GLR Parsing with Multiple Grammars for Natural Language Queries

HELEN MENG, PO-CHUI LUK, AND KUI XU
The Chinese University of Hong Kong,
AND
FULIANG WENG
Robert Bosch Corporation

This article presents an approach for parsing natural language queries that integrates multiple subparsers and subgrammars, in contrast to the traditional single grammar and parser approach. In using *LR(k)* parsers for natural language processing, we are faced with the problem of rapid growth in parsing table sizes as the number of grammar rules increases. We propose to partition the grammar into multiple subgrammars, each having its own parsing table and parser. Grammar partitioning helps reduce the overall parsing table size when compared to using a single grammar. We used the GLR parser with an LR(1) parsing table in our framework because GLR parsers can handle ambiguity in natural language. A parser composition technique then combines the parsers' outputs to produce an overall parse that is the same as the output parse of single parser. Two different strategies were used for parser composition: (i) parser composition by cascading; and (ii) parser composition with predictive pruning.
Our experiments were conducted with natural language queries from the ATIS (Air Travel Information Service) domain. We have manually translated the ATIS-3 corpora into Chinese, and consequently we could experiment with grammar partitioning on parallel linguistic corpora. For English, the unpartitioned ATIS grammar has 72,869 states in its parsing table, while the partitioned English grammar has 3,350 states in total. For Chinese, grammar partitioning reduced the overall parsing table size from 29,734 states to 3,894 states. Both results show that *grammar partitioning* greatly economizes on the overall parsing table size. Language understanding performances were also examined. Parser composition imparts a robust parsing capability in our framework, and hence obtains a higher understanding performance when compared to using a single GLR parser.

Categories and Subject Descriptors: I.2 [**Artificial Intelligence**]: Natural Language Processing - *Language parsing and understanding*
General Terms: Algorithms, Experimentation, Performance
Additional Key Words and Phrases: Generalized LR parsing, grammar partitioning, parser composition, lattice with multiple granularities

This work is primarily supported by the Intel Corporation, and was done mostly while the author was with Intel China Research Center.
Authors' addresses: Helen Meng, Po-Chui Luk and Kui Xu, Human-Computer Communications Laboratory, Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, Hong Kong, SAR, China; email: {hmmeng, pcluk, kxu}@se.cuhk.edu.hk; Fuliang Weng, Research and Technology Center, Robert Bosch Corporation, Palo, Alto, CA 94304; email: fuliang.weng@rtc.bosch.com.
Permission to make digital/hard copy of part of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date of appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.
© 2002 ACM 1530-0226/02/0600-0123 $5.00

1. INTRODUCTION

Natural language processing (NLP) is a desirable means for human-computer interaction, since it provides a natural, intelligible, and effective way for humans to interact with computers, and requires no specialized learning and training. Parsing is an important component technology in natural language systems. Many practical systems involve parsing technology, such as machine translation, speech recognition, language understanding, etc. The complexity of a parser grows quickly, and sometimes even exponentially, with the given grammar size; this is problematic for applications with large grammars. In order to balance parsing size and efficiency and improve accuracy, various approaches have been proposed, including a *modular parsing approach*. In this article, we report our research results on modular parsing through grammar partitioning and parser composition.

Modular parsing architectures are receiving an increasing amount of attention. There are a number of illustrative examples: Abney [1991] proposed a two-level chunking parser, which first converts an input sentence into chunks and then converts these chunks into a parse tree by means of an attacher. Kita et al. [1990; 1991; 1993] proposed an extended LR parsing algorithm, i.e., LR parsing with a category reachability test (the LR-CRT algorithm) for speech recognition. Their system uses two kinds of grammars: an intraphrase grammar and an interphrase grammar for two-level parsing through two-level symbol prediction, i.e., phrase category prediction and phone prediction. Amtrup [1995] introduced an approach that distributes grammar rule applications in multiple processors within a chart-parsing framework. Weng and Stolcke [1995] presented a general schema for partitioning a grammar into subgrammars, and the combination of subparsers for all subgrammars to achieve modular parsing. Ruland et al. [1998] developed a multiparser multistrategy architecture for noisy spoken languages.

The Generalized LR (GLR) parser is efficient and can handle ambiguity[1] by compactly storing all the alternatives [Tomita 1985; 1986]. However, Earley [1968] had shown previously that the number of states of LR parsers can grow exponentially with the size of a context-free grammar. Moore [2000] pointed out that an excessively long time is required to compute the LALR(1)[2] parsing table for a monolithic Penn Treebank grammar and to parse the test sentences in Penn Treebank using the GLR parser. This hampers the use of the GLR parser for large-scale natural language systems. In order to tackle this problem, we propose to adopt the *grammar partitioning* approach, where a large grammar is partitioned into multiple subgrammars. Our results on the ATIS data set show that this approach reduces the total number of states of the parsing tables by an order of magnitude and reduces the computational time for generating the parsing tables at the expense of less than a 50% increase in processing time. In our parsing framework, all parsing tables are constructed separately to achieve modular parsing. Hence, should there be a particular subgrammar in need of modification, we can simply regenerate its corresponding parsing table; which saves us the effort of regenerating the *entire* parsing table for the *whole* grammar. This eases the process of grammar development, promotes grammar reuse, and facilitates the use of dynamic online grammars.[3] All these factors

[1] A sentence is ambiguous if it has multiple parses or representation.
[2] LALR (lookahead LR) is one of methods for constructing an LR parsing table for LR/GLR parser.
[3] A dynamic grammar is a grammar on demand. Some commercial systems, such as Nuance Speech Recognizer, support dynamic grammars.

improve the scalability of natural language understanding systems to more complex domains. In the next stage of this framework, each subgrammar produced after grammar partitioning has its own corresponding subparser, and we use a *parser composition* approach [Luk et al. 2000] to combine the subparser outputs to produce an overall parse for the input sentence. *Parser composition by cascading* allows the subparsers to start and end at any position in the input sentence, and hence imparts robustness in handling extra-grammaticalities during parsing. However, cascading has the problem of over-generation of semantic and syntactic structures during parsing. An alternative parser composition method – parser composition with predictive pruning – offers a solution to this problem by enforcing top-down constraints, which improves parsing speed, but at the expense of reduced robustness.

The remainder of this article is organized as follows: Section 2 describes some previous work in GLR parsing; Section 3 reviews some concepts of grammar partitioning; Section 4 presents two parser composition algorithms – *parser composition by cascading* and *parser composition with predictive pruning* – to combine subparsers to obtain a single overall parse; Section 5 presents empirical results from parsing experiments based on English and translated Chinese queries from the ATIS domain; Section 6 describes our recent effort in designing a new and improved parser composition algorithm, which is a hybrid between cascading and predictive pruning; conclusions and future directions are provided in Section 7.

2. GLR PARSING

Since we have adopted GLR parsers as our basic parsing mechanism, this section reviews the basic principles of the LR parsing technique [Aho et al. 1986].

An LR(k) parser is an efficient bottom-up parser (i.e., it builds the parse tree in a bottom-up fashion) that can parse with context-free grammar rules. The parser consists of a linear stack, a parsing program, and a parsing table. The linear stack is used to store the grammar and state symbol during parsing. The parsing program reads its input from left to right while constructing a rightmost derivation of the input string using a lookahead system involving *k* symbols. The parsing table is precompiled from a given grammar and is constructed by an LR parser generator such as YACC [Johnson et al. 1975].[4]

LR parsers were originally developed for parsing programming languages, and are not intended to handle the ambiguities in natural language. Generalized LR (GLR) parsing, introduced by Tomita [1985], provides a general and efficient method for dealing with ambiguities, specifically action conflicts in the parsing table. Instead of a linear stack in an LR parser, a *graph-structured stack* (GSS) is used to simulate nondeterminism. In the algorithm, Tomita also used a compact data structure, *the packed shared parse forest,* to represent multiple parse trees for an ambiguous sentence. Shann [1991] compared the GLR parser with four different chart parsers: top-down, left-corner, Cocke-Kasami-Younger (CKY) [Younger 1967], and bidirectional [Steel et al. 1987], and found that in most cases the GLR parser performed better for highly ambiguous grammars.

[4] YACC is available as a command on the UNIX system, and is used to help implement compilers.

3. GRAMMAR PARTITIONING

Our grammar partitioning is based on the definitions in Weng and Stolcke [1995], which is a generalization of Korenjak's partitioning scheme [Korenjak 1969]. The former partitions a grammar based on its production rules, while the latter does the partition based on the nonterminals of a grammar. Our approach differs from previous work such as Abney's [1991] and Kita's [1990;1991;1993], in that theirs only allows two-level parsing, while our approach allows any number of levels/subgrammars, with possibly cyclic interaction between two subgrammars.

For illustration, we partitioned the grammar based on the nonterminals of a context-free grammar (CFG). The concept of the *virtual terminal* is introduced, and each virtual terminal is prefixed with *vt*. The virtual terminal is essentially a nonterminal, but acts as if it were a terminal [Korenjak 1969; Weng 1993]. The interaction among different sub-grammars is through two nonterminal sets, INPUT and OUTPUT, and a *virtual terminal* technique:

INPUT: This is a set of virtual terminals that were previously parsed by other sub-grammars. These virtual terminals, *vtA,* appear on the right-hand side (RHS) of some rules in the current subgrammar. *A* is a nonterminal node that appears on the left-hand side (LHS) of *some other* subgrammar(s).

OUTPUT: This is a set of nonterminals, e.g., *B,* that were parsed based on the current subgrammar. *B* appears on the LHS of some rules in the current subgrammar. Their corresponding virtual terminals, in this case *vtB*, are used by *other* subgrammars as their INPUT symbols.

In other words, we may view a partitioned subset of the production rules of a grammar as a single-value function; it takes the virtual terminals in the INPUT set [5] and returns a nonterminal in the OUTPUT set.[6] There is an ambiguity if a nonterminal that contains more than one parse tree is returned.

Korenjak [1967] gave some useful guidelines for grammar partitioning. As an initial step towards automatic grammar partitioning, Weng et al. [2000] also proposed a few guidelines, e.g., partition a grammar into subgrammars so that there are frequent interactions within subgrammars, but few interactions between subgrammars. Due to the lack of annotated training data in the ATIS domain, we manually partitioned grammars for the initial experiments.

Each partitioned subgrammar is assigned a *level index (ID)*. The level ordering criteria are as follows: a subgrammar is assigned to level *i* if its inputs are virtual terminals with level <*i* and their interaction is acyclic; two subgrammars are assigned to the same level index if they call each other. The master subgrammar is assigned with the highest-level index. The lowest-level index should be zero. It is possible that all sub-grammars except the master subgrammar are assigned to the same level due to recursive calling between subgrammars. This feature is particularly important for partitioning ambiguous syntactic grammar. The purpose of the level index is to avoid lower-level sub-

[5] For the simple cases, we replace all the occurrences of the nonterminals *A* in the INPUT set of subgrammar G_i with *vtA*. In general, we simply add rules ($A \rightarrow vtA$) to the G_i.

[6] For the simple cases, each subgrammar contains only one nonterminal *B* in OUTPUT set. In general, rules ($DUMMY \rightarrow B$) are added to subgrammar G_i, in order to output more than one nonterminal *B* in the OUTPUT set.

G_s:
1. S→NP VP
2. S→S PP
3. NP→n
4. NP→det n
5. NP→NP PP
6. PP→prep NP
7. VP→v NP

⇒

G_0:
INPUT = {vtNP, vtPP}
OUTPUT = {S}
1. S→vtNP VP
2. S→S vtPP
3. VP→v vtNP

G_1:
INPUT = {vtPP}
OUTPUT = {NP}
1. NP→n
2. NP→det n
3. NP→NP vtPP

G_2:
INPUT = {vtNP }
OUTPUT = {PP}
1. PP→prep vtNP

Fig. 1. The entire grammar G_S is partitioned into subgrammars G_0, G_1, and G_2.

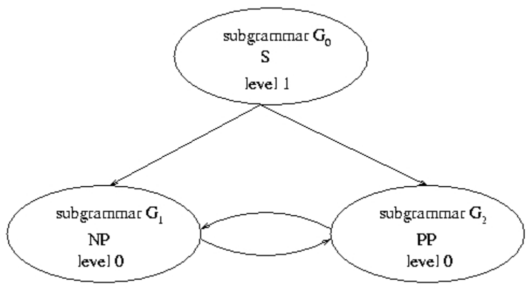

Figure 2. The calling graph of the subgrammars G_0, G_1, and G_2.

parsers (e.g., its subgrammar only contains preterminal rules) to parse the higher-level subparsers' output nodes. This way, the parser could save some unnecessary parsing operations.

We use the toy grammar in Figure 1 to illustrate a grammar partitioning.[7] The grammar G_S is partitioned into the subgrammars G_0, G_1, and G_2. Grammar terminals are in lowercase, and nonterminals in uppercase. We select the nonterminals *NP* and *PP* for grammar partitioning. The production rules with *NP* and *PP* are partitioned into subgrammars G_1 and G_2, respectively. In addition, for all production rules that have *NP* on their RHS, we replace *NP* with *vtNP* (see G_S and G_0). Similarly, for all production rules that have *PP* on their RHS, we replace *PP* with *vtPP* (see G_S and G_0). Hence, we end up with three sets of grammar rules G_0, G_1, and G_2 in *reduced form*, i.e., the production rules have no unused terminals or nonterminals. We used an LR(1) parsing table generator to

[7] This grammar is borrowed from Tomita [1986].

construct LR(1) parsing tables for these three partitioned subgrammars separately. The number of states in each LR(1) table derived from G_S, G_0, G, and G_2 is 19, 7, 6, and 4, respectively. We found that the total number of all the states in the three subgrammars G_0, G_1, and G_2 (=17) is smaller than that of G_S (=19).

The calling graph of these three subgrammars is shown in Figure 2, which captures their INPUT and OUTPUT relationships. The INPUT of G_0 are *vtNP* and *vtPP*, the OUTPUT of G_1 and G_2 are *PP* and *NP*, respectively. The directed edges *E* from G_0 to G_1 and from G_0 to G_2 are added to the calling graph. In the same way, there are two edges between G_1 and G_0. The level indices are assigned according to our level ordering criteria. Because G_1 and G_0 call each other in this case, they are assigned with the same level index.

4. PARSER COMPOSITION

As mentioned previously, a grammar is partitioned into several subgrammars, and each subgrammar operates with its own specialized subparser. We need to compose all subparsers in order to obtain an overall parse of the input. Each subparser is a GLR parser and contains an LR(1) parsing table. A *lattice with multiple granularity* (LMG) is introduced to serve as an interface among different subparsers. We have experimented with two parser composition algorithms, *parser composition by cascading* and *parser composition with predictive pruning*, which are described in this section.

4.1 Lattice with Multiple Granularity

With grammar partitioning of subparsers, our parsing framework requires an interface to record the INPUT and OUTPUT of all subparsers. This role is fulfilled by the LMG, which is a directed acyclic graph (DAG). The LMG has nodes that may be either terminals or virtual terminals, and the nodes are connected by transitions. Each node is assigned a name and an index. There is a sentence START node <*s*> which is assigned an index 0, and a sentence END node '$' which is assigned an index *n+1*, when *n* is the sentence length. Figure 3 shows an example of an LMG. The virtual terminal node *vtNP* is an OUTPUT of a subparser; this node covers two terminals nodes, *det* and *n*. Both terminals and virtual terminals are represented in the same lattice, but 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, which is why the structure is termed a *lattice with multiple granularity*.

Since a subparser needs to process an LMG as input, we modified our GLR parsing algorithm in a way similar to that proposed by Tomita [1986]. This allowed our GLR parsers to handle both the lattice structure and a string as input. The input LMG is parsed in topological order. Many subgrammars produce parses that cover a subsequence of the input string, and hence their subparsers need to decide where the parses (or subtrees) end within a sentence. A straightforward implementation is to consider the possibility of a robust END symbol '$' whenever we read in a new input symbol (terminal or virtual terminal) from the LMG.

4.2 Parser Composition by Cascading

Parser composition by cascading is a bottom-up parsing algorithm. The mechanism takes an input sentence and converts it into an LMG. The parsers at the lowest level are

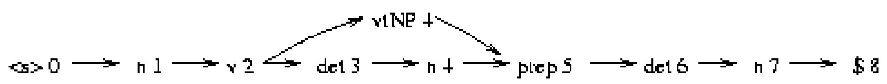

Fig. 3. An example of a lattice with multiple granularities (LMG) based on the sentence, “I saw a man with a telescope.” The nodes in the LMG are part of speech tags for the sentence.

activated to parse the LMG, and leave their corresponding virtual terminals on the LMG if they parse successfully. When all subparsers at given level have finished their processing at all nodes in the LMG, the cascading process moves to the next level. This parsing process continues until the top-level (SENTENCE) subparsers are activated. When no top-level subparser is successful, the sequence of the chunks that has the highest score is returned. Notice that, during the parsing process, score computation for selecting the best parse and pruning the bad parses uses the shortest-path algorithm. So, essentially, there is no separate module or pass for cases when the grammar does not cover the sentences. Since parsing starts from the lowest (terminal) level and proceeds to the highest level, level by level, it is called *cascading*.

We use a stack to determine the sequence of invocation of subparsers. The stack stores the terminals and virtual terminals according to the topological order in the LMG. During parsing, newly created virtual terminals are added dynamically to the stack. Each subparser starts to parse the (virtual) terminal on the top of stack in topological order, and this (virtual) terminal is popped off the stack after all subparsers are activated. All the subparsers share the same stack.

As an illustration, we parse the sentence *n v det n prep det n* with the partitioned grammars in Figure 1. Subgrammars G_1 and G_2 are assigned to level 0, and subgrammar G_0 to level 1. Figure 4 shows how the LMG changes by subparsers in level 0. In the following, we describe Figure 4 in detail.

For the first diagram in Figure 4, the sequence of the stack from top to bottom is ! *n, det, prep, n, det, v, n*. The exclamation mark indicates the top of the stack. The subparsers of G_1 and G_2 start to parse at *n* (index 7), which is on the top of the stack. Since the subparser of G_1 can successfully parse phrase *n* in the LMG, its output virtual terminal *vtNP* is created on the LMG. Thus, *n* was popped off the stack. The newly created *vtNP* is pushed onto the stack and added on the LMG. The stack becomes ! *vtNP*, *det*, ..., *n*. Next, the subparsers of G_1 and G_2 start to parse at *vtNP*, which is popped off the stack. Since no subparsers can successfully parse at *vtNP*, no virtual terminal can be created.

For the second diagram, the stack becomes ! *det, prep, n, det, v, n*. The subparsers of G_1 and G_2 start to parse at *det*. The subparser of G_1 can successfully parse the phrase *det n*. The output *vtNP* is added on the LMG.

For the third diagram, the content of the stack is ! *prep, n, det, v, n*. The subparser of G_2 can parse phrase *prep*, *vtNP*. The output *vtPP* is added on the LMG.

For the fourth and fifth diagrams, the content of the stack at this stage is ! *n, det, v, n*. The subparser of G_2 can parse the LMG in two ways. The first one ends before *prep*

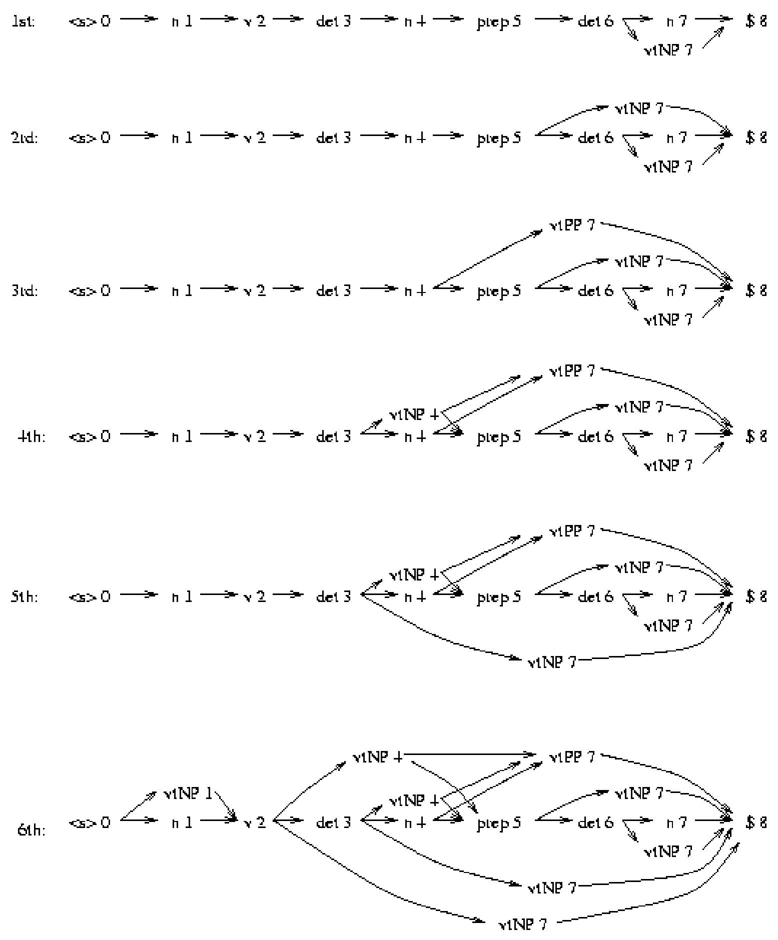

Fig. 4. Changes in the LMG due to subparsers in level 0 through parser composition by cascading.

(index 5), and the other one ends before '$' (end of the input string). Therefore, two *vtNP*s are added on the LMG, respectively.

The subparsers continue to operate in the same manner. The sixth diagram is the final stage of the LMG at level 0.

Next, the control flow advances to level 1, with the stack refreshed. The parsing process continues until the master parser G_0 is activated at every node in the LMG. We find the master parser can successfully parse the whole sentence, i.e., the master parser can create a virtual terminal *vtS* that covers the whole sentence. The resulting parse tree, which is attached to this *vtS*, is shown in Figure 5. From this example, we can see that our parsing framework supports a grammar that contains ambiguous production rules.

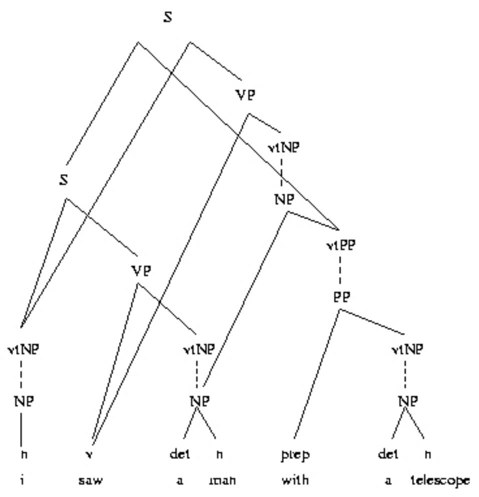

Fig. 5. The resulting parse forest from the subparser G_0.

4.3 Parser Composition with Predictive Pruning

Parsing composition by cascading invokes subparsers at every position in the LMG. Some of the virtual terminals created upon invocation do not enter the final parse forest. To avoid excessive invocation, we implemented an alternative parser composition strategy: *parser composition with predictive pruning*.

The parsing procedure also begins by converting the input sentence into an LMG. When a caller subparser reads a node (either a virtual terminal or a terminal) from the LMG, the to-be-activated subparser must satisfy the following constraints: nonerror action is obtained from the caller's parsing table, and the input node is a left-corner node of the subparser. Table I shows the left-corner nodes of the virtual terminals corresponding to our partitioned grammars. Satisfying the given constraint implies that the invoked subparser is more likely to return a virtual terminal *predicted* by the caller subparser, i.e., ACTION[caller's current state,*vt*]$\neq$ error. Notice that the left corner (*w*) can be either a terminal or a virtual terminal. If $vt \overset{+}{\Rightarrow} w...$, we can say *w* is the left corner of *vt*. The sign ($\overset{+}{\Rightarrow}$) denotes a derivation consisting of one or more steps. For our implementation, the left-corner nodes for every virtual terminal are precomputed prior to parsing. Subparsers in the caller's INPUT set that do not satisfy the predictive condition are pruned. Parsing starts with the master GLR subparser at the leftmost lattice node and ends when the final node of the LMG is reached. Since the activated subparsers must satisfy the caller subparser's predictive constraint and the ones that do not satisfy the constraint are pruned, this parser composition strategy is termed *predictive pruning*. Details of this parsing algorithm are presented in Weng et al. [2000]. Our prediction is similar to phrase category prediction used in Kita's approach [Kita et al. 1993]. However,

Table I
(The left-corner sets that correspond to each virtual terminal in the subgrammars of Figure 1)

Virtual terminal (*vt*)	**Left-corner nodes (*w*)**
vtNP (OUTPUT of G_1)	*n, det*
vtPP (OUTPUT of G_2)	*prep*

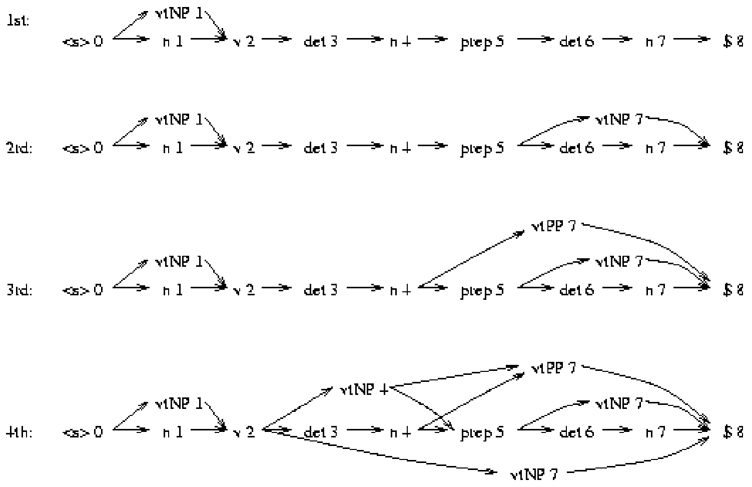

Fig. 6. Changes in the LMG by parser composition with predictive pruning.

Kita's work is state-based prediction for symbols, while our approach uses node-based prediction for activated subgrammars.

In predictive pruning, the stack is used to determine not only the sequence of invocation of subparsers, but also the parsing order. The LMG nodes are sorted topologically and the terminal nodes are pushed onto the stack in *reverse order*. Newly created virtual terminals are also dynamically pushed onto the stack.

As an illustration, we parse the same sentence, *n v det n prep det n*, using the partitioned grammars in Figure 1 by predictive pruning. Figure 6 shows how the LMG changes by subparsers at level 0. In the following, we describe Figure 6 in detail.

For the first diagram in Figure 6, the stack from top to bottom at this stage is ! *n, v det, n, prep, det, n, '$'*. The exclamation mark indicates the top of the stack. The master subparser, G_0, is activated to parse the LMG at node *n* (index 1), which is at the top of the stack. The subparser of G_1 satisfies the master parser's predictive constraints, since ACTION[initial state of GSS: 0, *vtNP*]≠ error from the parsing table of G_0, and *n* is the left corner of *vtNP* according to Table I. Therefore, the subparser of G_1 is activated at *n*. Since the subparser of G_1 can successfully parse phrase *n* in the LMG, it returns a virtual terminal with label *vtNP,* which will be added on the LMG as shown in the first diagram in Figure 6; it also pushes *vtNP* onto the top of the stack.

In the second diagram, the stack becomes ! *vtNP*, *v*, *det*, *n*, *prep*, *det*, *n*, '*$*'. The master subparser starts to parse at *vtNP*. It shifts *vtNP* and *v* onto its graph-structured stack (GSS). When it reads *det* (index 3) as the next input, the subparser of G_1 satisfies the predictive condition of the master parser. Then the subparser of G_1 is invoked at *det*. It can end at *n* (index 4), or continue to parse the next input, *prep*. When the subparser G_1 continues to parse *prep*, it calls the subparser of G_2 at *prep*. Next, the subparser G_2 calls the subparser G_1 at *det* with index 6. Since the subparser G_1 can successfully parse the sequence "*det n*", its created output, *vtNP,* is added on the LMG, as shown in the second diagram, and pushed onto the stack.

Control returns to subparser G_2, which started to parse *prep*. The subparser can successfully parse the LMG *prep*, *vtNP*. Therefore, *vtPP* is added on the LMG, as shown in the third diagram.

Next, control returns to the subparser G_1 which started to parse at *det* with index 3. This subparser can produce two virtual terminals, as shown in the fourth diagram.

Finally, the master parser can parse phrase *vtNP*, *v*, *vtNP* and create *vtS* to cover the whole sentence.

Thus far we have stepped through the processes of parser composition by cascading (Section 4.2), and parser composition by predictive pruning (Section 4.3) on the basis of the same input sequence. We see that cascading produces more virtual terminals than predictive pruning. However, both of these algorithms can create the same parse forest, as shown in Figure 6, after the master parser is called. This means some virtual terminals created by cascading do not contribute to the resulting parse forest.

5. EXPERIMENTS

We applied our parsing composition framework to the natural language queries in the Air Travel Information Systems (ATIS) corpus [Price 1990]. ATIS is the common project of a DARPA-sponsored research program.[8] We conducted a comparative study in parsing ATIS queries and experimented with the following:

1. a single GLR parser with an unpartitioned grammar; and
2. multiple subparsers with partitioned grammars, using both parser composition techniques, i.e., cascading as well as predictive pruning.

In this section, we present experimental results based on

1. grammar coverage;
2. parsing table size;
3. computational costs; and
4. natural language understanding performance.

5.1 Experimental Corpus

Our experiments utilize English ATIS-3 Class A queries. Class A queries are self-contained and can be processed individually without considering the discourse context.

[8] The ATIS data corpus can be purchased from the Linguistic Data Consortium. (http://www.ldc.upenn.edu).

Table II
Examples of English Sentences in the ATIS-3 Training Corpus
(together with their Chinese translations in spoken Cantonese style)

English: *Show me the most expensive one way flight from detroit to westchester country*	
Chinese: 話俾我知由底特律飛去西赤斯城最貴既單程航機	
English: *Show me the flights arriving on baltimore on june fourteenth*	
Chinese: 話俾我知係六月十四號飛到巴的摩爾既航機	
English: *Show me all united airlines first class flights*	
Chinese: 話我知所有聯合航空既頭等航班	

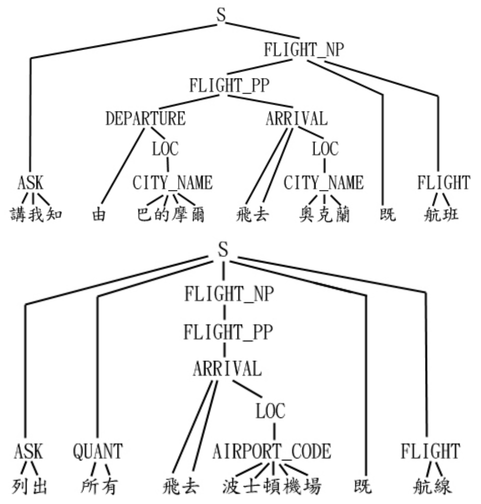

Fig. 7. Example output Chinese parse trees from our parser.

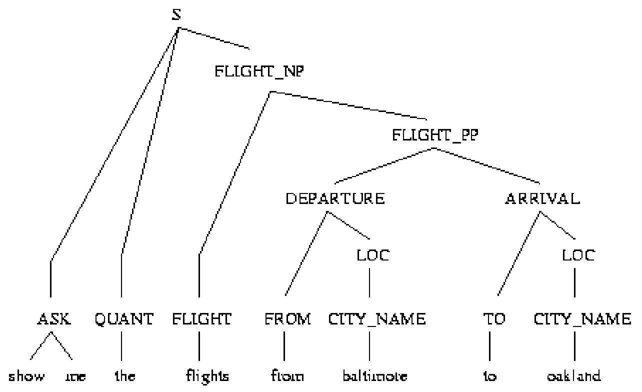

Fig. 8. Example output English parse tree from our parser.

The training set has 1564 queries, the 1993 test set has 448 queries, and the 1994 test set has 444 queries. Each query is labeled with a corresponding SQL query for database access. We translated the English queries into Chinese, and hence obtained a parallel corpus for our experiments. The translations are in *spoken Cantonese style*. Cantonese is a major Chinese dialect used in Hong Kong, South China, and in many overseas Chinese communities. Example queries are shown in Table II. Example output English and Chinese parse trees are shown in Figure 8 and Figure 7, respectively.

5.2 ATIS Grammar Development

Our ATIS grammar is a set of context-free rules. The grammar contains both semantic and syntactic structures. The low-level grammar rules are mainly semantic concepts typical of the ATIS corpus, such as CITY-NAME, CLASS-TYPE, MONTH-NUMBER, etc., obtained by a semiautomatic grammar induction algorithm [Siu et al. 1999], which uses a statistical clustering approach. The clustering algorithm includes spatial clustering and temporal clustering. Spatial clustering intends to extract semantic catergories by minimizing the Kullback Liebler distance. Temporal clustering aims to capture key phrases by maximizing mutual information. These two clustering procedures perform alternately and iteratively to form an induced grammar. After hand-editing, our semi-automatic grammar is formed. Siu et al. [2001] also ported their clustering algorithm to a translated ATIS corpus in Cantonese Chinese. The major difference between Chinese and English is that there are no word boundaries in Chinese sentences. Therefore, in order to obtain a Chinese grammar, tokenization is prior to grammar induction. The high-level grammar rules mainly describe phrases, such as a time phrase, flight preposition phrase, etc. The SENTENCE-level grammar rules, however, are generated using a data-driven approach, as follows:

We applied parser composition by cascading, where each training sentence is processed by the subparsers to form a lattice. The SENTENCE-level rules are obtained by traversing the "best" path through the lattice using the shortest-path algorithm [Hillier et al. 1995]. Such rules maximize the coverage of our training set. Thus, we formed grammars for the English and Chinese corpora with the unpartitioned and partitioned grammar sizes listed in Table III. Examples of English and Chinese ATIS-3 rules are as follows:

∗ S :=ASK FLIGHT_NP|AIRLINE_CODE FLIGHT_PP|QUEST TIME_NP|...

∗ ASK :=show me|list|tell me|give me|...

∗ FLIGHT_NP :=FLIGHT_NP AND FLIGHT_NP|FLIGHT FLIGHT_PP|...

∗ FLIGHT_PP :=which FLIGHT_PP|that FLIGHT_PP|ARRIVAL|DEPARTURE|...

∗ ARRIVAL :=TO LOC|TO TIME_NP|TO TIME_NP in LOC|...

∗ CITY_NAME:=westchester|atlanta|chicago|milwaukee|...

♦ S :=AIRLINE QUEST|AIRLINE_CODE 既 QUEST|ASK FLIGHT_NP|...

♦ ASK :=我 想 搵|話 比 我 知|話 我 知|請 問|...

♦ FLIGHT_NP :=[會] [係] FLIGHT_PP [既] FLIGHT|...

♦ FLIGHT_PP :=ARRIVAL|ARR_DEPART|BETWEEN|DEPARTURE ARRIVAL|...

♦ CITY_NAME :=亞 特 蘭 大|巴 的 摩 爾|波 士 頓|...

Table III
Grammar Statistics (based on the original unpartitioned grammar and the partitioned grammar)

Grammar Statistics	No partitioning		Partitioning	
	English	Chinese	English	Chinese
No. of rules	1650	1538	1818	1637
No. of SENTENCE-level rules	337	508	337	508
No. of terminals	602	515	602	515
No. of non-terminals	97	85	97	85
No. of virtual terminals	N/A	N/A	65	63
Total number of states in parsing table	72,869	29,734	3,350	3,894

5.3 ATIS Grammar Partitioning

Based on our grammar partitioning scheme as described previously, we manually partitioned our grammar rules into subgrammars, and created virtual terminals such as vtSTATE-NAME, vtCITY-NAME, vtAIRPORT-NAME, etc. Most of these are nonterminals corresponding to semantic concepts. The calling graph of our partitioned grammars is a directed acyclic graph.

The sizes of the partitioned grammars are listed in Table III (shaded portions). We observe that the sizes of the English and Chinese grammars are comparable across the parallel corpora. Grammar partitioning increases the total number of grammar rules because some rules are duplicated across multiple subgrammars. For instance, production rules (DIGIT$\rightarrow$ *one*) may appear in the subgrammar of COST, TIME, etc. The duplicated nonterminals do not constitute the OUTPUT of any subgrammars.

5.4 Parser Composition in ATIS

After grammar partitioning, we need to compose all the subparsers in order to obtain an overall parse for the input query. We used *both* parser composition algorithms to combine subparsers, i.e., parser composition by cascading and parser composition with predictive pruning. Our framework with *multiple* subgrammars and subparsers is compared with an alternative framework with a *single* GLR parser and a *single* grammar.

5.4.1 *Size of Parsing Table.* A canonical LR(1) parsing table is used in our experiments. We used the same LR(1) parsing table generator to construct the LR(1) tables for unpartitioned and partitioned grammars. The total number of states (rows) in the parsing table for partitioned grammars is the sum of the number of states in all sub-parsers. The last row of Table III shows that the unpartitioned English grammar has 72,869 states and the unpartitioned Chinese grammar has 29,734 states in their parsing tables. In comparison, the partitioned English grammar has 3,350 states and the partitioned Chinese has 3,894 states in total. Grammar partitioning greatly reduces overall parsing table sizes, and hence reduces the computation required to generate the parsing tables, the space and memory to store the tables, and time to access the tables during parsing. This result shows that grammar partitioning is desirable for large-scale natural language processing.

5.4.2 *Grammar Coverage.* Table IV shows the coverage of our grammars for both English and Chinese ATIS queries. *Full parse* means that we can produce a parse forest

Table IV
Grammar Coverage for the English and Chinese (shaded) ATIS-3 Corpora.
(CAS abbreviates parser composition by cascading; PP is parser composition with predictive pruning with partitioning; and GLR is the use of a single GLR parser with no partitioning, denoted 'Un-')

Based on the English ATIS-3 Corpus	**Training set**			**Test set 93**			**Test set 94**		
	Un-	**Partitioned**		**Un-**	**Partitioned**		**Un-**	**Partitioned**	
	GLR	**CAS**	**PP**	**GLR**	**CAS**	**PP**	**GLR**	**CAS**	**PP**
Full parse (%)	99.4	99.4	99.4	60.9	60.9	60.9	62.4	62.4	62.4
Partial parse (%)	0.0	0.6	0.5	0.0	39.1	34.2	0.0	37.6	35.1
No parse (%)	0.6	0.0	0.1	39.1	0.0	4.9	37.6	0.0	2.5
Based on the Chinese ATIS-3 Corpus	**Training set**			**Test set 93**			**Test set 94**		
	Un-	**Partitioned**		**Un-**	**Partitioned**		**Un-**	**Partitioned**	
	GLR	**CAS**	**PP**	**GLR**	**CAS**	**PP**	**GLR**	**CAS**	**PP**
Full parse (%)	98.7	98.7	98.7	44.6	44.6	44.6	57.4	57.4	57.4
Partial parse (%)	0.0	1.3	1.0	0.0	55.4	51.3	0.0	42.6	37.4
No parse (%)	1.3	0.0	0.3	55.4	0.0	4.0	42.6	0.0	5.2

covering the entire query; *partial parse* means there is at least one parse chunk covering a part of a query; *no parse* means there is no parse at all for the query.

The first row in Table IV shows that the different parsing strategies achieve the same full-parse coverage in English and Chinese queries. This reflects the consistency of our parsing framework. Full-parse coverage for English sentences varies between 60% to 62%; while that for Chinese sentences is lower, at 44% to 57% in the test sets. Since our SENTENCE-level grammar rules are derived from the shortest paths through the training lattices, they tend to be rather specific in structure, and may contribute to the higher full-parse coverage in the training set and the lower full-parse coverage in the test sets.
The single GLR parser approach cannot create any partial parse. For the parser composition approaches, composed subparsers place virtual terminals on the LMG if they can successfully generate subparses. Hence, partial parses can be obtained from both parser composition methods. When parser composition by cascading is used, a subparser can be activated at any position of the input query. A partial parse can also end at any position of the input query. When parser composition with predictive pruning is used, subparsers are activated only if they satisfy the predictive constraints, and the parsing process terminates when the master subparser reaches the sentence END or detects a grammatical error. Therefore, cascading obtains the highest percentage of partially parsed queries, followed by predictive pruning; and the single GLR obtains no partially parsed queries.

5.4.3 *Natural Language Understanding Performance*. In order to evaluate natural language understanding performance, the output parses are converted to semantic frames, each of which contains a list of key-value pair(s) representing the meaning of the sentence. The keys are designed with reference to the schema of SQL queries (for database access), which are provided in the ATIS corpus, e.g., CITY-NAME, FLIGHT-NUMBER, CLASS-TYPE. Producing the semantic frame is straightforward for the single GLR approach, since it outputs a single parse forest or tree. With grammar partitioning and parser composition, the output of the parsers is an LMG. We can find multiple paths from the sentence START <*s*> to sentence END '\$'. We apply the shortest-path algorithm

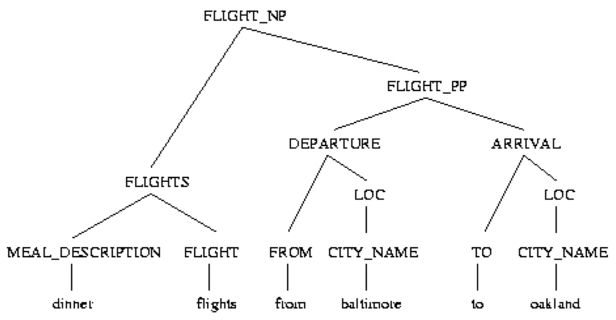

Fig. 9. The parse tree is attached to a virtual terminal vtFLIGHT_NP.

[Hillier et al. 1995] to find the "best path" in the LMG for deriving the semantic frame. The "best path" has the minimum total cost from the sentence START to sentence END. The cost of a path segment through the directed acyclic graph is computed as shown in Equation (1).

$$cZA = t - i \tag{1}$$

where

c is the cost of the path segment;
t is the total number of levels in our partitioned grammar (a constant);
I is the grammar level of the target node of the path segment.

Hence, a path that traverses through virtual terminals with subgrammars at higher levels is preferred over an alternate path that traverses through virtual terminals with subgrammars at lower levels. Only virtual terminals are considered in the "best path," since they carry semantic content. Our semantic interpreter walks through the parse tree that is attached to the virtual terminal in the best path and extracts semantic information to form a semantic frame. For example, Figure 9 shows a parse tree that is attached to a virtual terminal vtFLIGHT_NP in the best path. The generated semantic frame is shown in Table V. The contents of the frame are compared with the "reference" semantic frame, derived from the list of attributes in the corresponding SQL query from the ATIS corpus.

Results for natural language understanding are shown in Table VI. *Full match* refers to queries with exact matches between the generated semantic frame and the reference semantic frame; *partial match* refers to cases where only a fraction of the concepts between the semantic frame and reference frame match, and insertion, deletion, and substitution errors are all penalized; *no match* refers to cases where the concept error rate equals or exceeds 100% for the sentence.[9]

Sentences that obtain a full parse may not achieve a full match in their semantic interpretation, mainly due to the extra information incorporated into the reference frames. This fact is known as the "Principle of Interpretation" (PI) in the ATIS evaluation community. For example, the word "tonight" represents "time>=1800" and "time<2359"

[9] Insertion errors may cause the concept error rate to exceed 100%.

Table V
The Semantic Frame is Generated from the Parse Tree in Figure 9

MEAL_DESCRIPTION=dinner
DEPARTURE_CITY_NAME=baltimore
ARRIVAL_CITY_NAME=oakland

Table VI
Performance in Language Understanding of the English and Chinese ATIS-3 Corpora.
(CAS abbreviates parser composition by cascading; PP is parser composition with predictive pruning with partitioning; and GLR is use of a single GLR parser with no partitioning)

Based on the English ATIS-3 Corpus	**Training set**			**Test set 93**			**Test set 94**		
	Un-	**Partitioned**		**Un-**	**Partitioned**		**Un-**	**Partitioned**	
	GLR	**CAS**	**PP**	**GLR**	**CAS**	**PP**	**GLR**	**CAS**	**PP**
Error rate in semantic concepts (%)	8.0	7.5	7.8	41.2	9.9	33.8	40.4	11.7	29.1
Full match (% of sentences)	82.4	82.7	82.4	56.3	87.7	60.5	54.5	77.0	59.2
Partial match (% of sentences)	15.0	15.3	15.2	4.0	8.0	9.8	7.7	18.9	20.3
No match (% of sentences)	2.6	2.0	2.4	39.7	4.2	29.7	37.8	4.1	20.5
Based on the Chinese ATIS-3 Corpus	**Training set**			**Test set 93**			**Test set 94**		
	Un-	**Partitioned**		**Un-**	**Partitioned**		**Un-**	**Partitioned**	
	GLR	**CAS**	**PP**	**GLR**	**CAS**	**PP**	**GLR**	**CAS**	**PP**
Error rate in semantic concepts (%)	10.3	7.9	8.6	58.6	11.0	25.6	44.9	12.7	28.4
Full match (% of sentences)	77.6	80.8	80.2	37.5	78.6	61.6	49.8	72.7	58.6
Partial match (% of sentences)	18.5	16.9	17.1	6.0	16.7	23.2	7.4	22.3	24.1
No match (% of sentences)	3.9	2.2	2.7	56.5	4.7	15.2	42.8	5.0	17.3

Table VII
Examples of Errors in the Generated Semantic Frames

Input: *i'd like to fly late tomorrow from Minneapolis to long beach*	
Reference Semantic Frame (from given SQL query)	**Hypothesized Semantic Frame (from parse output)**

city_name=minneapolis
city_name=long beach
departure_time>=2000
departure_time>=2359
departure_time>=0
departure_time>=300
	departure_city_name=minneapolis
	arrival_city_name=long beach

Input: *show me the ground transportation in the salt lake city airport*	
Reference Semantic Frame (from given SQL query)	**Hypothesized Semantic Frame (from parse output)**
city_name=salt lake city	airport_code=the salt lake city airport
Input: *which flights arrive in saint louis from saint paul on thursday morning*	
Reference Semantic Frame (from given SQL query)	**Hypothesized Semantic Frame (from parse output)**

city_name=st. paul
city_name=st. louis
arrival_time>=0
arrival_time>=1200
	departure_city_name=st. paul
	arrival_city_name=st. louis
	arrival_time=morning

Table VIII
Computational Costs for the English and Chinese (shaded) ATIS-3 Corpora.
(CAS abbreviates parser composition by cascading; PP is parser composition with predictive pruning with partitioning; and GLR is the use of a single GLR parser on partitioning. Italicized percentages in parentheses are savings of PP relative to CAS)

Based on the English ATIS-3 Corpus	**Training set**			**Test set 93**			**Test set 94**		
	Un-	**Partitioned**		**Un-**	**Partitioned**		**Un-**	**Partitioned**	
	GLR	**CAS**	**PP**	**GLR**	**CAS**	**PP**	**GLR**	**CAS**	**PP**
Total parsing time (in s, Sun Ultra 5)	42.8	82.8	52.4 *(36.3%)*	8.2	19.2	10.9 *(43.2%)*	9.7	21.7	12.8 *(41.0%)*
Aver. no. of states visited including failed sun-parsers	60.0	134.2	95.0 *(29.2%)*	37.5	107.8	60.6 *(43.8%)*	47.7	122.9	75.7 *(38.4%)*
Aver. no. of rules reduced including failed sub-parsers only	34.5	67.5	46.6 *(31.0%)*	21.9	52.1	28.4 *(45.5%)*	27.2	59.4	36.5 *(39.6%)*
Aver. no. of states visited, successful sub-parsers only	59.8	109.7	84.3 *(23.2%)*	32.6	85.1	51.9 *(39.0%)*	39.4	97.0	65.5 *(32.5%)*
Aver. no. of rules reduced, successful sub-parsers only	34.4	67.5	46.5 *(31.1%)*	19.6	52.1	28.4 *(45.5%)*	23.3	59.3	36.5 *(38.4%)*
Based on the Chinese ATIS-3 Corpus	**Training set**			**Test set 93**			**Test set 94**		
	Un-	**Partitioned**		**Un-**	**Partitioned**		**Un-**	**Partitioned**	
	GLR	**CAS**	**PP**	**GLR**	**CAS**	**PP**	**GLR**	**CAS**	**PP**
Total parsing time (in s, Sun Ultra 5)	30.2	85.2	51.0 *(40.1%)*	5.7	17.9	11.5 *(35.8%)*	6.7	25.3	12.4 *(51.0%)*
Aver. no. of states visited including failed sun-parsers	54.9	138.8	81.4 *(41.4%)*	34.4	99.4	54.8 *(44.9%)*	42.4	145.1	65.9 *(54.6%)*
Aver. no. of rules reduced including failed sub-parsers only	22.1	43.7	25.1 *(42.6%)*	13.6	29.9	16.0 *(46.5%)*	16.7	44.8	19.6 *(56.3%)*
Aver. no. of states visited, successful sub-parsers only	54.5	96.9	67.5 *(30.3%)*	21.9	65.0	42.0 *(35.4%)*	33.6	98.4	51.9 *(47.3%)*
Aver. no. of rules reduced, successful sub-parsers only	22.0	43.5	25.0 *(42.6%)*	9.0	29.7	16.0 *(46.1%)*	13.7	44.6	19.6 *(56.1%)*

today, “time>=0” and “time<=600” the following day. By comparison, our semantic interpreter only generates the key-value pair “time=tonight” in our frame, and the PI is not available to us for incorporation in our generated semantic frames. Some examples of errors are illustrated in Table VII. As can be seen, due to the absence of PI, we tend to over-penalize ourselves in understanding.

From Table VI, we see that the single GLR approach has the highest concept error rate, since it produces no partial parse. Conversely, both parser composition algorithms produce partial parses on the LMG and attain a higher natural language understanding performance. In addition, cascading has a lower concept error rate than predictive pruning which is correlated with the parse coverage. Cascading attempts to parse chunks of the input at all lattice nodes, while predictive pruning invokes virtual terminals only if they abide the left-corner predictive constraints.

5.4.4 *Computational Costs.* We compared the computational costs among the three parsing strategies. Computational cost is measured in terms of the total parsing time, the number of states visited, and the number of rules reduced. Results are tabulated in Table VIII for the English and Chinese ATIS corpora. The total parsing time (first row) does not include the load time for data, e.g., the parsing table. We measured only the CPU time used by the parsing process. All experiments were conducted on a Sun Solaris Ultra 5 machine in order to minimize time variations due to different system architectures.

From Table VIII, we observe that the single GLR parser is the most economical parsing strategy when compared with the other two parsing methods. This is true for both Chinese and English ATIS, since the single GLR only handles a string as input, but the composed subparsers need to handle a lattice as input. Parsing a lattice takes more computation in searching, as reflected by our measurements. Besides, partitioned subparsers insert a robust END to active state nodes in the GSS during parsing, and this robustness feature also increased the parsing time. However, we should note that the single GLR parser has the largest parsing table. Hence, the parsing table required a much longer loading time compared to the two parser composition methods. Therefore, from this perspective, the single GLR parser is less desirable, and may become impractical when the grammar becomes very large.

As we migrated from cascading to predictive pruning, we observed consistent improvements in all aspects of the parsing computation. In predictive pruning, the master GLR subparser (SENTENCE-level subparser) started at the leftmost lattice nodes. For the sake of comparison, we restricted the cascading strategy such that the master subparser (SENTENCE-level subparser) could only start at the leftmost lattice node.

The total parsing times for the test sets were shortened by 35.8% to 51.0%, and the trend was maintained for the subset of the sentences with full parses, i.e., the parsing times were shortened by 19.0% to 38.9%. The number of states visited was reduced by 25.4% to 41.8% when failed subparsers are counted, and by 26.6% to 46.2% when only successful subparsers are counted. Finally, the number of rules reduced were lower by 19.8% to 36.3% when failed subparsers are counted, and by 26.5% to 46.2% when only successful subparsers are counted.

6. TOWARDS A HYBRID PARSER COMPOSITION ALGORITHM

Our ongoing effort is devoted towards improvements in natural language understanding performance, while maintaining a decent level of parsing efficiency. We modified the parser composition algorithm such that the subgrammars are not explicitly labeled with level numbers, since for highly ambiguous syntactic grammars, it is often difficult to impose strict levels to the partitioned grammars. Instead the parser treats all sub-grammars as being at equal levels. As parsing proceeds from right to left, *all* subparsers are invoked in *all* (virtual) terminals in the LMG. In an attempt to maintain efficiency, the new parser composition algorithm checks to see if the left-corner condition is satisfied with respect to the current (virtual) terminal before it invokes the subparser.[10] Hence, our new parser composition algorithm may be viewed as a *hybrid* of the cascading and predictive pruning algorithms.

While the grammar partitions do not have strict level numbers, we still need to traverse the LMG to find the "best path" from the initial node to the final node. In general, we prefer a path that involves high-level parses over low-level parses. We modified the

[10] This is different from our previous cascading algorithm, which does not enforce the left-corner constraint. Instead, as long as the current (virtual) terminal is present in the right-hand-side of the grammar rule, the sub-parser is invoked.

Table IX
Performance in Language Understanding for the English ATIS-3 Corpora.
(We use the new *hybrid* version of cascading with predictive pruning for parser composition with partitioned grammars)

Based on the English ATIS-3 Corpus	Test set 93	Test set 94
Error rate in semantic concepts (%)	5.9	8.8
Full match (% of sentences)	91.5	84.0
Partial match (% of sentences)	6.0	12.1
No match (% of sentences)	2.5	3.8

scoring algorithm to retain some notion of the subgrammar *level* of parse. Equation (2) shows our new scoring algorithm.

$$s=(s_1+s_2+\ldots+s_k)\lambda^n \qquad (2)$$

where

s	denotes the score of a parse tree;
$s_1, s_2, \ldots, s_k$	denote the score of its children if it contains k children;
λ	denotes a constant factor (set at 1.1);
n	denotes the number of terminals covered by the parse tree.

This new scoring algorithm computes a score for the parse tree attached to each node in the LMG. The score of a terminal is 1. The value of λ is chosen to be slightly above 1 to reward the grammar nodes at higher levels in the parse tree.

We have implemented this new method of parser composition and scoring, as well as re-evaluated our performance. We have also attempted to neutralize the mismatch in the application of the Principles of Interpretation (PI), which are present in the reference semantic frames but not in the generated semantic frames. For example, if the generated semantic frame is able to extract the departure time to be “mid-morning” but not the time range that corresponds to mid-morning (which is found in the reference frame), we do not penalize for the mismatch.

The latest results (in English only) are tabulated in Table IX. These results should be compared with the English results in Table VI, where *cascading* is used for parsing composition. The semantic concept error rates are between 5.9% and 8.8%, reduced by 2%-4% (absolute). There is also a general increase in the number of sentences that attain a full match in semantic concepts.[11]

7. CONCLUSIONS AND FUTURE WORK

In this article, we have presented a modular parsing framework, grammar partitioning and parser composition, applied to the ATIS corpora in English as well as its translated Chinese version. Our ATIS grammars are partitioned manually, with reference to the schema derived from the SQL queries for database access. Each partitioned subgrammar

[11] This performance lies within the range of the natural language results in the ATIS evaluation [Pallet et al. 1994]. We referenced results from several evaluation sites in 1993, where the four evaluation sites had error rates ranging between 6% to 10.5% for the ATIS Class A queries. Our results are not directly comparable with those for 1994, because some evaluation sites have trained on the 1993 test set, in order to test on the 1994 test set. In 1994, the five evaluation sites had error rates ranging between 3.8% to 9.4% for the Class A queries.

operates with its corresponding subparser. We *compose* the outputs of the subparsers together to form an overall parse for an input string. Our experimental results show that grammar partitioning can reduce the overall parsing table size by an order of magnitude, when compared to the use of a single context-free grammar derived from the ATIS training sets. The full parse coverage of the single GLR parser is the same as the parser composition approaches. However, parser composition can produce partial parses and thus attains a higher understanding accuracy. We also compared two strategies of parser composition, cascading and predictive pruning. Cascading applies subparsers at every position in the input string (or lattice) in the order specified by a calling graph of the collection of subgrammars. We used the shortest-path algorithm to find the best path through the multiple subparser outputs, so as to cover the entire input string. Predictive pruning follows left-corner predictive constraints when invoking the various subparsers, and is therefore more computationally efficient than cascading. The additional computation in cascading (compared to predictive pruning) is expended in the production of more partial parses, since cascading allows parse trees to begin and end in all locations of the input sentence. Ongoing work includes the development of a hybrid parser composition approach, embedded in a *multiparser architecture* that can compose different kinds of parsers (GLR parser, Earley parser, etc.) We are also experimenting with automatic methods for partitioning grammars to replace the manual process, as well as incorporate probabilities to rank order alternative parse trees in the output on the WSJ data in Penn Treebank.

ACKNOWLEDGMENT

We would like to thank several anonymous reviewers for their comments and suggestions.

REFERENCES

ABNEY, S. 1991. Parsing by chunks. In *Principle-Based Parsing: Computation and Psycholinguistics*, R. C. Berwick et al., Eds. Kluwer Academic Publishers, 1991.

AHO, A., SETHI, I, R. and ULLMAN, J. 1986. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, Reading, MA: 1986.

AMTRUP, J. 1995. Parallel parsing: Different distribution schemata for charts. In *Proceedings of the 4th International Workshop on Parsing Technologies* (ACL/SIGPARSE, Sept.1995), 12-13.

EARLEY, J. 1968. An efficient context-free parsing algorithm. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, 1968.

HILLIER, F. S. and LIEBERMAN, G. J. 1995. *Introduction to Operations Research*. 6th ed. McGraw-Hill, 1995.

JOHNSON, S. C. 1975. YACC: Yet Another Compiler Compiler. Tech. Rep. CSTR 32, Bell Laboratories, Murray Hill, NJ., 1975.

KITA, K., MORIMOTO, T., and SAGAYAMA, S. 1993. LR parsing with a category reachability test applied to speech recognition. *IEICE Trans. Inf. Syst. E 76-D*, 1 (1993), 23-28.

KITA, K., TAKEZAWA, T., HOSAKA, J., EHARA, T., and MORIMOTO, T. 1990. Continuous speech recogition using ywo-level LR parsing. In *Proceedings of the International Conference on Spoken Language Processing*, 21.3.1, 905-908.

KITA, K., TAKEZAWA, T., and MORIMOTO, T. 1991. Continuous speech recognition using two-level LR parsing. *IEICE Trans. E 74*, 7 (1991), 1806-1810.

KORENJAK, A. 1969. A Practical method for constructing LR(k). *Commun. ACM 12*, 11 (Nov. 1969).

LUK, P. C., MENG, H., and WENG, F. 2000. Grammar partitioning and parser composition for natural language understanding. In *Proceedings of the International Conference on Spoken Language Processing* (Beijing, 2000).

MOORE, R.C. 2000. Improved left-corner chart parsing for large context-free grammars. In *Proceedings of 6th International Workshop on Parsing Technologies* (ACL/SIGPARSE, Feb. 2000).

PALLET ET AL. 1994. The 1993 benchmark tests for the ARPA spoken language program. In *Proceedings of the DARPA Spoken Language Technology Workshop* (1994), 15-40.

PRICE, P. 1990. Evaluation of spoken language systems: The ATIS domain. In *Proceedings of the ARPA Human Language Technology Workshop* (1990), 91-95.

RULAND , T., RUPP, C., SPILKER, J., WEBER, H.. and WORM, K. 1998. Making the most of multiplicity: A multi-parser multi-strategy architecture for the robust processing of spoken language. In *Proceedings of ICSLP* (1998).

SANN, P. 1991. Experiments with GLR and chart parsing. In *Generalized LR Parsing*. Kluwer Academic. 1991, 17-34.

SIU, K.C. and MENG, H. 1999. Semi-automatic acquisition of domain-specific semantic structures. In *Proceedings of EUROSPEECH* (1999).

SIU, K.C. and MENG, H. 2001. Semi-automatic grammar induction for bi-directional English-Chinese machine translation. In *Proceedings of EUROSPEECH* (Sept 2001).

STEEL, S. and ROECk, A. D. 1987. Bi-directional parsing. In *Proceedings of the 1987 AISB Conference* (London, 1987). Wiley, New York, NY.

TOMITA, M. 1985. *Efficient Parsing for Natural Language*. Kluwer Academic, Boston, MA, 1985.

TOMITA, M. 1986. An efficient word lattice parsing algorithm for continuous speech recognition. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing* (ICASSP, April 1986*)*, 1569-1572.

WARD, W. 1990. The CMU Air Travel Information Service: Understanding spontaneous speech. In *Proceedings of Speech and Natural Language Workshop* (June 1990), 127-129.

WENG, F. 1993. Handling syntactic extra-grammaticality. In *Proceedings of the 3rd International Workshop on Parsing Technologies* (ACL/SIGPARSE, Aug. 1993).

WENG, F. and STOLCKE, A. 1995. Partitioning grammar and composing parsers. In *Proceedings of the 4th International Workshop on Parsing Technologies* (ACL/SIGPARSE, Sept. 1995)..

WENG, F., MENG, H., and LUK, P. C. 2000. Parsing a lattice with multiple grammars. In *Proceedings of the 6th International Workshop on Parsing Technologies* (ACL/SIGPARSE, Feb. 2000).

YOUNGEr, D. H. 1967. Recognition and parsing of context free languages in time n^3. *Inf. Control 10* (1967), 189-208.

Received November 2001; revised May 2002; accepted June 2002.