Parsing a Lattice with Multiple Grammars

Fuliang Weng[1] Helen Meng[2]
Po Chui Luk[2]
[1]Speech Technology and Research Laboratory
SRI international
333 Ravenswood Ave, Menlo Park, CA 94025
fuliang@speech.sri.com
[2]Human-Computer Communication Laboratory
Department of System Engineering and Engineering Management
the Chinese University of Hong Kong
Shatin, NT, Hong Kong, China
{hmmeng,pcluk}@se.cuhk.edu.hk

Abstract

Efficiency, memory, ambiguity, and robustness are the central issues in natural language parsing. Because of the complexity of natural language, different parsers may be suited only to certain subgrammars. In addition, grammar maintenance and updating may have adverse effects on tuned parsers. Motivated by these concerns, [25] proposed a grammar partitioning and top-down parser composition mechanism for loosely restricted Context-Free Grammar (CFG) grammars [1]. In this paper, we report on significant progress, i.e., (1) developing guidelines for the grammar partition through a heuristic clustering algorithm, (2) devising a new mix-strategy composition algorithms for any rule-based grammar partition in a lattice framework, and 3) initial but encouraging parsing results for Chinese queries from a translated Air Travel Information System (ATIS) corpus.

1 Introduction

Efficiency, memory, ambiguity, and robustness are the central issues in natural language parsing [5, 21, 7, 19, 12, 24, 14]. Among others, Earley's algorithm is often used for its superior performance due to a good combination of top-down prediction and bottom-up bookkeeping. Schabes' algorithm precompiles all the prediction steps and some completion steps in Earley's algorithm, and has resulted in improved average performance [19] while maintaining the cubic time complexity and square space complexity of Earley's algorithm. Tomita [21] generalized Knuth's LR(k) parsers to allow nondeterminism, and reported faster performance than Earley's parser. However, in the worst case, as different researchers [23, 7] have pointed out, (G)LR parsers have exponential time and space complexity. By utilizing a lookup table for *reduce* steps in a GLR parser, Kipps [9] was able to limit the time complexity to $O(n^3)$, but this approach led to a less practical system. Nederhof and Satta [14] further improved the GLR parser through splitting a *reduce* step into more primitive operations that could remove redundancy in parsing. To deal with real-world natural language parsing, others [18, 12, 24] proposed ways to handle extra-grammatical cases for GLR parsers.

[1]The same restriction as Tomita's original algorithm.

As reported in this paper, we have continued our work in grammar partition and parser composition [25] that explores advantages of different parsing algorithms. We have also investigated guidelines for grammar partitioning, the relationship between partitioning and state space selection, and practical ways for composing different parsing algorithms, in the context of robust parsing of Chinese queries from a translated ATIS corpus.

In Section 2, we briefly review some concepts of grammar partitioning, and discuss a few principles for the task and their relationship with state space selection for the parsers mentioned above. Section 3 presents an improved parser composition algorithm. Section 4 describes some initial but encouraging experiments on parsing Chinese queries. The final section presents our conclusion and describes our future research directions.

2 Grammar Partitioning

To facilitate our discussion, we first briefly review the motivation and concepts for grammar partition [25]. Then, we give some intuitive guidelines for the partitioning and their relationship with state space selection for the parsers analyzed by [13].

2.1 Motivation and Concept

Earley [4] and Ukkonen [23] proved that the following artificial grammar G is exponential for LR(k) parsers, and even for any right parsers, respectively:

$S \rightarrow A_i \quad (1 \leq i \leq n)$

$A_i \rightarrow a_j A_i \quad (1 \leq i \neq j \leq n)$

$A_i \rightarrow a_i B_i \mid b_i \quad (1 \leq i \leq n)$

$B_i \rightarrow a_j B_i \mid b_i \quad (1 \leq i \leq n)$

These results present a potentially severe problem for LR(k) parsers and their applications to natural language processing. However, as illustrated in [25], if we partition the grammar into n subgrammars, parse each subgrammar separately, and combine subparsers for the subgrammars, we can obtain a satisfactory compromise in terms of speed and memory.

From a practical point of view, in addition to the possibility of causing an exponential number of compiled states, sublanguages in a large parser may be described by pre-existing subgrammars using specialized parsing algorithms. Implementation and maintenance constraints may prevent integration of the sublanguage modules into a monolithic system. A framework for partitioning grammars and composing parsers would be useful.

In the rest of this paper, we use the definitions in [25] for grammar partition. Intuitively, we partition the production set into subsets, and create subgrammars for the subsets, accordingly. The interaction among different subgrammars is through nonterminal sets, i.e., INPUT and OUTPUT, and a virtual terminal technique. The INPUT to a subgrammar is a set of nonterminals that were previously parsed by other subgrammars. The OUTPUT of a subgrammar is those nonterminals that were parsed based on this subgrammar and used by other subgrammars as their INPUT symbols. For every nonterminal A in the INPUT set of a subgrammar, there is an augmented rule $A \rightarrow vt_A$ for that grammar, where vt_A is called a *virtual terminal* acting as if it were a terminal [10, 24, 25]. A directed calling graph for the subgrammar set of G is defined as (V, E), where V is the subgrammar set and $E = (A, B)$, with the overlap of the OUTPUT symbols of B and the INPUT symbols of A being nonempty. It can be proved that the partitioned grammar will lead to the same recognizable language. It is straightforward to see that this grammar partition definition is more general than the one defined with respect to nonterminal symbol set [10].

2.2 Guidelines for Grammar Partitioning

Despite the fact that grammar partitioning is desirable, it is not clear how it should be done in any general case. We describe properties that may help in understanding this subject.

First, let us look at the two extreme cases: the coarsest and finest partitions. The coarsest partition is the grammar itself, and all the production rules are included in the set. The finest grammar partitioning is the set of all singletons with one exact grammar rule in each set. The INPUT/OUTPUT set members are the nonterminal symbols on the right-hand side/left-hand side (RHS/LHS) of the rules in these singletons. So, an inverse problem of grammar partitioning is the clustering of smaller subgrammars into bigger subgrammars based on calling relations among different subgrammars.

Intuitively, we may want to create clusters that have frequent interaction within cluster members, but fewer interactions between members of different clusters. This will reduce the possibility of combining common prefix computations for highly ambiguous grammars. However, it may increase the chance of computing a common prefix repetitively. Our solution is to form as big a cluster as possible within the affordable size limit.

A simple heuristic algorithm for building subgrammars based on their sizes is as follows.

Create a weighted graph. Its node set consists of all the clusters in the closure. Its edge set, together with its weights, is defined through the following two steps.

For each pair of clusters:

1. If the INPUT of one cluster overlaps with the OUTPUT of the other cluster, put the total size of the two symmetric overlapping sets on the edge connecting the two clusters.
2. If the total is zero, no edge is created between the two clusters.

Perform the following merge-split steps iteratively for every cluster, until all the clusters reach their size limits.

Merge:

1. Compute the transitive closure of the calling relations for the cluster.
2. If the size of the closure (or the size of its compilation) is smaller than a preset size limit, make this closure as the new cluster, and at the same time remove the old clusters that form this new cluster.

3. If the size of the closure (or its compilation) is larger than the preset size limit, perform a split operation on this closure.

4. If a cluster has an empty INPUT set and merging with its parent clusters does not result in larger clusters than the preset size, duplicate the cluster and merge it with its parents.

Split:

Loop until the input cluster is disconnected and its sub-clusters reach desirable sizes.

1. Reduce the weights of all the edges by 1.

2. Remove edges that have weight 0.

When training data is available, the weights on the edges between different clusters can be defined on the actual calls between the corresponding clusters. The merge and split steps can be performed using the empirical data, which, in a way, is parallel to explanation-based grammar chunking [16].

If dotted rules are considered, the state space of the viable prefix recognition NFA [1, 3] (VP-NFA) is one extreme partition with each dotted rule as a state (or, a partitioned subgrammar). In the VP-NFA, for each pair of states $A \rightarrow \alpha.X\beta$ and $A \rightarrow \alpha X.\beta$, there is a transition with X as its label, and for each pair of states $A \rightarrow \alpha.B\beta$ and $B \rightarrow .\gamma$, there is a ϵ-transition. If we collapse the end states of ϵ transitions in the VP-NFA with their corresponding start states (if multiple ϵ transitions go to a state node, we duplicate it and its out-going transitions for each state to be merged), we obtain the state space of Schabes' algorithm. The resulting state transition automaton is still nondeterministic with each state containing only one kernel dotted rule. On the other hand, if we determinize the VP-NFA, we obtain another extreme partition, i.e., the state transition automaton for GLR parsers. Two measurements can be used for the state merge decisions. One is the maximum degree of nondeterminism for each state. In other words, one can merge two out-going transitions with the same label and from the same state to reduce the nondeterminism until the nondeterministic degree reaches a preset limit. The other measure is the maximum length of common prefix paths. Notice that the common prefix may contain nonterminals, therefore it is not simply the lookahead. If training data is available for the state space, similarly, we can search for the two parameters empirically, and decide the clusters. Our two measurements provide some details to the observation in [13] that allowing different degrees of nondeterminism in the state transition graphs would lead to different time and space complexity: Schabes' algorithm allows a maximum degree of nondeterminism in the state transition, while Tomita's algorithm allows a minimum degree of nondeterminism.

3 Parsing with Multiple Grammars

Weng and Stolcke [25] presented a top-down style algorithm that combines several parsers with partitioned subgrammars, with a restriction that the grammar must be noncyclic and does not have certain ϵ-productions, the same restriction as Tomita's original parser [15]. Here, we will present a flexible parsing composition algorithm that does not have such restriction on the grammar.

The intuition behind this algorithm is that each parser with a subgrammar will try to parse from the current input position and place its output nonterminal (virtual terminal) onto the same lattice, if successful. The representation used for interfacing different parsers is a chart or a lattice. The nodes of the lattice are spatial positions, and its transitions represent terminals/virtual terminals that cover positions indicated by their corresponding start and end nodes. Usually, a terminal covers one position, while a virtual terminal can cover multiple positions. During parsing, both terminals and virtual terminals are represented in the same lattice, but, partial constituents and nonterminals that are not the output of any subgrammar cannot be placed there. This implies that only the granularity represented by subgrammars is present in the lattice, and therefore, is termed a lattice with multiple granularity (LMG). In addition, this representation gives us a good control over the lattice density for parsing. These features differentiate LMG from the traditional chart used in other parsing algorithms [8, 2, 17, 27].

Dealing with word lattice input for a single GLR parser was solved by Tomita [22]. Two steps in Tomita's algorithm are different from the standard GLR parsing algorithm: the lattice is topologically sorted for synchronization; for *shift* action, an additional procedure tests whether the current word $word_i$ is adjacent to any word on the top of the graph-structured stack. This same idea can be used for the Earley parser, left-corner parser, and shift-reduce parser. That is, instead of moving from i to $i+1$, the new algorithm will look for the *next* symbol and for its left adjacent symbol in the lattice. However, direct application of this idea would force topological sorting to be performed each time a virtual terminal is added to the lattice. In the new composition algorithm, we use a stack to store the initially sorted lattice input, and dynamically add newly found virtual terminals to the stack. In this way, we can avoid performing the constant topological sort required by the simplistic approach.

In the original definition [25], we duplicated multiple subgrammars for a partition element with multiple output nonterminals. However, it is not necessary to make such duplication for the purpose of parsing because we can easily overcome the difficulty in handling multiple output symbols. To do so, we let $\{O_j^i\}_{j=1}^k$ be the output of a partition element G_i. If we add rule set $\{S_i \rightarrow O_j^i\}_{j=1}^k$ to G_i, use S_i as its start symbol for partition element G_i, and label the output node in the lattice with corresponding O_j^i during parsing, the correctness of derivation is still guaranteed. In the rest of this paper, without loss of generality, we can safely adopt the more general definition.

We have developed a new composition algorithm for parsers with different subgrammars, given a word lattice as its input. Without loss of generality, we use GLR parsers. Other parsers, such as Earley's parser or the shift-reduce parser, can be used in combination.

Let the word lattice be $L = (N, E)$, where N is a node set, indicating spatial positions, and E is a set of directed transitions, indicating the ranges the (virtual) terminals cover. Functions $start(dt)$, $end(dt)$, and $word(dt)$ return the start, the end, and the word label of directed transition dt, respectively. Let σ be the stack that stores the topologically sorted transition sequence from L.

Suppose that psr_i is the parser for subgrammar G_i, psr_0 is the parser for the top grammar G_0, and the largest subgrammar index is K. Each psr_i takes a (virtual) terminal stack σ as input (call by value), and returns a list of pairs with form (v, d), where d is the end node for virtual terminal v of subgrammar G_i if parsed successfully. If no parse is found, it returns ϕ.

Parser Composition Algorithm: This algorithm replaces PARSE() and PARSEWORD() in Farshi's version of Tomita's algorithm [15]. In this algorithm, ACTOR(A, Q, w), COMPLETER(R, Γ, T_f), and SHIFTER(Q, Γ, T_f) are similar to that version, with a few minor changes: w, the to-be-processed word, replaces a_{i+1} as the next word for processing in ACTOR; U_j, the state set at node j in L, replaces U_{i+1} as a state set next to U_i when w corresponds to the transition from U_i to U_j. Both U_i and L are shared by all the parsers. However, graph-structured stack Γ and shared-packed forest T_f are private to each parser. Started parser list SP records all the parsers started at a particular location in L to avoid infinite recursion.

1. Topologically sort L into $dt_0, ..., dt_l$, and push all the transitions onto stack σ in the reverse order[2].
2. $\Gamma = \phi$, $T_f = \phi$, $r = nil$ (initializing graph-structured stack, forest, and root).
3. Create in Γ a vertex v_0 labeled with state id s_0, $U_0 = \{v_0\}$.
4. $SP = \phi$ (initializing the started parser list).
5. $dt = pop(\sigma)$, $SP = \{(0, dt, L)\}$, $psr_0(dt, \sigma)$.

PARSER $psr_m(dt, \sigma)$**,** $m = 0, ..., K$**:**

1. Loop until dt is null do
   (a) $i = start(dt)$, $j = end(dt)$, $w = word(dt)$.
   (b) $A = U_i$; $R, Q = \phi$.
   (c) repeat
      i. if $A \neq \phi$ then do ACTOR(A, Q, w).
      ii. else if $R \neq \phi$ then do COMPLETER(R, Γ, T_f).

      until $A == \phi \cap R == \phi$ (done with reductions).
   (d) PREDICTIVE-SUB-PARSING(dt, σ).
   (e) SHIFTER(Q, Γ, T_f).
   (f) $dt = pop(\sigma)$.

[2]Notice that the resulting stack σ has $l + 1$ element with w_0 on the top of σ.

Predictive Sub-parser: The parameters for procedure PREDICTIVE-SUB-PARSING are call-by-name. In the procedure, $OUTPUT_k$ is the OUTPUT set of subgrammar G_k, and $PVT(U_i, w)$ is a function that takes a set of states and a (virtual) terminal, and returns a set of legitimate virtual terminals to be parsed at that point.

PREDICTIVE-SUB-PARSING(dt, σ):

1. $flag$ = true.
2. while $flag$ do
   (a) $flag$ = false.
   (b) $i = start(dt)$, $j = end(dt)$, $w = word(dt)$.
   (c) For $k = 1$ to K do
   if $OUTPUT_k \cap PVT(U_i, w) \neq \phi$ and $(k, dt, L) \notin SP$ then do
      i. $SP = (k, dt, L) \cup SP$.
      ii. $lvt = psr_k(dt, \sigma)$.
      iii. For $(m, d) \in lvt$, $m \in PVT(U_i, w)$ and m is not in L to cover from i to d then[3]
         A. add m to L: create a transition mdt with label m that connects from node i to node d.
         B. push mdt onto stack σ.
         C. $flag$ = true.

Notice that function PREDICTIVE-SUB-PARSING only adds virtual terminals to the stack and lattice, and these virtual terminals have the same start node as the transition being processed. Therefore, the updated stack still maintains the topological order for the updated lattice. This leads to a straightforward correctness proof of the algorithm.

In PREDICTIVE-SUB-PARSING, function $PVT(U_i, w)$, serving as a subparser filter, has not been defined. The occurrence of $PVT(U_i, w)$ in the algorithm is not essential for its correctness. However, without PVT it would start multiple parsers at every node in the lattice, that is, a big efficiency concern. Our technique for addressing this problem resembles predictive calling of subparsers in the strict top-down composition algorithm [25], or $FIRST$ in the conventional parsing table generator.

Let $VT(i) = \{vt | i \text{ is a state}, vt \text{ is a virtual terminal}, \exists s, shift\ s \in \text{ACTION}(i, vt)\}$. The predictive set of virtual terminals for a given state i and a (virtual) terminal w is defined as $PVT(i, w) = \{vt | vt \in VT(i), w \text{ is a (virtual) terminal}, VT(i), vt \stackrel{*}{\Rightarrow} w ...\}$. To overload the notation a little bit, we define the predictive set of virtual terminals for a set of given states Q and a (virtual) terminal w as: $PVT(Q, w) = \bigcup_{i \in Q} PVT(i, w)$. The algorithm to realize the definition is straightforward, and therefore omitted.

[3] For getting all the possible parse trees, somewhat more complicated test needs to be made, that is, to see whether the top level structure (rule) has already been built.

Intuitively, for any state in U_i, if there is a *shift s* step under a virtual terminal for that state in the ACTION table of the caller parser, we add the virtual terminal to $PVT(U_i, w)$. Those virtual terminals that do not have w as their left corner can be pruned.

For other types of parsers, we can also compute a set of virtual terminals, given a nonterminal (or a set of nonterminals) and a left corner, similar to the way PVT or $FIRST$ is computed. Thus, we can avoid invoking subparsers excessively.

In addition to predictive pruning, parallelization is an approach to the efficiency problem. Notice that all legitimate subparsers in PVT for a particular transition in the lattice can be parallelized without affecting other parts of the algorithm.

To address both parsing accuracy and efficiency simultaneously, we can incorporate probabilistic models into lattice. Estimation of ngram word transition models is standard. In our lattice framework, head words can be passed to virtual terminals for more detailed probabilistic estimation to better capture certain long distance dependency without lexicalizing the grammar. The latter approach leads to a huge grammar in any practical system [6]. However, a potential sparse data problem for the introduction of lexicalized virtual terminals can be partially addressed by the relatively easy-to-control granularity of grammar partitioning.

To address the robustness for real-world applications, the techniques used in the EGLR parser [24] can be adapted in the lattice framework, because, in the EGLR parser, the three editing operations (insertion, deletion, and substitution) can be applied to both terminals and virtual terminals. Modifying input is equivalent to adding transitions in the lattice [11, 26]. The only additional issue is that the cost function for the optimal path selection may not be a uniform unit cost, but should be associated with the number of words changed when virtual terminals are in the editing operations.

4 Experiments

Ideally, we want to test grammar partitioning and parser composition in its full scale. However, before the full implementation of the algorithms described above, we took a shortcut to evaluate the feasibility in a real-world natural language application. We describe a concrete partition of a Chinese grammar in a translated ATIS domain, and a robust cascading parsing and composition. We then report the parsing and understanding results for the Chinese ATIS domain.

4.1 Cascading Parsing for Chinese ATIS Grammars

English ATIS tasks contain queries regarding air travel information, including flights leaving from or arriving at locations at various dates and times, services of certain flights, and airline routes. The Chinese queries used in the experiments are translated from an English ATIS corpus.

Based on the guidelines in the split-merge algorithm for grammar partitioning, we manually partitioned the rule set into subgrammars, such as STATE-NAMES, CITY-NAMES, AIRPORT-CODE, with corresponding virtual terminals prefixed with *vt*. In this particular case, the calling graphs of the subgrammars are acyclic and, therefore, they can be assigned with level ids. Their corresponding subparsers are GLR parsers, compiled separately, and assigned with the same ids. The top-level sentence subgrammar is not yet completed and deployed, because of the problem of flexible word order in Chinese. However, the LMG representation allows a robust parser to compose all the legal sequences of virtual terminals and choose the best path that covers the most words in the input string.

The whole parsing process proceeds as follows. The control mechanism takes a Chinese query sentence and converts it into an LMG. Then, the subparsers at the lowest level are activated to parse the LMG, and leave their corresponding virtual terminals on the LMG when parsing succeeds. If no more virtual terminals can be added to the LMG, the subparsers at one level higher are activated and start to parse the same LMG. This process continues until the highest level is reached, when the robust sentence-level parser finishes its job of determining the best virtual terminal sequence. For the semantic evaluation described in subsection 4.2, a semantic interpreter converts syntactic virtual terminals into semantic frames of key-value pairs.

The main difference between the parsing composition algorithm described in Section 3 and the one used here is the adoption of a robust master parser instead of a GLR parser. This gives us a good way to handle a real-world application, but does not offer any predictive pruning at the sentence level. However, when a robust predictive parser is tuned for an application, it can replace the current one without affecting the other components, which is the nice feature of the approach described in this paper.

4.2 Experimental Results

Our experiments use a translated Chinese ATIS-3 corpus, specifically the Class A queries. It contains 1564 queries from the training set, 448 queries from the 1993 test set and 444 queries from the 1994 test set. Each utterance is labeled with a corresponding SQL query for information retrieval.

The grammar rules used for parsing were developed based on 375 queries from the MIT subset of the training set, and the remaining 1189 queries of the training set were held for tuning. When parse errors occur in any of the held-out sentences, we modify the grammar rules to incorporate the changes. Evaluation experiments on 1993 and 1994 test sets were conducted. Table 1 shows the general statistics of the hand-crafted grammar. As we can see from the table, the majority of the rules (807 out of 962) are directly related to the lexicon, and only about 150 rules are related to phrase structures.

Applying the parsing algorithm described in Section 4.1 and the above grammar on the training set and the two test sets, we obtained the results shown in Table 2, where the error rates are measured against the translated standard semantic key-value pairs supplied in the ATIS-3 corpus by Linguistic Data Consortium. *Full understanding* refers to utterances with full matches between the semantic categories in our frame and those in the standard answer with no error. *Partial understanding* refers to partial matches with an error rate between zero and one. *No understanding* refers to no matches.

In addition to the incomplete grammar coverage, parsing errors can have sources in translation errors or implicit information. The former are mainly caused by wrong or imprecise translation from English to Chinese. The latter are due to a discrepancy in the semantic meaning translation between the standard answers and our semantic interpreter. For example, *tonight* has semantic meaning of $time \geq 1800 \& time \leq 2359$, while our interpreter has only a symbolic value. We are developing a correction for this discrepancy.

We also compiled Table 3 summarizing the parsing statistics. From the table, we observe that the speed of our parser is very fast for this application. In addition, the number of rules reduced and states visited in the best paths and in the successful sub-parses are less than those in the overall parsing process, which means that further pruning is possible.

Because the current master parser does not use explicit sentence-level grammar rules, we do not have direct parsing table size comparison between partitioned and unpartitioned grammars. However, when we take those best paths from all the training and testing data that have more than one occurrence, and use them as the sentence rules (totally 203 rules), we obtained following parser size numbers. For the unpartitioned grammar, the total number of GLR states is 9521, while for the partitioned grammars, the total number of GLR states is only 2804, out of which 928 states are used by the sentence-level master grammar. This shows that, at least in the ATIS domain, grammar partitioning does lead to the desirable performance.

5 Conclusions and Future Work

We have presented guidelines for grammar partitioning through a split-merge algorithm, and proposed a new mix strategy parser composition algorithm based on a flexible LMG representation, so that we can alleviate size, speed, and robustness problems in real-world natural language applications. We also reported our initial but encouraging parsing results in the Chinese ATIS domain.

We plan to further integrate the predictive robust parsing (PRP) algorithm [24] into the current system, and to use probabilistic modeling for better pruning and selecting of correct parses. To alleviate the current grammar writing bottleneck in our parsing experiments, we will also work with semi-automatically induced grammars [20].

Acknowledgments

This work is mostly supported by the Direct Grant from the Chinese University of Hong Kong, and, in part, supported by SRI International, under Mediated Space project. Part of this work was done while the first author was visiting the Chinese University of Hong Kong.

References

[1] A. Aho and J. Ullman. *Principles of Compiler Design.* Addison-Wesley, Reading, MA, 1977.

[2] J. Amtrup. Parallel Parsing: Different Distribution Schemata for Charts. In *IWPT1995*, 1995.

[3] H. Chen, J. Qian, and Y. Sun. *Principles of Compilation for Programming Languages (in Mandarin).* XinHua Publisher, Beijing, 1984.

Types	size	Remarks
rules	962	all the rules in the grammar
lexical rules	807	rules deriving lexical entries
nonterminals	75	all the nonterminals
virtual terminals	49	all the virtual terminals
terminals	415	all the words

Table 1: Grammar rule statistics.

[4] J. Earley. *An Efficient Context-free Parsing Algorithm.* Ph.D. thesis, Carnegie Mellon University, 1968.

[5] J. Earley. An Efficient Context-free Parsing Algorithm. *Communications of ACM*, 13(2), 1970.

[6] J. Eisner and G. Satta. Efficient parsing for bilexical context-free grammar and head automaton grammars. In *Proceedings ACL 1999*, 1999.

[7] M. Johnson. The Computational Complexity of Tomita's Algorithm. In *Proceedings of the 1st International Workshop on Parsing Technologies*, 1989.

[8] M. Kay. Algorithm Schemata and Data Structures in Syntactic Processing. In B. Grosz, K. Jones, and B. Webber, editors, *Readings in Natural Language Processing.* Morgan Kaufmann Publishers, Inc., 1986.

[9] J. Kipps. Analysis of Tomita's Algorithm for General Context-Free Parsing. In *Proceedings of the 1st International Workshop on Parsing Technologies*, 1989.

[10] A. Korenjak. A Practical Method for Constructing LR(k) Processors. *Communications of ACM*, 12(11), 1969.

[11] B. Lang. A Generative View of Ill-Formed Input Processing. In *ATR Symposium on Basic Research for Telephone Interpretation (ASTI)*, 1989.

[12] A. Lavie and M. Tomita. An Efficient Noise-Skipping Parsing Algorithm for Context-Free Grammars. In *Proceedings of the 3rd International Workshop on Parsing Technologies*, 1993.

[13] M.-J. Nederhof. An Optimal Tabular Parsing Algorithm. In *Proceedings of ACL 1994*, 1994.

[14] M.-J. Nederhof and G. Satta. Efficient Tabular LR Parsing. In *Proceedings of ACL 1996*, 1996.

[15] R. Nozohoor-Farshi. GLR Parsing for e-Grammars. In Masaru Tomita, editor, *Current Issues in Parsing Technology.* Kluwer Academic Publishers, 1991.

[16] M. Rayner and D. Carter. Fast Parsing Using Pruning and a Grammar Specialization. In *Proceedings of 1996 ACL*, 1996.

[17] T. Ruland, C. Rupp, J. Spilker, H. Weber, and K. Worm. Making the Most of Multiplicity: A Multi-Parser Multi-Strategy Architecture for the Robust Processing of Spoken Language. In *Proceedings of ICSLP 1998*, 1998.

[18] H. Saito and M. Tomita. GLR Parsing for Noisy Input. In *Proceedings of the 2nd International Workshop on Parsing Technologies*, 1991.

[19] Y. Schabes. Polynomial Time and Space Shift-Reduce Parsing of Arbitrary Context-Free Grammars. In *Proceedings of 1991 ACL*, 1991.

[20] K. Siu and H. Meng. Semi-automatic Acquisition of Domain-specific Semantic Structures. In *Proceedings of the 6th European Conference on Speech Communication and Technology*, 1999.

[21] M. Tomita. *Efficient Parsing for Natural Language.* Kluwer Academic Publishers, 1985.

[22] M. Tomita. An Efficient Word Lattice Parsing Algorithm for Continuous Speech Recognition. In *Proceedings of ICASSP-86*, 1986.

[23] E. Ukkonen. Lower Bounds on the Size of Deterministic Parsers. *Journal of Computer and System Sciences*, 26:153–170, 1983.

[24] F. Weng. Handling Syntactic Extra-grammaticality. In *Proceedings of the 3rd International Workshop on Parsing Technologies*, 1993.

[25] F. Weng and A. Stolcke. Partitioning Grammars and Composing Parsers. In *Proceedings of the 4th International Workshop on Parsing Technologies*, 1995.

[26] F. Weng and Y. Wang. *Introduction to Computational Linguistics (in Mandarin).* Sino Social Sciences Publisher, Beijing, 1998.

[27] K. Worm. A Model for Robust Processing of Spontaneous Speech by Integrating Viable Fragments. In *Proceedings of COLING/ACL 1998*, 1998.

	Training set	1993 test set	1994 test set
No. of query	1564	448	444
Ave. sent. length (# of chars)	19.33	16.63	20.17
Ave. error rate	7.96%	11.35%	13.61%
Full understanding	80.50%	77.46%	70.94%
Partial understanding	17.33%	17.86%	23.87%
No understanding	2.17%	4.69%	5.18%

Table 2: Error rate and accuracy statistics.

	Training set	1993 test set	1994 test set
Total time for entire set (Sun Ultra 5 under Solaris)	78 seconds	8 seconds	14 seconds
Ave. no. of states visited incl. failed subparsers	57.01	47.31	60.41
Ave. no. of rules reduced incl. failed subparsers	12.61	10.57	12.94
Ave. no. of states visited incl. only successful subparsers	36.72	29.37	38.40
Ave. no. of rules reduced incl. only successful subparsers	12.00	9.74	12.36
Ave. no. of states visited in the best paths	29.08	24.82	30.00
Ave. no. of rule reduced in the best paths	9.04	7.93	9.19

Table 3: Statistics of the grammar used for the experiments.