indicates that v is expected at the states 2, 3, 8, and 9. Only the state 3 is valid because the other states require reduce actions which need previous words. Thus the parse starts from state 3. The current word v is shifted and the next state 6 is determined which is expecting n. Figure 3-4 shows this situation.

We consult the reverse LR table in the same way. Namely the right-to-left parse starts from the state 2 and the next state 7 is decided after v is shifted. (Figure 3-5. States numbers and the expecting terminals for the left-bound parsing are written in italic fonts with underscore bars.)

Here we perform the right-to-left parse first. State 7 is ready for the reduce action 4 by n. But the action "reduce 4" can not be performed now even on the assumption that JUNCT(W-1, W-2) returns TRUE, because the current stack does not contain enough reduce constituents. That means the reduce action 4 is suspended until the left-to-right parsing is ready for the reduce action 4.

Therefore we proceed with the right-bound parsing now. W-3 (n) is expected by state 6. On the assumption that JUNCT(W-2, W-3) returns TRUE, n is shifted and the new state 2 is determined from the left-to-right action table (Figure 3-6).

The new state 2 is ready for the reduce action 2 (NP --> n) by v, p, $. On the assumption that JUNCT(W-3, W-4) returns TRUE, this reduce action is performed. The left-to-right goto table indicates that the new state is 10. (Figure 3-7)

The next word W-4 is expected by state 10. On the assumption that JUNCT(W-3, W-4) returns TRUE, W-4 is shifted and the new state 5 is determined (Figure 3-8).

Figure 3 - 4

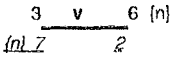

Figure 3 - 5

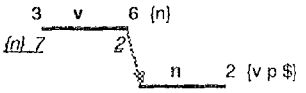

Figure 3 - 6

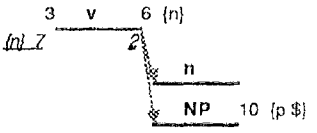

Figure 3-7

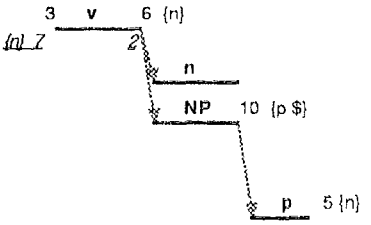

Figure 3-8

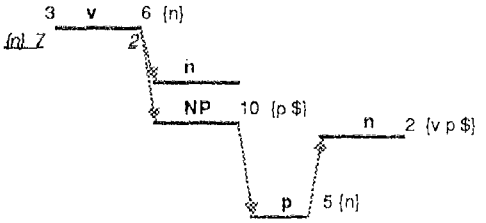

Figure 3-9

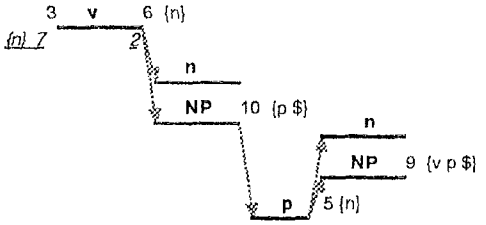

Figure 3-10

The parse continues in this way (Figure 3-9 ~ Figure 3-12).

In Figure 3-12 the new state 10 is ready for the reduce action by $ according to the left-to-right action table. Thus

[1]In practice the juncture verifier should return the probability of juncture instead of just TRUE / FALSE.

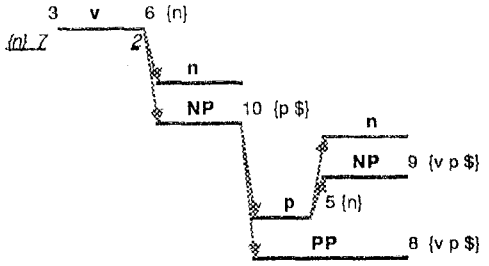

Figure 3-11

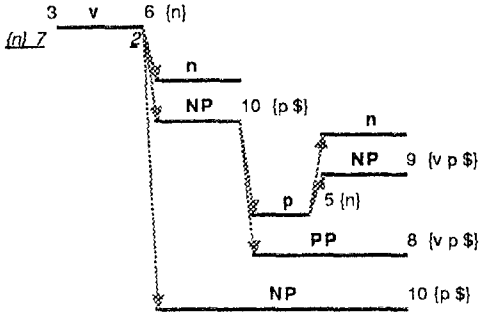

Figure 3-12

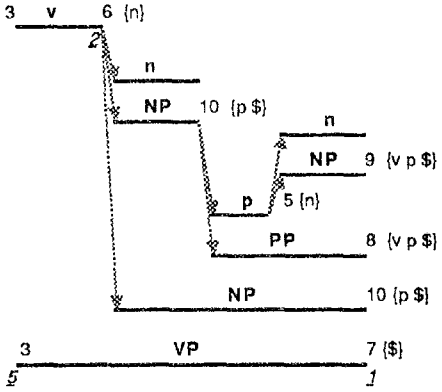

Figure 3-13

the action "reduce 4" is performed. The next state 7 is also ready for the reduce action by $. But this reduce action (S --> NP VP) is interrupted because the parsing stack does not have enough constituents. At this point the suspended right-to-left parse can be resumed because the suspended action "reduce 4" is done. The new state number 5 is determined from the right-to-left goto table. (Figure 3-13)

The first word W-1 is expected by state 5. On the assumption that JUNCT(W-1, W-2) returns TRUE, W-1 is shifted and the new state number 3 is determined from the reverse LR table. (Figure 3-14)

The new state 3 is ready for the reduce action by v, p and $. Since W-1 is the first word in the lattice, the action "reduce 2 (NP --> n)" is performed. (Figure 3-15)

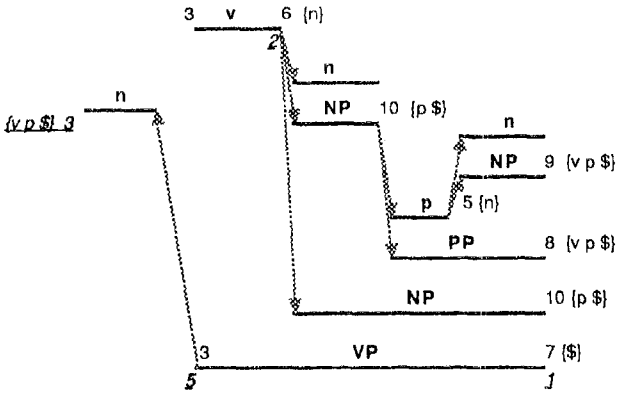

Figure 3-14

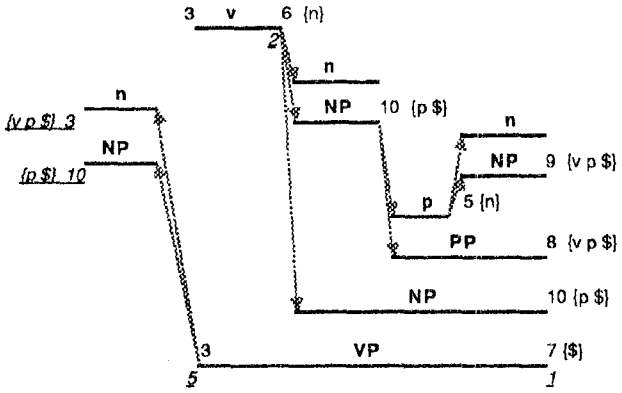

Figure 3-15

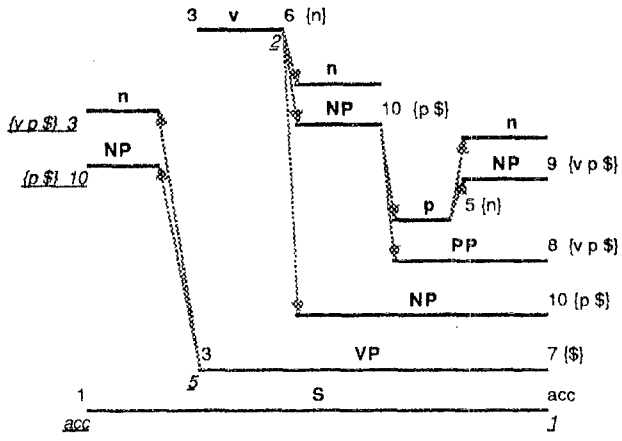

Figure 3-16

State 10 is ready for the reduce action by $. Thus the action "reduce 1 (S --> VP NP)" is performed, which indicates that the suspended left-to-right action "reduce 1" is also done. (Figure 3-16 shows the end of parsing.)

3.3. Bi-directional GLR from Multiple Anchors

We have considered the parse from one anchor word in the previous example. The bi-directional GLR can be started from more than one word in the following way.

[1] Provide each word with its starting states for both right-bound and left-bound parsing from the action tables.

[2] Start bi-directional GLR parsing from each word in parallel.

[3] At the reached state s_i, check if there any nonterminals already exist which s_i is expecting according to the goto table [along the row of state s_i under the column labeled with the nonterminal symbol]. (Since parsing proceeds in parallel, the nonterminal may have been created already.) If JUNCT(current-word,previously-created-nonterminal) returns TRUE, shift this nonterminal onto the current word just the same way as the standard "shift action" for terminals. Note that this "nonterminal shift action" does not prevent the regular shift/reduce/accept actions at state s_i.[2]

3.4. Parsing Words in Order of Probability

In the previous section we showed that the parsing can start from multiple anchors. This assures that the parse can start from any word in any order. This parsing method is very suitable for speech recognition, because the parsing can proceed in the order of probability of each word in the lattice.

3.5. Parsing Incomplete Lattice

In the previous example the lattice contained every necessary word. If the lattice is complete, the generalized LR parsing method suffices [9]. It is often the case, however, that some words are missing in the output from the speech recognizer. In an attempt to use the generalized LR parsing technique for parsing an incomplete lattice [6] or for parsing a noisy input sequence [5], all possibly viable symbols are checked. Especially, handling missing symbols in the early stage of parsing requires a lot of search. The bi-directional GLR parsing can handle missing words more elegantly in that only highly plausible missing candidates are explored as follows.

Suppose W-4("p") is missing from the lattice in Figure 3-3[3]. In parsing the lattice in the order of probability, the parse is suspended after W-3 is shifted. At this moment the left-to-right parsing is expecting "p" as the following word of W-3 and the right-to-left parsing is expecting "p" as the previous word of W-5. Therefore we can assuredly predict "p" is missing between W-3 and W-5.

In case more than one word is missed in the gap, creating expected dummy words tentatively from one side or both from left side and from right side can solve the problem. A top-down speech input verifier which checks the likelihood of dummy words should be incorporated, because search may grow significantly by indiscreet creation of dummy words.

4. Parsing Noisy Speech Input

Saito et al. implemented the system which parses the noisy speech input [5]. In that system the parser analyzes the phoneme sequence from left to right as exploring the possibilities of substituted, inserted, and missing phonemes. Consequently a much bigger search was required than conventional text parsing. Thus the efficient GLR parsing technique was adopted. Since the parse proceeds strictly from left to right pruning the low-scored partial parses, it is sometimes hard to parse the speech input whose beginning part is very noisy. For example, the speech input "ROEAIBIGAZUZIQKISURU" (the correct phoneme sequence is "OYAYUBIGAZUKIZUKISURU" which means "I have a burning pain in the thumb.") can not be parsed correctly by the GLR parser, because of the noisy initial part. To apply the bi-directional parsing technique to this problem, we need to make a word lattice from the phoneme sequence, because

- The current speech recognition device [3] does not give us the probability of each phoneme in the sequence.
- A single phoneme is too primitive to be an anchor symbol.

The word lattice built from the phoneme sequence "ROEAIBIGAZUZIQKISURU" is shown in Figure 4-1. This lattice clearly shows that the correct parse "OYAYUBI GA ZUKIZUKI SURU" can be obtained.

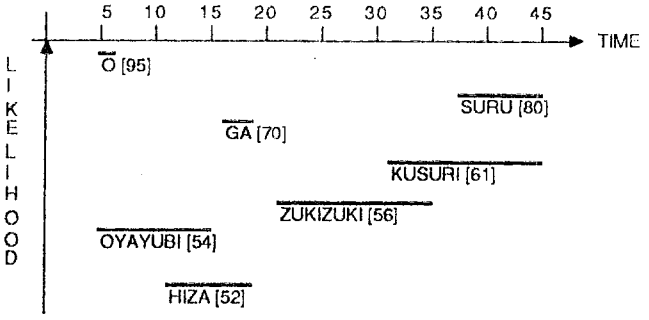

Figure 4-1: Word Lattice from the Phoneme Sequence

[2] In practice, however, regular shift actions do not have to be performed in many cases, because the nonterminals previously created are likely to have a high score due to the fact that the parse starts with anchor symbols. This heuristic method can reduce search.

[3] Such function words as prepositions and articles are likely to be missing in speech recognition.

We tested 125 sentences (5 speakers spoke 25 sentences.) in the domain of doctor-patient conversation. 111 sentences were parsed correctly by the regular GLR method (recognition rate: 89.6 %). 6 more sentences were parsed correctly by the bi-directional parsing of the word lattice (recognition rate: 93.6 %). The remaining 8 sentences were very badly pronounced, in which content words are missing. It is necessary to ask the speaker to say the sentence again or to only speak the unclear portion.

5. Concluding Remarks

We have introduced the bi-directional GLR parsing as a robust parsing technique and how the method is applied, especially for parsing the lattice of words hypothesized by the speech recognizer using the strong power of handling missing words.

The prototype parser has been implemented. Preliminary results show that the robustness power is very effective especially for the lattice where missing words exist in the beginning part.

Acknowledgements

The author is grateful to Dr. Masaru Tomita for the useful comments on this work. The author also thanks the members of the Center for Machine Translation for comments and advice.

References

1. Giachin, E. and Rullent, C. Robust parsing of severely corrupted spoken utterances. 12th International Conference on Computational Linguistics (COLING88), Budapest, Hungary, August, 1988.

2. Hayes, P. J., Hauptmann, A. G., Carbonell, J. G., and Tomita, M. Parsing Spoken Language: a Semantic Caseframe Approach. COLING86, Bonn, August, 1986.

3. Hiraoka, S., Morii, S., Hoshimi, M. and Niyada, K. Compact Isolated Word Recognition System for Large Vocabulary. IEEE-IECEJ-ASJ International Conference on Acoustics, Speech, and Signal Processing (ICASSP86), Tokyo, April, 1986.

4. Massimo Poesio and Claudio Rullent. Modified Caseframe Parsing for Speech Understanding Systems. Proceedings of the Tenth International Joint Conference on Artificial Intelligence, Milan, August, 1987.

5. Saito, H. and Tomita, M. Parsing Noisy Sentences. 12th International Conference on Computational Linguistics (COLING88), Budapest, Hungary, August, 1988.

6. Saito, H. A Phoneme Lattice Parsing for Continuous Speech Recognition. Tech. Rept. TR-I-0033, ATR Interpreting Telephony Research Laboratories, July, 1988.

7. Satta, G. and Stock, O. Head-Driven Bidirectional Parsing: A Tabular Method. 1st International Workshop on Parsing Technologies, Pittsburgh, USA, August, 1989.

8. Tomita, M.. *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems.* Kluwer Academic Publishers, Boston, MA, 1985.

9. Tomita, M. An Efficient Word Lattice Parsing Algorithm for Continuous Speech Recognition. IEEE-IECEJ-ASJ International Conference on Acoustics, Speech, and Signal Processing (ICASSP86), Tokyo, April, 1986.

10. Woods, W. A., Bates, M., Brown, G., Bruce, B., Cook, C., Klovstad, J., Makhoul, J., Nash-Webber, B., Schwartz, R., Wolf, J., and Zue, V. Speech Understanding Systems - Final Technical Report. Tech. Rept. 3438, Bolt, Beranek, and Newman, Inc., Cambridge, Mass., 1976.