Multi-Parser Architecture for Query Processing

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

*Intel China Research Center, Intel Corp, Beijing, China
**Human-Computer Communications Laboratory
The Chinese University of Hong Kong, Hong Kong SAR, China.
{kui.xu, fuliang.weng}@intel.com; {hmmeng, pcluk}@se.cuhk.edu.hk

Abstract

Natural language queries provide a natural means for common people to interact with computers and access to on-line information. Due to the complexity of natural language, the traditional way of using a single grammar for a single language parser leads to an inefficient, fragile, and often very big language processing system. Multi-Parser Architecture (MPA) intends to alleviate these problems, and the modularized MPA also has the advantage of easier portability to new domains and distributed computing. In this paper, we investigate the effect of using different types of parsers on different types of query data in MPA. Three data sets and two types of sub-parsers, particularly a predictive cascading composition for pre-compiled Earley parsers[1], have been examined. Results show that partitioning grammars leads to superior speed performance for the Earley-style parser across the three data sets. GLR parser is faster than Earley parser in the partitioned case, but it can lead to an excessive memory usage for the un-partitioned case.

1. Introduction

Natural language queries, sometimes as the back-end to speech recognizers, provide a natural means for common people to interact with computers and access to on-line information. Due to the complexity of natural language itself, the grammar that describes the query language can be very complex. The traditional way of using a single grammar leads to an inefficient, fragile, and often very big processing system. These problems become more apparent, with the increasing demand for natural language applications over the inter-net.

Various methods to deal with these parsing issues have been studied, e.g., [1],[2],[10],[5]. The proposed MPA by [10],[11],[4]intends to alleviate these problems simultaneously by partitioning a single grammar into multiple sub-grammars and composing sub-parsers for the corresponding sub-grammars. The MPA is highly modularized, and has the advantage of re-usable sub-grammars, which is conducive towards distributed computing as well as portability to new application domains. Our current work presents several enhancements: (i) a methodology for automatic grammar partitioning; (ii) an improved parser composition method known as predictive cascading; and (iii) the use of an Earley parser in addition to the GLR parser used previously. Experiments were conducted with three data sets – a keyword list, a semantic grammar and a syntactic grammar. Results show that grammar partitioning with composition of Earley parsers can speed up processing substantially for both semantic and syntactic grammars, compared to the un-partitioned grammars. Furthermore, while the GLR parser is faster than Earley; the latter utilizes memory more efficiently.

[1] From hereon all the occurrences of Earley parsers refers to the pre-compiled Earley parser.

2. The Multi-Parser Architecture

MPA involves the two main processes of grammar partitioning and parser composition. Grammar partitioning divides a grammar into multiple sub-grammars. Each sub-grammar is used by its corresponding sub-parser, and their parsing results are composed to produce an overall parsing output for a natural language query. The interaction among sub-grammars/sub-parsers is achieved by using a virtual terminal technique. The virtual terminal is essentially a non-terminal, but acts as if it were a terminal. The INPUT set to a sub-grammar is a set of virtual terminals that were previously parsed by other sub-grammars. The OUTPUT set of a sub-grammar is a set of non-terminals that are parsed based on this sub-grammar; and used by other sub-grammars as their INPUT sets. Hence a partition (subset) of production rules can be viewed as a multi-valued function – it takes the virtual terminals in the INPUT set as input, and returns a set of non-terminals in the OUTPUT set as output. Results on manually partitioned grammars were presented previously [11],[4]. In this paper, we will present a method and results for automatic grammar partitioning.

Two methods for parser composition were also presented: composition by cascading and composition by predictive pruning [11],[4]. Cascading is a bottom-up parsing procedure starting from the terminal level to and moves to the SENTENCE level. It begins by converting the input sentence into a lattice called LMG, and invokes parsers at each lattice position. During parsing, newly created virtual terminals (vt) are added dynamically to the stack and LMG. A stack stores the terminals and virtual terminals according to the topological order of the LMG. Cascading has the advantage of parser robustness, but is relatively slow due to excessive parser invocation at each lattice (LMG) position. To avoid this, predictive composition is a top-down procedure where the caller sub-parser invokes a callee sub-parser only if the latter satisfy the constraint that the input node must be its left corner. The ones that do not satisfy the constraint are pruned. While the top-down approach has the advantage of execution efficiency, it may at times be too constrained to be robust. In subsection 2.2, we present our improved parser composition method known as predictive cascading.

2.1. Automatic Grammar Partitioning

To see the parser effect on automatically partitioned grammar, we use a syntactic grammar derived from the ATIS subset in the Penn Treebank, which is different from the semi-automatically derived semantic grammar [6] used in our previous work [11],[4].

The Penn Treebank contains a subset of the ATIS-3 corpus[2], which is a set of 577 parsed queries including Class A (self-contained) and Class D (dependent on discourse context) queries. Parse trees of these queries are provided, with the tree terminals being part-of-speech (POS) tags[3]. A parse tree example is shown in Figure 1. Our grammar rules are extracted from the parse trees, and for simplicity we ignore the null elements XXX as well as co-indexing in the grammar rules (hence WHNP-1 is the same as WHNP, see Figure 2). As we extract the grammar rules from the training parse trees, we also record the frequency of invocation between rules, e.g. Rule 2 in Figure 2 (PP-DIR→IN NP) has called Rule 1 (NP→NNP) once. Here the INPUT set of Rule 2 is the non-terminal NP and the OUTPUT set is PP-DIR.

Our automatic grammar partitioning procedure begins with the set of finest grammar partition, each of which contains exactly one grammar rule. Then we attempt to cluster the sub-grammars to form larger ones based on the frequency of their interaction. In other words, we want to cluster grammar partitions that have frequent caller/callee interactions into a larger sub-grammar. The procedure references a calling matrix, where the entry at row i and column j is the frequency of sub-grammar i calling sub-grammar j (see Figure 3).

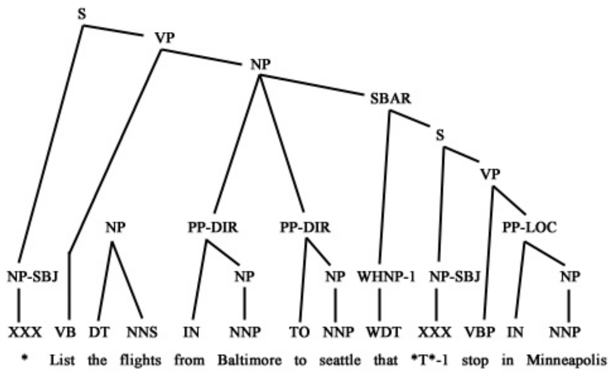

Figure 1: An example parse tree drawn from an ATIS sentence from the Penn Treebank.

There are three steps in the clustering procedure:

- 1. The initial step checks for sub-grammars with an empty input set, duplicate and merge the sub-grammar for each of its caller sub-grammars. For example, grammar partition 4 in Figure 2 is (WHNP→WDT). Its INPUT set is empty (since WDT is a POS terminal), and its OUTPUT set is {WHNP}. For grammar partition 8, the INPUT set is {WHNP S} and the OUTPUT set is {SBAR}. Hence partition 8 is a caller of 4, and we merge them to form:

  Grammar partition 11:

  NPUT= {S}; OUTPUT={SBAR};

  SBAR→ WHNP S; WHNP→WDT

  We then update the calling matrix as shown in Figure 4. Columns 4 and 8 are deleted and column 11 is added. The other entries remain unchanged.

0: NP → DT NNS
1: NP → NNP
2: PP-DIR → IN NP
3: PP-DIR → TO NP
4: WHNP → WDT
5: PP-LOC →IN NP
6: VP →VBP PP-LOC
7: S →VP
8: SBAR →WHNP S
9: NP →NP PP-DIR PP-DIR SBAR
10: VP →VB NP

Figure 2: Grammar rules extracted from the parse tree in Figure 1.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0
7	0	0	0	0	0	0	1	0	0	0	1
8	0	0	0	0	1	0	0	1	0	0	0
9	1	0	1	1	0	0	1	0	1	0	0
10	0	0	0	0	0	0	0	0	0	1	0

Figure 3: Calling matrix of the grammar in Figure 2.

	0	**1**	**2**	**3**	**5**	**6**	**7**	**9**	**10**	**11**
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0
5	0	1	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	0	0
7	0	0	0	0	0	1	0	0	1	0
9	1	0	1	1	0	0	0	0	0	1
10	0	0	0	0	0	0	0	1	0	0
11	0	0	0	0	0	0	1	0	0	0

Figure 4: Updated calling matrix of the grammar in Figure 2.

- 2. The next step in our clustering procedure is an iterative step. We find the maximum value in the calling matrix, and merge the two sub-grammars into a new grammar. Merging is conditioned upon two thresholds: (i) the frequency in the matrix must be greater than the minimum count of X (set at 3) ; and (ii) the size of the merged grammar must lie below the threshold Y (set at 100) to avoid infinite growth. Grammar size is measured according to Equation (1): where P is the set of production rules, and length(x) is the length of string x.

[2] The Penn Treebank and ATIS corpora are available from the Linguistic Data Consortium www.ldc.upenn.edu.
[3] The set includes 32 POS tags in total.

$$|G| = \sum_{(A \to \alpha) \in P} length(A\alpha) .....Eqn(1)$$

- 3. The final step in our clustering procedure is to merge two sub-grammars if the OUTPUT set of one contains all the non-terminals in the OUTPUT set of the other.

As such we obtained 46 grammar partitions based on the syntactic rules derived from the 577 Penn Treebank ATIS training parse trees.

2.2. Parser Composition by Predictive Cascading

Predictive cascading is a parser composition method that combines the merits of predictive pruning and cascading. In this process, all edges of the lattice are topologically sorted into a stack, just as simple cascading, but in reversed topological order. The LMG will then be parsed from the last edge to the first one. When one sub-parser parses successfully, an output virtual terminal (a new edge) will be placed on the LMG and be added to the stack. This edge will be later tested by a predictive procedure that determines whether an edge is the left corner of a sub-grammar (or a callee sub-parser) or not. In other words, if an edge is a left corner of a sub-grammar, the edge is the first terminal of some derived string from the sub-grammar. This means, the corresponding sub-parser could be invoked for further parsing. Otherwise, the sub-grammar is not invoked.

3. Experiments

For experiments, two different styles of parsers has been used. One is Earley parser, where the prediction step is pre-compiled. The other parser is GLR parser [6],[7]. These two parsers have their own different characteristics. Earley parser does not encode all the prefix paths in a table, while GLR parser tries to do it as much as possible. This usually makes GLR faster but bigger, compared with Earley parser.

Three different data sets are used for our experiment. The first one is a Chinese keywords list grammar used for key words extraction; the second one is semi-automatically constructed ATIS semantic grammar [4]; the third one is the fully annotated Penn Tree Bank ATIS grammar. All the grammar has two versions, one is partitioned, and the other is un-partitioned.

Two sets of experiments are conducted. One is to compare the speed between GLR parser and Earley parser on partitioned grammars. The other is to compare the speed between partitioned approach and un-partitioned approach for different grammar sets. Due to excessive GLR parsing table size in Penn Treebank ATIS grammar, we only use Earley parser for the effect on speed. Both experiments use Intel Pentium Ⅲ 933MHz, 512 Mbytes memory and Windows 2000 Professional.

The three grammars and their partitioning methods are explained below.

The Chinese keywords list is obtained from an international news corpus, which has 15426 keywords (rules). For partitioning, all the keywords were first sorted in alphabetical order. Then, the list of keywords is split into 78 sub-grammars with 200 keywords each. The top non-terminals are named as $KEYWORD_i$, where i is the index of the sub-grammar. Here are some examples of the keyword rules.

$KEYWORD_0 \rightarrow$ 巴 拉 克
$KEYWORD_0 \rightarrow$ 巴 勒 斯 坦
$KEYWORD_0 \rightarrow$ 半 岛
$KEYWORD_0 \rightarrow$ 办 法
$KEYWORD_0 \rightarrow$ 包 括
$KEYWORD_0 \rightarrow$ 饱 经 风 霜
$KEYWORD_0 \rightarrow$ 饱 受

The ATIS semantic grammar is a set of context-free rules, but contains both semantic and syntactic structures. The low level grammar rules are mainly semantic concepts of ATIS type, such as CITY-NAME, CLASS-TYPE, MONTH-NUMBER, etc. They are obtained by a semi-automatic grammar induction algorithm [4]. In this experiment, we partitioned the 1297 semantic rules into 64 sub-grammars. Examples of English ATIS-3 rules are given below.

S → ASK FLIGHT_NP|...
ASK → show me | list | tell me | give me | ...
FLIGHT_NP → FLIGHT FLIGHT_PP
FLIGHT → flight | flights | flight number | ...
FLIGHT_PP → DEPARTURE | ARRIVAL | ...
DEPARTURE → leaving CITY_NAME | ...
CITY NAME → phoenix | new york | seattle

The automatically partitioned syntactic grammars, are derived from the hand-bracketed Penn Tree bank, ATIS subset (577 sentences including class-A and class-D). We use sentences, which are extracted from the Penn TreeBank LDC, POS input file as our input for parser. In this experiment, we partitioned the 416 rules into 46 sub-grammars. Examples of partitioned grammar are listed below.

S → VP
VP → VB NP NP
NP → DT NNP NNP NNP NNS
NP → RB DT NNP NNP NNS
NP → DT NNP NNP NNS
NP → DT JJS JJ NN NNS
NP → JJ NNS
NP → NNP

For the keyword grammar experiment, test sentences are hand-crafted queries. 200 Chinese queries are used as the test sentences. Average sentence length is 10.6 characters. Max length is 19 characters. In the semantic grammar case, ATIS 1993 test data is used. 1564 ATIS training set has been used as the test sentences. Average sentences length is 11.2 words. Max length is 46 words. For the Penn Treebank ATIS case, most of the ATIS sentences, i.e. 500 out of 577 POS tags sentences, are used as the test sentences. The average sentences length is 6.5 tags, and the max length is 13 tags. The full parse coverage is 99.7% in both partitioned grammar and no partitioned grammar.

The first set of the experiments is to compare their speed for the partitioned grammar case. The difference is shown in table 1 below. In this experiment, all the grammars are partitioned and tested in both pre-compile earley parser and GLR parser. Results have shown that GLR parser runs faster in most cases, especially in more complicated grammars.

	Earley # of sentences per second	GLR # of sentences per second
Keyword list	40.00	25.0
ATIS semantic grammar	42.30	55.9
ATIS Penn Tree Bank grammar	1.99	9.8

Table 1: Speed comparison between Earley parser vs. GLR parser.

The second set of experiments compares the speed between partitioned and no partitioned grammars. Results have shown that Earley parser runs faster for the partitioned case than for the un-partitioned case (Table 2).

	No partitioning # of sentences per second	Partitioning # of sentences per second
Keyword list	0.016	40.00
ATIS semantic grammar	0.29	42.30
ATIS Penn Tree Bank grammar	0.22	1.99

Table 2: Comparison of Earley parser's speed for partitioned vs. un-partitioned grammars.

A third experiment was conducted to see the effect of left-corner prediction for Earley parser. The result has shown that using prediction in cascading gives 40 times of speed up, as compared with the previous cascading composition.

4. Comparison and Conclusion

This paper systematically examines the speed effect using three different grammar sets, from a simple keyword list to a complicated syntactic grammar, with two different styles of parsers. The experimental results have shown that in general, the GLR parser is faster than the Earley parser. In case of Earley parser, the partitioned case is much faster than the un-partitioned one. The results are consistent for simple grammars and complicated grammars, and for manually partitioned grammars as well as automatically partitioned grammar.

[1] proposed two-level chunking parser, which first converts an input sentence into chunking sequence for lower level processing, then uses an attacher to connect these chunks together for higher level processing. But, no systematic speed result has been reported for the parser. [5] developed Multi-Parser Multi-Strategy Architecture for noisy input. It uses a full parser first, and then a partial parser when the full parser does not return a result. CMU's Janus parser [14] uses a similar strategy. However, these multiple parsers are used in an ordered fashion. That is, only when the more restricted parser fails the more robust parser gets started. Therefore, there is no partitioning in the architecture. [12, 13] systematically examined the effect of different left corner constraints on speed. Our results have shown a more significant improvement when using grammar partitioning. We attribute the difference to the block effect, i.e., when a sub-grammar does not meet a left corner constraint, the whole sub-grammar is pruned.

5. References

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

[2] Amtrup, J. Parallel Parsing: Different Distribution Schemata for Charts. Proceesings of the 4th International Workshorp on Parsing Technologies, Prague, pp.12-13, Sep 1995.

[3] Korenjak, A., "A Practical Method for Constructing LR(k)", CACM 12, 11, 1969.

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

[5] Ruland, T., et al "Making the Most of Multiplicity: A Multi-Parser Multi-Strategy Architecture for the Robust Processing of Spoken languages". Proceedings of ICSLP, 1998.

[6] Siu, K.C. and Meng, H. "Semi-automatic acquisition of domain-specific semantic structures. In proc. of Eurospeech, 1999.

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

[8] Tomita, M. "An Efficient Word Lattice Parsing Algorithm for Continuous Speech Recognition" Proceedings of ICASSP, 1986.

[9] Weng, F.L. "Handling Syntactic Extra-Grammaticality". Proceedings of 3rd International Workshop on Parsing Technologies, 1993.

[10] Weng, F.L. and Stolcke, A. "Partitioning Grammar and Composing Parsers". Proceedings of 4-th International Workshop on Parsing Technologies, 1995.

[11] Weng, F.L., Meng, H. and Luk, P.C., "Parsing a Lattice with Multiple Grammars". Proceedings of 6-th International Workshop on Parsing Technologies, 2000.

[12] Robert C. Moore, "Improved Left-corner Chart Parsing for Large Context-free Grammars". Proceedings of 6-th International Workshop on Parsing Technologies, 2000.

[13] John Dowding, Robert Moore, Francois Andry, and Douglas Moran. Interleaving syntax and semantics in an efficient bottom-up parser. In Proc. of the 32 nd Annual Meeting of the Association for Computational Linguistics, pages 110--116, Las Cruces, NM, June27--30 1994

[14] M. Woszczyna, N. Aoki-Waibel, F. D. Buo, N. Coccaro, T. Horiguchi, K. amd Kemp, A. Lavie, A. McNair, T. Polzin, I. Rogina, C. P. Rose, T. Schultz, B. Suhm, M. Tomita, and A. Waibel. JANUS-93: Towards Spontaneous Speech Translation. In Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'94), 1994.