Evaluation of LTAG Parsing with Supertag Compaction

Olga Shaumyan, John Carroll and David Weir
School of Cognitive and Computing Sciences
University of Sussex
Brighton, BN1 9HQ
UK
{olgas,johnca,davidw}@cogs.susx.ac.uk

1. Introduction

One of the biggest concerns that has been raised over the feasibility of using large-scale LTAGs in NLP is the amount of redundancy within a grammar's elementary tree set. This has led to various proposals on how best to represent grammars in a way that makes them compact and easily maintained (Vijay-Shanker and Schabes, 1992; Becker, 1993; Becker, 1994; Evans, Gazdar and Weir, 1995; Candito, 1996). Unfortunately, while this work can help to make the *storage* of grammars more efficient, it does nothing to prevent the problem reappearing when the grammar is processed by a parser and the complete set of trees is reproduced. In this paper we are concerned with an approach that addresses this problem of *computational* redundancy in the trees, and evaluate its effectiveness.

2. LTAG parsing

LTAG parsing involves (at least) the following two steps. Each word in the input sentence is associated with that set of (elementary) trees (also called supertags) from the grammar that it can anchor. In large-scale grammars such as the XTAG grammar (XTAG-Group, 1999) and the LEXSYS grammar (Carroll *et al.*, 1998b), due to lexical ambiguity, there are usually a great many trees that can anchor each word. Once all of these elementary trees or supertags have been found, the parser must explore ways in which they can be composed—using substitution and adjunction—to produce complete parses of the input string.

In various experiments using a large, automatically produced LTAG, Sarkar, Xia and Joshi (2000) measured the time to derive a set of shared derivation forests representing all derivations for each sentence. They used a grammar with 6,789 tree templates and 2,250 sentences of length 21 words or less, and concluded that the amount of syntactic lexical ambiguity and the number of clauses in a sentence are more significant factors in determining the time taken to compute a parse forest than sentence length.

To date, the most popular way of addressing the computational problem of lexical ambiguity in LTAG parsing involves supertag filtering, where another step is included in the parsing processes (between the two phases described above) which involves filtering out some of the possible supertags for words in the sentence (Joshi and Bangalore, 1994; Bangalore, 1997a; Bangalore, 1997b; Chen, Bangalore and Vijay-Shanker, 1999). This can dramatically reduce the time it takes to find all ways in which supertags can be combined together into complete parses. Sarkar et al. demonstrate the potential benefit: parsing their 2,250 sentences with all supertags took 548,000 seconds, but this reduced to 21,000 seconds when the maximum number of supertags per word was limited to 60, and to a mere 31.2 seconds when lexical ambiguity was completely eliminated.

However, a drawback of this approach is that, since supertag filtering cannot be 100% accurate, a proportion of desirable supertags are filtered out, resulting in some parses being lost. For example in Sarkar et al.'s experiment, a limit of 60 supertags per word resulted in over 40% of the sentences receiving no parse at all.

3. Elementary computation sharing

There is an alternative approach to the problem of lexical ambiguity in parsing that removes some of the computational redundancy that results from lexical ambiguity[1]. Given some parsing algorithm, each elementary

* We are extremely grateful to Fei Xia for providing us with the grammar used in these experiments, and to Fei Xia and Anoop Sarkar for the help and advice they have given.

1. Note that the approach described in this section could be combined with supertag filtering.

 Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6), pp. 1–4. Universitá di Venezia.

tree can be viewed as invoking some fragment of computation (an elementary computation). Evans and Weir (1998) showed that elementary computations corresponding to bottom-up parsing can be expressed as finite state automata (FSA). All elementary computations for the supertags associated with a word can be combined into a single FSA. By minimizing this automaton (using standard minimization algorithms) sharing of elementary computation is achieved. The hope is that this will lead to significant reductions in parsing time.

To date, this proposal has only received limited evaluation. Carroll *et al.* (1998a) demonstrated that for a large hand-crafted grammar the number of states was significantly reduced by merging and minimizing the FSA associated with a word. For example, the numbers of states in the automaton for the word *come* (associated with 133 supertags) was reduced from 898 to 50, for *break* from 1240 to 68, and *give* from 2494 to 83.

This paper improves on this evaluation in two ways: firstly, the grammar used is automatically acquired, so we are not open to the charge that it was designed to make this technique work particularly well; secondly, we measure parse time, not just numbers of states for individual words. Even when the number of states is significantly reduced it is not clear that parse time (as opposed to recognition time) will drop. This is because in order that parse trees be recoverable from the parse table, a considerable amount of book-keeping is required when the table is being completed. This increases both space and time requirements.

4. Experimental evaluation

We used a grammar that was automatically induced by Fei Xia (1999) from sections 00–24 of the Wall Street Journal Penn Treebank II corpus (Marcus, Santorini and Marcinkiewicz, 1993). This is very similar to the grammar used by Sarkar, Xia and Joshi (2000) and Sarkar (2000), though slightly larger, containing around around 7,500 elementary trees.

We implemented the algorithm described by Evans and Weir (1997) and Evans and Weir (1998), the details of which are not repeated here. Prior to parsing, the grammar is precompiled as follows. For each word, the set of trees that it can anchor is determined. This results in a total of 11,035 distinct tree sets. For each of these tree sets we first build what we refer to as an unmerged FSA. This automaton contains a separate progression of transitions for each of the trees in the set; using these automata for parsing gives a conventional LTAG parsing algorithm which we used to give a baseline for our evaluation. To evaluate the approach of Evans and Weir we implemented a parser that used minimized versions of the automaton with sharing of common elementary computation fragments.

There are 80,538 non-minimized automata (involving 488,421 states). Thus there is a total of 80,538 occurrences of one of the grammar's elementary trees in the 11,035 tree sets. When these nonminimized automata are minimized we have one automaton for each of the 11,035 tree sets; these automata contain a total of 153,022 states. Thus, minimization gives an overall compaction of a factor of 3.19. In order to determine the computational benefit of elementary computation sharing we ran both the merged and unmerged parsers on a set of 14,272 test sentences of lengths 1–45 words taken from sections 00–24 of the Penn Treebank corpus. The results are shown in Table 1[2]. It is clear that numbers of items and CPU time are smaller for the merged parser, and that the savings increase with longer sentences.

Given a tokenized sentence to be parsed, the time shown includes time to make a chart corresponding to the sentence length, look up definitions for each word and seed the chart with them, and then fill the chart. The results show that the parse time for the merged parser is around 0.6 that of the umerged parser, and that this ratio is fairly consistent as the length of sentence increases. This is shown in Figure 1.

5. Discussion

We have presented an empirical evaluation of the automaton-based LTAG parsing algorithm presented by Evans and Weir (1998). We used a grammar automatically generated from Penn Treebank trees with two parsers: one in which elementary trees were processed individually, and one in which overlapping elementary computations were shared. The results show that merging elementary computations results in a significant, though not spectacular, reduction in parse time, despite the increased amount of book-keeping required to make recovery of parse trees possible. In future work we plan to determine exactly how much this book-keeping adds to parse time by implementing a version of the merged parser in which book-keeping is omitted.

2. We ran both parsers on one 750MHz processor of a unloaded Sun Blade 1000 workstation with 1.5GB memory.

One rather significant drawback of this evaluation is that the amount of lexical ambiguity in this grammar is far less than is found in large-scale wide-coverage grammars such as the XTAG grammar (XTAG-Group, 1999). Although the tree-bank includes examples of a wide variety of syntactic constructions, for any individual word, the number of syntactic contexts (corresponding to alternative supertag possibilities for that word) that actually occur in the Penn Tree Bank is generally far less than those that would be included in the lexical entry for that word in a wide-coverage grammar. This is particularly true for words with low frequency. In future work we plan to look into ways of obtaining more complete mapping from a lexical item to the set of supertags it can anchor.

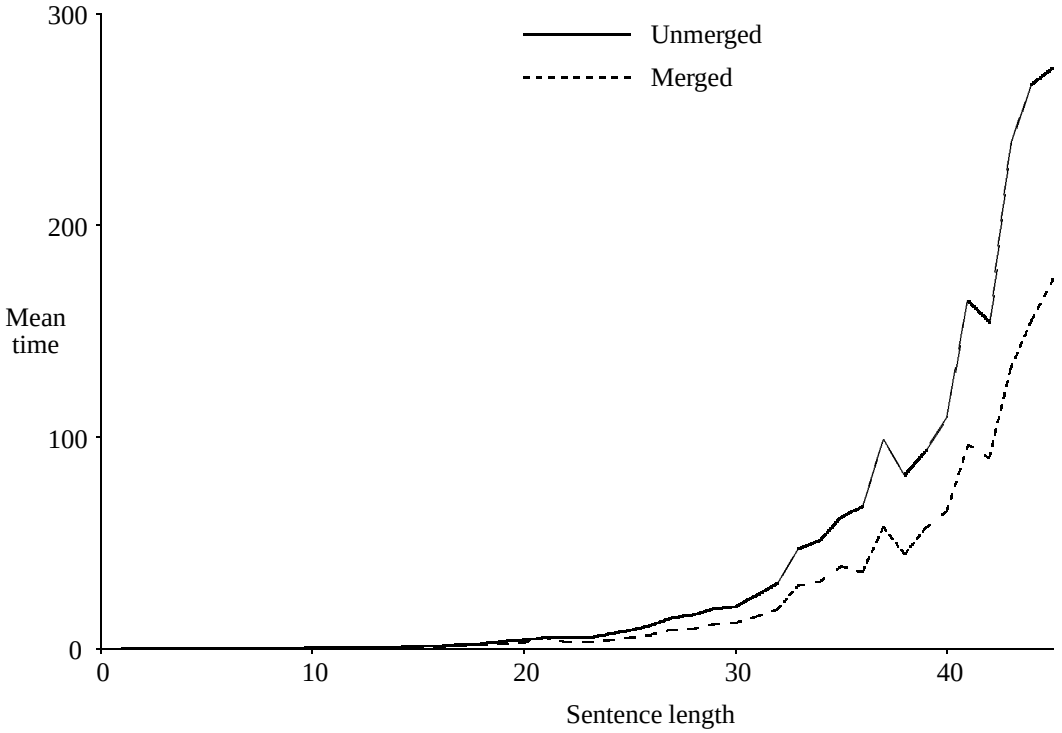

Figure 1: Comparison of Running Times

References

Bangalore, Srinivas. 1997a. *Complexity of Lexical Descriptions and its Relevance to Partial Parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia.

Bangalore, Srinivas. 1997b. Performance Evaluation of Supertagging for Partial Parsing. In *Proceedings of the Fifth International Workshop on Parsing Technologies*.

Becker, Tilman. 1993. *HyTAG: A new type of Tree Adjoining Grammar for hybrid syntactic representation of free word order languages*. Ph.D. thesis, Universitat des Saarlandes.

Becker, Tilman. 1994. Patterns in metarules. In *Proceedings of the Third International Workshop on Tree Adjoining Grammars*, pages 9–11.

Candito, Marie-Hélène. 1996. A principle-based hierarchical representation of LTAGs. In *Proceedings of the 16th International Conference on Computational Linguistics*, Copenhagen, Denmark, August.

Carroll, John, Nicolas Nicolov, Olga Shaumyan, Martine Smets and David Weir. 1998a. Grammar Compaction and Computation Sharing in Automaton-based Parsing. In *Proceedings of the First Workshop on Tabulation in Parsing and Deduction*, pages 16–25.

Carroll, John, Nicolas Nicolov, Olga Shaumyan, Martine Smets and David Weir. 1998b. The LEXSYS Project. In *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks*, pages 29–33.

Chen, John, Srinivas Bangalore and K. Vijay-Shanker. 1999. New Models for Improving Supertag Disambiguation. In *Proceedings of the Eighth Conference of the European Chapter of the Association for Computational Linguistics*.

Evans, Roger, Gerald Gazdar and David Weir. 1995. Encoding Lexicalized Tree Adjoining Grammars with a Nonmonotonic Inheritance Hierarchy. In *Proceedings of the 33rd Meeting of the Association for Computational Linguistics*, pages 77–84.

Evans, Roger and David Weir. 1997. Automaton-based Parsing For Lexicalized Grammars. In *Proceedings of the Fifth International Workshop on Parsing Technologies*, pages 66–76.

Evans, Roger and David Weir. 1998. A structure-sharing parser for lexicalized grammars. In *Proceedings of the 36th Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics*, pages 372–378.

Joshi, Aravind and Srinivas Bangalore. 1994. Disambiguation of super parts of speech (or supertags): almost parsing. In *Proceedings of the 15th International Conference on Computational Linguistics*, pages 154–160.

Marcus, Mitchell, Beatrice Santorini and Mary Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Sarkar, Anoop. 2000. Practical Experiments in Parsing using Tree Adjoining Grammars. In *Proceedings of the Fifth International Workshop on Tree Adjoining Grammars and Related Frameworks*.

Sarkar, Anoop, Fei Xia and Aravind Joshi. 2000. Some Experiments on Indicators of Parsing Complexity for Lexicalized Grammars. In *Efficiency in Large-Scale Parsing Systems*. Workshop held at COLING 2000.

Vijay-Shanker, K. and Yves Schabes. 1992. Structure Sharing in Lexicalized Tree-Adjoining Grammar. In *Proceedings of the 14th International Conference on Computational Linguistics*, pages 205–211.

Xia, Fei. 1999. Extracting Tree Adjoining Grammars from Bracketed Corpora. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium(NLPRS-99)*.

XTAG-Group, The. 1999. A Lexicalized Tree Adjoining Grammar for English. Technical Report `http://www.cis.upenn.edu/~xtag/ tech-report/tech-report.html`, The Institute for Research in Cognitive Science, University of Pennsylvania.

Sentence length	# items unmerged	# items merged	mean time unmerged	mean time merged
1	3	1	0.0	0.0
2	19	5	0.0	0.0
3	104	17	0.0	0.0
4	327	52	0.0	0.0
5	681	121	0.0	0.0
6	1256	235	0.0	0.0
7	2298	473	0.0	0.0
8	3923	864	0.1	0.0
9	5844	1347	0.1	0.1
10	9022	2180	0.2	0.1
11	12674	3203	0.2	0.2
12	18000	4649	0.4	0.2
13	26110	6928	0.5	0.4
14	34074	9165	0.7	0.5
15	47564	12969	1.0	0.8
16	62771	17481	1.3	1.0
17	80515	22809	1.8	1.4
18	99121	27909	2.3	1.7
19	128028	36790	3.3	2.5
20	163347	47322	4.1	2.9
21	193701	56268	5.1	5.2
22	277740	80430	5.1	3.1
23	274474	81562	5.0	3.1
24	354912	101143	6.8	4.1
25	427291	124919	8.5	5.3
26	532109	154792	10.9	6.7
27	683355	195608	14.5	8.9
28	731932	208338	15.9	9.3
29	855873	253130	18.8	11.6
30	873492	258383	19.7	12.3
31	1089989	314794	25.2	15.1
32	1291749	371601	30.8	18.6
33	1838306	556519	47.2	30.2
34	1917227	574944	51.1	31.6
35	2364987	710872	62.1	38.9
36	2487632	651374	67.1	36.1
37	3381691	982343	98.6	57.4
38	2864371	780416	82.0	44.7
39	3290281	979203	93.4	57.1
40	3755657	1106993	109.5	65.1
41	4993534	1467100	164.2	96.4
42	4843654	1380099	154.3	90.0
43	7071346	1983426	238.5	132.1
44	7655510	2282781	266.5	155.3
45	7772779	2317031	274.3	174.3

Table 1: Mean numbers of items and parse times (CPU seconds) per sentence, for sentences of length 1–45 words.