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 Communications Laboratory
Department of Systems 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, robustness and scalability 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 Grammars (CFGs). In this paper, we report on significant progress, i.e., (1) developing guidelines for the grammar partition through a set of heuristics, (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 and English queries from an Air Travel Information System (ATIS) corpus.

1 Introduction

Efficiency, memory, ambiguity, robustness, and scalability 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

*The author is currently with Intel China Research Center, No.1 GuangHua Road, Beijing 100020, China. His new e-mail address is fuliang.weng@intel.com.

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.

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 and English queries from an ATIS corpus.

In Section 2, we briefly review some concepts of grammar partitioning, and discuss a few guidelines 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 results on parsing Chinese and English 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 concept 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_n 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 of different types would be useful for the scalability of a parsing system.

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 of B and the INPUT 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 criterion is to form as big a cluster as possible within an affordable size limit.

First we create a weighted graph describing the connections among different clusters. Its node set consists of all the clusters in the grammar. When no training data is available, its edge set and weights are obtained through the following procedure. For each pair of clusters, intersect one cluster's INPUT and the other cluster's OUTPUT, take the total size of the two symmetric intersections as the weight on the connecting edge, and create an edge between the two clusters only when the weight is above a certain threshold.

When training data is available, probability weights on the edges can be computed based on the actual calls between the corresponding clusters, in a way, related to the grammar chunking approach [16].

For better cluster quality, we present a set of simple heuristics for the refinement of grammar clusters. There are two types of refinements, merge to form bigger clusters, and split to form smaller clusters. The merge operations are:

1. Compute the transitive closures of the calling relations for clusters, and replace the original clusters with their closures.

2. If a cluster has an empty INPUT set, duplicate the cluster for each of its parent clusters, and merge the copies with its parents.

The split operations can be through removing the edges with low weights or the ones that lead to minimum changes with respect to its original graph using *Kullback-Leibler* distance.

If dotted rules are considered, the state space of the viable prefix recognizer 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 calling graph must not have left recursion, a somewhat stronger constraint than 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 new algorithm is that each subparser 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. A subparser is invoked by a caller subparser only when certain predictive pruning conditions are satisfied to avoid excessive parser invoking. The representation used for interfacing different parsers is a special chart or 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 as 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* adjacent symbol to the current node 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 such a way that the topological order is always preserved for the changing LMG. By doing this, we avoid performing a constant topological sorting 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 $\{O^i_j\}^k_{j=1}$ be the output of a partition element G_i. If we add rule set $\{S_i \rightarrow O^i_j\}^k_{j=1}$ 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^i_j 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 $\emptyset$.

Parser Composition Algorithm: This algorithm replaces PARSE() and PARSEWORD() in Farshi's version of Tomita's recognition algorithm [15]. In this algorithm, ACTOR(A, Q, w), COMPLETER(R, Γ), and SHIFTER(Q, Γ) 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. U_i's and L are shared by all the parsers. However, graph-structured stack Γ and shared-packed forest are private to each parser. SP is a list of triples (pid, dt, σ), where pid is a parser id, dt is the transition being processed, and σ is the sorted transition stack. 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[1].

2. $\Gamma = \emptyset$. (initializing graph-structured stack).

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

```
3. Create in $\Gamma$ a vertex $v_0$ labeled with state id $s_0$, $U_0 = \{v_0\}$.

4. $SP = \emptyset$ (initializing the started parser list).

5. $dt = pop(\sigma)$, $SP = \{(0, dt, \sigma)\}$, $psr_0(dt, \sigma)$.
```

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) $A = U_i$; $R, Q = \emptyset$.
   (c) repeat
         i. if $A \neq \emptyset$ then do ACTOR($A, Q, w$).
         ii. else if $R \neq \emptyset$ then do COMPLETER($R, \Gamma$).
       until $A == \emptyset \cap R == \emptyset$ (done with reductions).
   (d) PREDICTIVE-SUB-PARSING($dt, \sigma$).
   (e) SHIFTER($Q, \Gamma$).
   (f) $dt = pop(\sigma)$.
```

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 $OUTPUT_k \cap PVT(U_i, w) \neq \emptyset$ and $(k, dt, \sigma) \notin SP$ then do
         i. $SP = (k, dt, \sigma) \cup SP$.
         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[2]
              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 $\sigma$.
              C. $flag$ = true.
```

[2]For getting all the possible parse trees, somewhat more complicated test needs to be made, that is, to see whether the top level structures (rules) have already been built.

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 $VT(i) = \{vt | i \text{ is a state}, vt \text{ is a virtual terminal}, \exists s, \textit{shift } s \in \text{ACTION}(i, vt)\}$. The predictive set of virtual terminals for a given state i and a (virtual) terminal w is defined as $PVT(i, w) = \{vt | vt \in VT(i), w \text{ is a (virtual) terminal}, VT(i), vt \overset{*}{\Rightarrow} w...\}$. 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.

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 Chinese and English grammars in an ATIS domain, and a robust cascading parsing and composition. We then report the parsing and understanding results for the ATIS domain.

4.1 Cascading Parsing for the 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 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 indices (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 flexible word order problem in Chinese and many variations of sentence structures in English. 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 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[3]. 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 viterbi-style robust master parser instead of a GLR parser. This gives us a flexible way to deal with extra-grammatical queries even without any sentence-level grammar rules. As a consequence, it also derives a preliminary set of sentence-level grammar rules for future refinements as we will see in Section 4.2. On the other hand, when both predictive pruning and robust handling are integrated and tuned in a GLR parser, it can replace the current one without affecting the other components, which is the desired feature of the approach described in this paper.

4.2 Experimental Results

Our experiments use the English ATIS-3 corpus and its translated Chinese version, 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.

[3]When there is a recursion at the same level of the calling graph, we can repetitively activate the subparsers at that level until no more virtual terminals can be added to the LMG, which corresponds to the 'while *flag* loop' in Predictive sub-parser algorithm.

For the Chinese experiment, the non-sentence level 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. Evaluations were conducted on 1993 and 1994 test sets. Table 1 shows the general statistics of the hand-crafted grammar. As we can see from the table, the majority of the rules (825 out of 1013) are directly related to the lexicon, and only about 178 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 Chinese translation of the 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. *Part. 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 summarized the parsing statistics in Table 3. 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, as an approximation, we can take as the sentence level rules those best paths from all the training data that occur more than once. In the Chinese experiment, we obtained 205 such rules, in addition to the original 1,013 lower level rules. For the unpartitioned grammar, the total number of GLR states is 10,395, while for the partitioned grammars, the total number of GLR states is only 2,825, out of which 852 states are used by the sentence-level subgrammar.

To speed up the English experiment, we constructed English lower level rules based on the Chinese rules and repeated the same processes for parsing and for obtaining English sentence level rules as we did for Chinese. English parsing results are listed next to its Chinese counterpart in Tables 1 to 3. There are additional 198 sentence level English rules. For the unpartitioned grammar, the total number of GLR states is 10,233, while for the partitioned grammars, the total number of GLR states is only 2,128, out of which 671 states are used by the sentence-level subgrammar. Both Chinese and English experiments show that, at least in the ATIS domain, grammar partitioning leads to the desirable effect of reducing parser sizes.

5 Conclusions and Future Work

We have presented guidelines for grammar partitioning through a set of heuristics, 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 ATIS domain.

We plan to further integrate the predictive robust parsing 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

Part of this work was done while the first author was visiting the Chinese University of Hong Kong. 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.

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.

[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.

Types of rules	sizes		Remarks
	Chinese	English	
non-top-level rules	1013	953	all the rules in the grammar
lexical rules	825	773	rules deriving lexical entries
nonterminals	77	81	all the nonterminals
virtual terminals	50	52	all the virtual terminals
terminals	419	473	all the words

Table 1: Grammar rule statistics.

	Training set 1564 queries		1993 test set 448 queries		1994 test set 444 queries	
	Chinese	English	Chinese	English	Chinese	English
Ave. sent. length (# of chars/words)	19.33	11.24	16.63	10.26	20.17	11.46
Ave. error rate (%)	7.94	7.30	11.22	9.69	13.61	12.40
Full understanding (%)	80.56	83.95	77.90	87.05	70.95	76.35
Part. understanding (%)	17.26	14.13	17.41	10.49	23.87	21.40
No understanding (%)	2.17	1.92	4.69	2.46	5.18	2.25

Table 2: Error rate and accuracy statistics.

	Training set 1564 queries		1993 test set 448 queries		1994 test set 444 queries	
	Chinese	English	Chinese	English	Chinese	English
Total time for entire set (in seconds, Ultra sparc 5)	21	14	5	4	7	5
Ave. no. of states visited incl. failed subparsers	64.9	36.6	53.3	33.3	68.5	38.8
Ave. no. of rules reduced incl. failed subparsers	13.4	14.6	11.3	13.2	13.7	15.0
Ave. no. of states visited, only successful subparsers	39.0	30.2	31.3	27.0	40.1	31.1
Ave. no. of rules reduced, only successful subparsers	12.7	13.9	10.4	12.5	12.9	14.3
Ave. no. of states visited in the best paths	30.7	24.1	26.0	21.6	31.1	24.7
Ave. no. of rules reduced in the best paths	9.5	11.0	8.2	9.9	9.5	11.1

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