Automatic Grammar Partitioning for Syntactic Parsing

Po Chui Luk*, Fuliang Weng, and Helen Meng***

***Human-Computer Communications Laboratory**

Department of Systems Engineering and Engineering Management

The Chinese University of Hong Kong

****Intel China Research Center**

{pcluk, hmmeng}@se.cuhk.edu.hk, fuliang.weng@intel.com

Full Paper Submission

Abstract

Natural language processing technologies offer ease-of-use of computers for average users, and ease-of-access to on-line information. Natural language, however, is complex, and the traditional methods of parsing with a single grammar and parser may result in an inefficient and large system that is difficult to maintain and is fragile in dealing with language irregularities. This paper begins by reviewing an alternative effort in grammar decomposition (also known as grammar partitioning) for natural language parsing, which aims to alleviate these problems. We then propose a novel automatic approach for grammar partitioning, in comparison with a random method of partitioning. Our experiments show that syntactic GLR parsing is formidable for the Wall Street Journal corpus in the Penn Treebank when a single grammar is used. This is due to too many grammar rules for parsing table generation. However, grammar partitioning solves the problem and offers a viable alternative. Our results also show that our automatic grammar partitioning method based on the mutual information criterion fares better than a random partitioning method and exhibits efficiency in parsing as well as high parse coverage.

1 Introduction

Natural language provides a natural means for common people to interact with computers and access on-line information. Due to the complexity of natural language itself, the grammar that describes the query language can be very complex. As a central task in language processing, the traditional way of using a single parser and single grammar leads to an inefficient, fragile, and often very large processing system. Using the co-occurrence feature in the natural language, grammar rules can be automatically clustered into sub-grammars to achieve modular parsing. Modular parsing has several advantages: (1)

improved trade-off between space and efficiency; (2) improved robustness; (3) improved re-usability of sub-grammars for distributed computing and for easier portability to new application domains.

In the past decade, many studies are related to modular parsing have been reported. Abney [1] proposed a two-level chunking parser, which converts an input sentence into the chunks and then uses an attacher to convert these chunks into a parse tree. Amtrup [3] introduced an approach that distributes grammar rule applications in multiple processors within a chart parser framework. Ruland et al [8] developed a multi-parser multi-strategy architecture for noisy spoken languages.

Our previous work has presented a methodology to partition the Air Travel Information Systems (ATIS) semantic grammar [10] *manually* into multiple sub-grammars, each having its own LR parsing table [2] and Generalized LR (GLR) parser [11]. Our results showed that grammar partitioning helps reduce the overall parsing table size when compared to using a single grammar [7]. In this paper, we propose a *automatic grammar partitioning* method for syntactic parsing based on the Wall Street Journal (WSJ) sentences in the Penn Treebank. Hence we have scaled up from the ATIS corpus to the WSJ corpus using an automatic partitioning method. We use the sub-parsers with partitioned grammars to parse the test data in the WSJ.

This paper is organized into the following sections. Section 2 briefly reviews the definition of grammar partitioning and presents the method of automatic grammar partitioning that uses a mutual information criterion. Section 3 describes how to compose the sub-parsers with partitioned grammars to obtain an output parse. Section 4 presents our experimental results based on the WSJ sentences in the Penn Treebank. The final section concludes and gives our future direction.

2 Grammar Partitioning

2.1 Definitions and Concepts

We first review some definitions and concepts [13] used in the paper for easier readability. We focus on context-free grammars (CFG), since they are the backbones for most natural language processing systems, such as in [5] and [9].

We partition the entire grammar into sub-grammars based on the grammar's production rules. The interaction among different sub-grammars is through non-terminal sets, i.e. INPUT and OUTPUT, and a virtual terminal technique. The virtual terminal is essentially a non-terminal, but acts as if it were a terminal. In the remaining paper, a virtual terminal is prefixed with *vt*. The INPUT of a sub-grammar is

a set of non-terminals that were previously parsed by other sub-grammars. To be more precise, these non-terminals are on the right hand side (RHS) of some rules in this sub-grammar and on the left hand side (LHS) of certain rules in some other sub-grammar(s). The OUTPUT of a sub-grammar are those non-terminals that were parsed based on this sub-grammar and used by other sub-grammars as their INPUT symbols. To be more precise, those non-terminals are on the LHS of some rules in this sub-grammar and are INPUT set members of some other subset(s). In other words, we may view a partitioned subset of production rules of a grammar as a multiple-valued function, it takes non-terminals in the INPUT set as its input, and returns non-terminals in the OUTPUT set as its output. A directed calling graph for the sub-grammar set of G is then defined as (V, E), where V is the sub-grammar set, A and B are sub-grammars in V and E={(A, B)}, with the overlap of the OUTPUT of B and the INPUT of A being nonempty. This grammar partition definition is more flexible than the non-terminal-based grammar partition defined in [6]. In the following sub-section, we will show a real example to describe the INPUT and OUTPUT concepts.

2.2 Automatic Grammar Partitioning Method

Our grammar rules are extracted from the parsed WSJ sentences in the Penn Treebank. A parse tree example is shown in Figure 1. The tree terminals are part-of-speech (POS) tags which are prefixed with #. Grammar rules extracted from the parse tree in Figure 1 are shown in Figure 2. We record the frequency of invocation between rules, for example Rule 3 (PP-DIR → #IN NP) in Figure 2 has called Rule 4 (NP → NP NP-ADV) once.

Based on the discussion of [14], grammar partitioning can be seen as the reverse process of grammar clustering. Our partitioning procedure begins with the set of the finest grammar partition, i.e., each partition contains exactly one grammar rule. For instance, if a grammar partition only contains Rule 3, the INPUT set of Rule 3 is the non-terminal NP and the OUTPUT set is PP-DIR. For training on the parsed sentence in Figure 1, we use a calling matrix in Figure 2 to record the calling frequencies between sub-grammars. We can find the OUTPUT and INPUT of the sub-grammars in the rows and columns of the matrix respectively. For the calling matrix in Figure 2, the entry at row *i* and column *j* is the frequency of sub-grammar *i* calling sub-grammar *j*. We attempt to cluster the sub-grammars to form larger ones based on the co-occurrences of their interaction. The measurement for co-occurrences of the rules is explained in the next sub-section. Our grammar partitions are formed such that rules within the same partition have frequent caller/callee interactions, but such interactions are infrequent

for rules belonging to different grammar partitions. As a initial step, we find sub-grammars with an empty INPUT set (zero row in the calling matrix), duplicate and merge the sub-grammar for each its caller sub-grammars. As mentioned, grammar rules from the parse tree in Figure 1 are listed in Figure 2. We begin by forming a partition for every grammar rule. Grammar partition 9 in Figure 2 is (NP-SBJ → #NNP #NNP #NNP). Its input set is empty, since all grammar symbols in the RHS of the rule are POS terminals. Grammar partition 1 is a caller of partition 9, so we can merge them to form a new grammar partition.

2.3 Calculating Mutual Information

To measure the caller/callee co-occurrences between two sub-grammars, we use the Mutual Information [4].

Suppose G_i and G_j are two sub-grammars in a grammar partition that contains n sub-grammars, the mutual information of these two sub-grammars is defined as follows:

$$MI(G_i, G_j) = \log \frac{P(G_i, G_j)}{P(G_i)P(G_j)} \qquad (1)$$

$P(G_i, G_j)$ is defined as the probability of G_i calling G_j.

$$P(G_i, G_j) = \frac{Freq(G_i, G_j)}{\sum_{i,j=1}^{n} Freq(G_i, G_j)} \qquad (2)$$

where $Freq(G_i, G_j)$ is the actual frequency of G_i calling G_j.

$P(G_i)$ is defined as the probability of G_i being a caller, and $P(G_j)$ is defined as the probability of G_j being a callee.

$$P(G_i) = \frac{\sum_{j=1}^{n} Freq(G_i, G_j)}{\sum_{i,j=1}^{n} Freq(G_i, G_j)} \text{ and } P(G_j) = \frac{\sum_{i=1}^{n} Freq(G_i, G_j)}{\sum_{i,j=1}^{n} Freq(G_i, G_j)} \qquad (3)$$

Then, $MI(G_i, G_j)$ becomes:

$$MI(G_i, G_j) = \log \frac{Freq(G_i, G_j)}{\sum_{j=1}^{n} Freq(G_i, G_j) \sum_{i=1}^{n} Freq(G_i, G_j)} + \log \sum_{i,j=1}^{n} Freq(G_i, G_j) \qquad (4)$$

The second term of the above equation is constant for each iteration (calling matrix remains unchanged), so maximizing $MI(G_i, G_j)$ is equivalent to maximizing the first term. For each merge iteration, we find the maximum value of $MI(G_i, G_j)$ in the calling matrix, and merge the corresponding

sub-grammars pair together to form a larger sub-grammar. The merge operation is iterated until the process reaches the maximum number of iterations allowed[1].

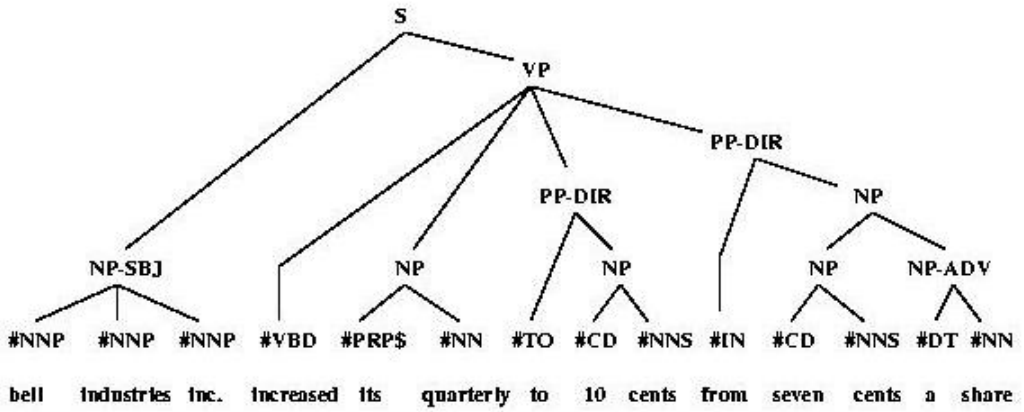


Figure 1. An example parse tree drawn from the WSJ sentences in the Penn Treebank.

1: S → NP-SBJ VP
2: VP → #VBD NP PP-DIR PP-DIR
3: PP-DIR → #IN NP
4: NP → NP NP-ADV
5: NP-ADV → #DT #NN
6: PP-DIR → #TO NP
7: NP → #CD #NNS
8: NP → #PRP$ #NN
9: NP-SBJ → #NNP #NNP #NNP

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Σ**
1	0	1	0	0	0	0	0	0	1	*2*
2	0	0	1	0	0	1	0	1	0	*3*
3	0	0	0	1	0	0	0	0	0	*1*
4	0	0	0	0	1	0	1	0	0	*2*
5	0	0	0	0	0	0	0	0	0	*0*
6	0	0	0	0	0	0	1	0	0	*1*
7	0	0	0	0	0	0	0	0	0	*0*
8	0	0	0	0	0	0	0	0	0	*0*
9	0	0	0	0	0	0	0	0	0	*0*
Σ	*0*	*1*	*1*	*1*	*1*	*1*	*2*	*1*	*1*	*9*

Figure 2. Grammar rules extracted from the parse tree in Figure 1 and the calling matrix of the grammar.

$MI(G_1, G_2)$	$=log[1/(2\times1)]+log(9)$
$MI(G_1, G_9)$	$=log[1/(2\times1)]+log(9)$
$MI(G_2, G_3)$	$=log[1/(3\times1)]+log(9)$
$MI(G_2, G_6)$	$=log[1/(3\times1)]+log(9)$
$MI(G_2, G_8)$	$=log[1/(3\times1)]+log(9)$
$\boldsymbol{MI(G_3, G_4)}$	$\boldsymbol{=log[1/(1\times1)]+log(9)}$
$MI(G_4, G_5)$	$=log[1/(2\times1)]+log(9)$
$MI(G_4, G_7)$	$=log[1/(2\times2)]+log(9)$
$MI(G_6, G_7)$	$=log[1/(1\times2)]+log(9)$

Figure 3. Computed mutual information according to the calling graph in Figure 2 and $MI(G_3,G_4)$ obtained the highest value.

If our training data set contains only a single sentence as shown in Figure 1, grammar partition 3 (Rule 3: PP-DIR → #IN NP) will merge with grammar partition 4 (Rule 4: NP → NP NP-ADV), because they

[1] The maximum number of iterations is chosen to be 2,000, as described in section 4.

have the highest Mutual Information, $MI(G_3, G_4)$ in the matrix, as shown in Figure 3. The newly merged sub-grammar will contain Rules 3 and 4, its INPUT set has the non-terminals NP-ADV and NP, and the OUTPUT set contains PP-DIR.

After automatic grammar partition using the mutual information criterion, we are left with some small partitions. These are then merged if they share the same OUTPUT set.

For each merge operation, we also limit the grammar size within a bound so to prevent the case when many grammar rules cluster into one large grammar. Grammar size of grammar G is measured according to the equation below:

$$|G| = \sum_{(A \to \alpha) \in P} length(A\alpha) \qquad (5)$$

where P is the set of production rules, $\{A \to \alpha\}$ is one of the rule in P and $length(x)$ is the length of string.

3 Composing Sub-parsers for Multiple Grammars

Each sub-grammar has its own LR(1) parsing table and GLR parser. We compose all sub-parsers in order to obtain an overall parse of the input sentences. A lattice with multiple granularities (LMG) serves as an interface to record the INPUTs and OUTPUTs of all sub-parsers. An LMG is a directed acyclic graph whose nodes indicate spatial positions and whose edges are either terminals or virtual terminals, the non-terminals outputted by any sub-grammar. We modify our GLR parsing algorithm in a way similar to that proposed by Tomita [12] in order to handle an input lattice and LMG. The algorithm is allowed to start and end within LMG so that multiple sub-parsers can operate on it [14].

The parser composition method used here is a bottom-up parsing algorithm. The input sentence is first converted to an LMG. The invocation of sub-parsers is from right to left. During parsing, newly created virtual terminals are added dynamically to the LMG. Figure 4 shows how a sub-parser parses a prepositional phrase. The parse trees are directly derived from the edges of the LMG.

Syntactic parsing becomes demanding in terms of computational time and memory if we invoke all sub-parsers in all edges in the LMG. To improve on efficiency, we count the frequencies of terminals and virtual terminals that appear in the leftmost of the RHS of the rules according to the training data for each grammar cluster, i.e., the frequencies of the left corners. Given an incoming input edge of an LMG, we use this information to rank the priority of invocation of sub-parsers in an

N-best[2] list. The sub-parsers that are in the N-best list are invoked, and all the rest sub-parsers are discarded. However, each sub-parser could end at any edge in the LMG. To avoid creating too many new edges on the LMG for each parser invocation, we only select one resultant parse tree as an output parse if the parser returns successfully.

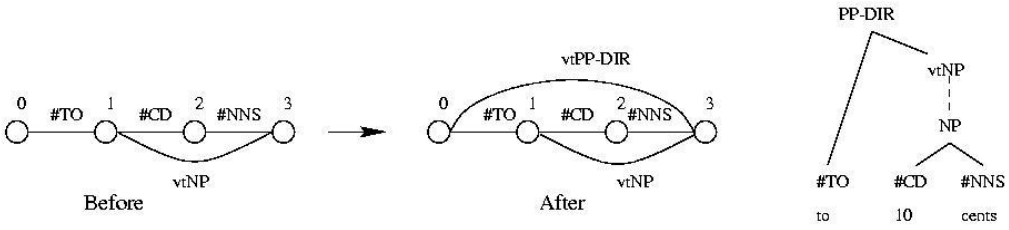


Figure 4. The LMG before and after invocation of a sub-parser at TO and its parse tree with vtPP-DIR as the output virtual terminal.

4 Experimental Setup

To measure the effectiveness of our automatic partitioning approach, we compare its GLR parser sizes with a random grammar partition of similar number of sub-grammars and similar sub-grammar sizes. The single non-partitioned grammar is too large for compilation into a monolithic GLR parser, and therefore is not included in the comparison.

4.1 Automatic Grammar Partitioning by the Mutual Information Criterion

Automatic grammar clustering was trained on sections 02 to 21 of the Wall Street Journal of Penn Treebank, which contains 41,603 sentences. There are altogether 20,696 rules extracted from the training sentences. We set the maximum sub-grammar size be 1,000. There are two parameters for iterative merge operation. We set minimum calling frequency of two proposed merge sub-grammars be 4 in order to avoid sparse data. We also set the maximum number of iterations be 2,000.[3] As a result, we obtained 357 sub-grammars.

4.2 Random Grammar Partitioning without any Heuristics

As a control experiment, we randomly cluster the extracted grammar rules into clusters without any heuristics. In order to obtain comparable number of grammar clusters as the one using mutual information criterion, we partition the 20,696 rules into 59 production rules per cluster. All rules in all

[2] N is chosen to be 3, as described in section 4.

[3] If we set the maximum sub-grammar size be 500 and minimum calling frequency be 4, the iterative merge operation will end at around 2,000 iterations (we could not merge any two sub-grammars further). To control the processing time for merge operations, the maximum number of iterations

the clusters are unique. We cluster the first 59 rules as one cluster, and the next 59 rules as another cluster, so far and so forth. As a result, we can obtain 351 partitioned sub-grammars.

4.3 Parsing Table Size

We compare our automatic grammar partitioning method with the random clustering. We used the same LR(1) parsing table generator to construct LR(1) tables for these two set of sub-grammars. The total number of states (rows) in the parsing table for the partitioned grammars is the sum of the number of states in all sub-parsers. Table 1 shows the total number of states in the parsing tables for these two sets of sub-grammars. Figure 5 and Figure 6 show the distribution of the grammar cluster in number of states and the number of states/rule respectively. From these two figures, we find that some automatic partitioned grammars are much smaller in terms of parsing table states. The result shows we can save the overall parsing table sizes if using MI for grammar partitioning. This reduces the memory to store the parsing tables and time to access the table during parsing. The smaller numbers of states and the ratios also imply good determinism of the sub-parsers.

	MI	**Random**
Number of grammar clusters	357	351
Total states in parsing table	231,907	469,484

Table 1. Partitioned grammar statistics for two different clustering methods. One is based on mutual information (MI) and the other one is randomly clustered without any heuristics (random).

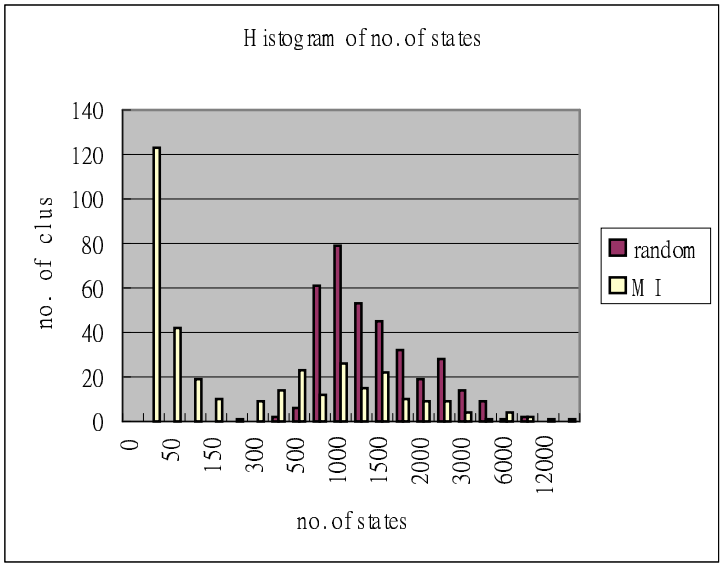
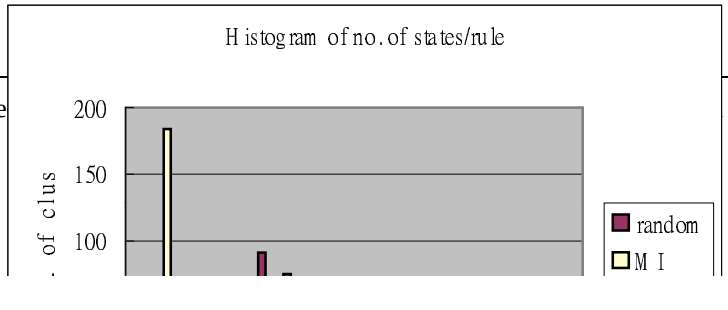


Figure 5. The histogram of number of states in the partitioned grammar set.



allowed is cappe at 1,000.

Figure 6. The histogram of number of states per rule in the partitioned grammar set.

4.4 Parsing Results

We use our parsing framework with multiple sub-grammars to parse the WSJ sentences in Penn Treebank's section 23. For the N-best list, we choose N being 3 so to select the top 3 ranked sub-parsers for invocation. The input sentences are POS tags provided in the corpus. The parsing output of each input sentence is an LMG. Since there may be multiple paths in the LMG, a Viterbi algorithm is used to find the shortest path in the LMG to represent the sentence. Table 2 shows the parsing results of the WSJ sentences in the test set. Parse coverage is the percentage of sentence that can be fully parsed (there is a parse tree covering the whole input query). The experiment was performed on the machine with configuration of Intel Pentium III 1GHz CPU, 512 Mbytes memory. The high parse coverage implies that the partitioned grammar can lead to the recognizable language similar to that from the entire un-partitioned grammar.

	WSJ in section 23
No. of sentences	2403
Maximum sentence length	50
Average sentence length	20.4 POS
Parse coverage	92.1%
Parsing speed	4.3seconds/sentence

Table 2. The parsing results of the WSJ test data in the Penn Treebank.

5. Conclusions and Future Work

In this paper, we have presented our automatic grammar partitioning algorithm and experimented with the Wall Street Journal sentences in the Penn Treebank. This effort scales up our previous work based on the Air Travel Information Service (ATIS) corpus. Grammar rules are derived from the parse trees provided in sections 02 to 21 of the Penn Treebank. Our novel automatic grammar partitioning algorithm is one way of grammar clustering. Grammar rules or sub-grammars are clustered together to form a larger partition if they have frequent caller/callee interactions. Clustering is based on the mutual information criterion which measures the caller/callee co-occurrences. Our experimental results show that this automatic grammar partitioning approach can reduce parsing table size, when compared with a random decomposition into a comparable number of grammar clusters. Our partitioned grammars can also be used for Generalized LR parsers to parse section 23 of the Penn Treebank that serves as an open test set. In order to save the computational cost during parsing, we constrained the invocation of sub-parsers and also output edges creation on the LMG. Our results are encouraging on two counts: (1) the parsing procedure was formidable had we used a single unpartitioned grammar, because there were too many grammar rules for the GLR parser; and (2) we achieved a parse coverage of 92% for the test set, which indicates that the automatically partitioned grammar captures the similar recognizable language as the entire original unpartitioned grammar. In addition, it also suggests that our parsing framework is generalizable to unseen test data. In the near future, we plan to further incorporate lexical information into our parsing framework, and to use statistical parsing to select the best path through the LMG to give the output parse tree.

References

[1] Abney, S., Parsing by Chunks, ln Principle-Based Parsing: Computation and Psycholinguistics, R. C. Berwick et al. (eds), Kluwer Academic Publishers, 1991.

[2] Aho, A., Sethi, R. and Ullman, J. Compiliers: Principles, Techniques, and Tools. Reading, MA: Addison-Wesley, 1986.

[3] Amtrup, J., “Parallel Parsing: Different Distribution Schemata for Charts”, In Proceedings of the 4th International Workshop on Parsing Technologies, Prague, Sep 1995.

[4] Church, K. and Hanks, P., “Word Association Norms, Mutual Information, and Lexicography”. In Computational Linguistics, Vol. 16, No. 1., pages 22-29, 1990.

[5] Goddeau, D., “Use Probabilistic Shift-reduce Parsing in Speech Recognition System”. In Proceedings of International Conference on Spoken Language Processing, pages 321-324, October 1992.

[6] Korenjak, A., “A Practical Method for Constructing LR(k)”, CACM 12, 11, 1969.

[7] Luk, P.C., Meng, H. and Weng, F., “Grammar Partitioning and Parser Composition for Natural Language Understanding”, In Proceedings of the International Conference on Spoken Language Processing, Beijing, China. October 2000.

[8] Ruland, T., Rupp, C., Spilker, J., Weber, H., and Worm, K., “Making the Most of Multiplicity: A Multi-Parser Multi-Strategy Architecture for the Robust Processing of Spoken languages”, In Proceedings of International Conference on Spoken Language Processing, 1998.

[9] Seneff, S., “Tina: A Natural Language System for Spoken Language Applications”. Computational Linguistics, 18(1): 61-86, 1991.

[10] Siu, K.C. and Meng, H., “Semi-automatic Acquisition of Domain-specific Semantic Structures”, In Proceedings of Eurospeech, 1999.

[11] Tomita, M. Efficient Parsing for Natural Language, Kluwer Academic Publishers, MA, 1985.

[12] Tomita, M. “An Efficient Word Lattice Parsing Algorithm for Continuous Speech Recognition”, In Proceedings of ICASSP, 1986.

[13] Weng, F.L. and Stolcke, A. “Partitioning Grammars and Composing Parsers”, In Proceedings of 4th International Workshop on Parsing Technologies, 1995. For the full paper, see web page http://www.speech.sri.com/people/fuliang/publications.html.

[14] Weng, F.L., Meng, H. and Luk, P.C., “Parsing a Lattice with Multiple Grammars”, In Proceedings of 6th International Workshop on Parsing Technologies, February 2000.