Practical Experiments in Parsing using Tree Adjoining Grammars[*]

Anoop Sarkar

Dept. of Computer and Information Sciences
University of Pennsylvania
200 South 33rd Street,
Philadelphia, PA 19104 USA
anoop@linc.cis.upenn.edu

Abstract

We present an implementation of a chart-based head-corner parsing algorithm for lexicalized Tree Adjoining Grammars. We report on some practical experiments where we parse 2250 *sentences from the Wall Street Journal using this parser. In these experiments the parser is run without any statistical pruning; it produces all valid parses for each sentence in the form of a shared derivation forest. The parser uses a large Treebank Grammar with* 6789 *tree templates with about* $120,000$ *lexicalized trees. The results suggest that the observed complexity of parsing for LTAG is dominated by factors other than sentence length.*

1. Motivation

The particular experiments that we report on in this paper were chosen to discover certain facts about LTAG parsing in a practical setting. Specifically, we wanted to discover the importance of the worst-case results for LTAG parsing in practice. Let us take Schabes' Earley-style TAG parsing algorithm (Schabes, 1994) which is the usual candidate for a practical LTAG parser. The parsing time complexity of this algorithm for various types of grammars are as follows (for input of length n):

$O(n^6)$ - TAGs for inherently ambiguous languages

$O(n^4)$ - unambiguous TAGs

$O(n)$ - bounded state TAGs e.g. the usual grammar G where $L(G) = \{a^n\ b^n\ e\ c^n\ d^n \mid n \geq 0\}$ (see (Joshi *et al.*, 1975))

The grammar factors are as follows: Schabes' Earley-style algorithm takes $O(|A||I \cup A|Nn^6)$ worst case time and $O(|A \cup I|Nn^4)$ worst case space, where n is the length of the input, A is the set of auxiliary trees, I is the set of initial trees and N is maximum number of nodes in an elementary tree.

Given these worst case estimates we wish to explore what the observed times might be for a TAG parser. It is not our goal here to compare different TAG parsing algorithms, rather it is to discover what kinds of factors can contribute to parsing time complexity. Of course, a natural-language grammar that is large and complex enough to be used for parsing real-world text is typically neither unambiguous nor bounded in state size. It is important to note that in this

[*] I would like to thank Aravind Joshi, Carlos Prolo and Fei Xia for their help and suggestions. This work was partially supported by NSF Grant SBR 8920230.

paper we are not concerned with *parsing accuracy*, rather we want to explore *parsing efficiency*. This is why we do not pursue any pruning while parsing using statistical methods. Instead we produce a shared derivation forest for each sentence which stores, in compact form, all derivations for each sentence. This helps us evaluate our TAG parser for time and space efficiency. The experiments reported here are also useful for statistical parsing using TAG since discovering the source of grammar complexity in parsing can help in finding the right *figures-of-merit* for effective pruning in a statistical parser.

2. Treebank Grammar

The grammar we used for our experiments was a Treebank Grammar which was extracted from Sections 02–21 of the Wall Street Journal Penn Treebank II corpus (Marcus *et al.*, 1993). We are grateful to Fei Xia for use of this grammar which was part of a separate study (Xia, 1999). The extraction converted the *derived* trees of the Treebank into *derivation* trees which represent the attachments of lexicalized elementary trees. There are 6789 tree templates in the grammar with $47,752$ tree nodes. Each word in the corpus selects some set of tree templates. The total number of lexicalized trees is $123,039$. The total number of word types in the lexicon is $44,215$. The average number of trees per word is 2.78. However, the average gives a misleading overall picture of the syntactic lexical ambiguity in the grammar.[1] Figure 1 shows the syntactic lexical ambiguity of the 150 most frequent words in the corpus. We shall return to this issue of lexical ambiguity when we evaluate our TAG parser. Finally, some lexicalized trees from the grammar are shown in Figure 2.

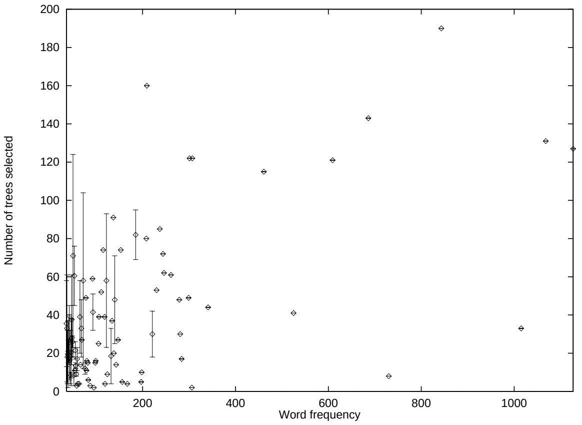


Figure 1: Number of trees selected by the 150 most frequent words in the input corpus. (x-axis: Word Frequency; y-axis: Number of Trees Selected)

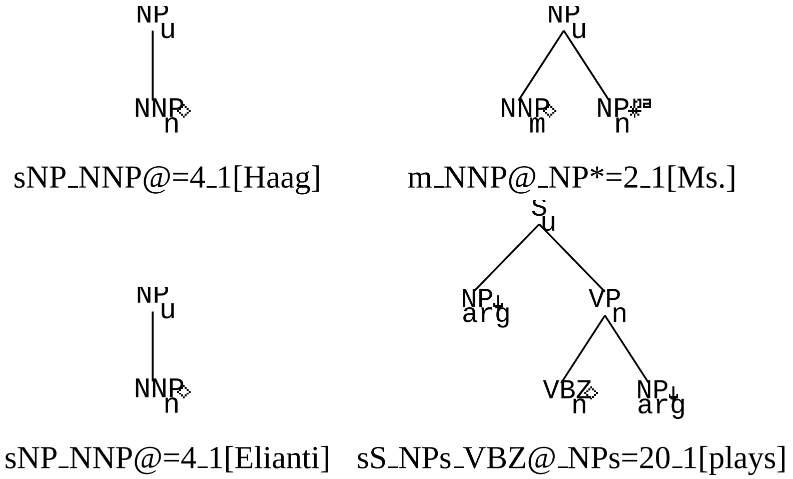
3. The Parser

3.1. *Parsing Algorithm*

The parser used in this paper implements a chart-based head-corner algorithm. The use of head-driven prediction to enchance efficiency was first suggested by (Kay, 1989) for CF parsing (see (Sikkel, 1997) for a more detailed survey). (Lavelli & Satta, 1991) provided the first head-driven algorithm for LTAGs which was a chart-based algorithm but it lacked any top-down prediction. (van Noord, 1994) describes a Prolog implementation of a head-corner parser for LTAGs which includes top-down prediction. Significantly, (van Noord, 1994) uses a different closure relation from (Lavelli & Satta, 1991). The head-corner traversal for auxiliary trees starts from the footnode rather than from the anchor.

The parsing algorithm we use is a chart-based variant of the (van Noord, 1994) algorithm. We use the same head-corner closure as proposed there. We do not give a complete de-

[1] We define the (syntactic) lexical ambiguity for a lexicalized TAG as the number of trees selected by a lexical item. Note that in a fully lexicalized formalism like LTAG, lexical ambiguity includes (to some extent) what would be considered to be a purely syntactic ambiguity in other formalisms.



sNP_NNP@=4_1[Haag] m_NNP@_NP*=2_1[Ms.]

sNP_NNP@=4_1[Elianti] sS_NPs_VBZ@_NPs=20_1[plays]

Figure 2: Example lexicalized elementary trees from the Treebank Grammar. They are shown in the usual notation: $\diamond$ = *anchor*, $\downarrow$= *substitution node*, $*$ = *footnode*, **na** = *null-adjunction constraint*. These trees can be combined using substitution and adjunction to parse the sentence *Ms. Haag plays Elianti*.

scription of our parser here since the basic idea behind the algorithm can be grasped by reading (van Noord, 1994). Our parser differs from the algorithm in (van Noord, 1994) in some important respects: our implementation is chart-based and explicitly tracks *goal* and *item* states and does not perform any implicit backtracking or selective memoization, we do not need any additional variables to keep track of which words are already ‘reserved’ by an auxiliary tree (which (van Noord, 1994) needs to guarantee termination), and we have an explicit *completion* step.

3.2. *Parser Implementation*

The parser is implemented in ANSI C and runs on SunOS 5.x and Linux 2.x. Apart from the Treebank Grammar used in this paper, the parser has been tested with the XTAG English Grammar and also with a Korean grammar.

The implementation optimizes for space at the expense of speed, e.g. the recognition chart is implemented as a sparse array thus taking considerably less than the worst case n^4 space and the lexical database is read dynamically from a disk-based hash table. For each input sentence, the parser produces as output a shared derivation forest which is a compact representation of all the derivation trees for that sentence. We use the definition of derivation forests for TAGs represented as CFGs, taking $O(n^4)$ space as defined in (Vijay-Shanker & Weir, 1993; Lang, 1994).

4. Input Data

The data used as input to the parser was a set of 2250 sentences from the WSJ Penn Treebank. The length of each sentence was 21 words or less. The average sentence length was 12.3 and the total number of tokens was $27,715$. These sentences were taken from the same sections as the input Treebank Grammar. This was done to avoid any processing difficulties which are incurred for handling unknown words properly.

5. Results

In this section we examine the performance of the parser on the input data (described in §4).[2] Figure 3 shows the time taken in seconds by the parser plotted against sentence length.[3] We see a great deal of variation in timing for the same sentence length, especially for longer sentences. This is surprising since all time complexity analyses reported for parsing algorithms assume that the only relevant factor is the length of the sentence. In this paper, we will explore whether sentence length is the only relevant factor.[4]

Figure 4 shows the median of time taken for each sentence length. This figure shows that for some sentences the time taken by the parser deviates by a large magnitude from the median case for the same sentence length. Next we considered each set of sentences of the same length to be a sample, and computed the standard deviation for each sample. This number ignores the outliers and gives us a better estimate of parser performance in the most common case. Figure 5 shows the plot of the standard deviation points against parsing time. The figure also shows that these points can be described by a linear function.

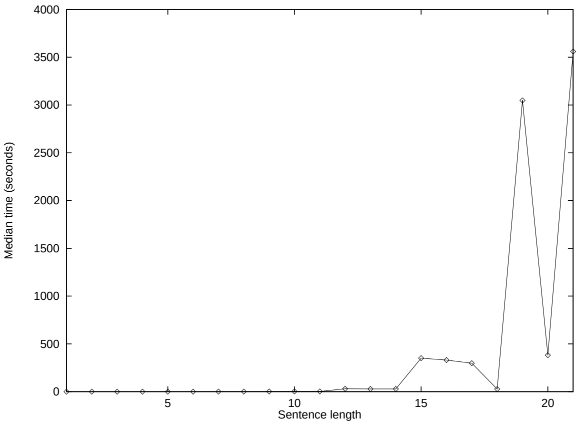


Figure 4: Median running times for the parser. (x-axis: Sentence length; y-axis: Median time in seconds)

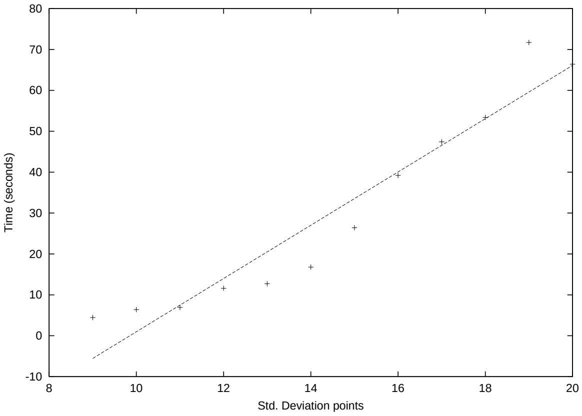


Figure 5: Least Squares fit over std. deviation points for each sentence length. Error was 9.078% and 13.74% for the slope and intercept respectively. We ignored sentences shorter than 8 words due to round-off errors; cf. Figure 3 (x-axis: Std. deviation points; y-axis: Time in seconds)

[2] The data was split into 45 equal sized chunks and parsed in parallel on a Beowulf cluster of Pentium Pro 200Mhz servers with 512MB of memory running Linux 2.2.

[3] From the total input data of 2250 sentences, 315 sentences did not get a parse. This was because the parser was run with the start symbol set to the label S. Of the sentences that did not parse 276 sentences were rooted at other labels such as FRAG, NP, etc. The remaining 39 sentences were rejected because a tokenization bug did not remove a few punctuation symbols which do not select any trees in the grammar.

[4] A useful analogy to consider is the run-time analysis of *quicksort*. For this particular sorting algorithm, it was detemined the distribution of the order of the numbers in the input array to be sorted was an extremely important factor to guarantee sorting in time $\Theta(nlogn)$. An array of numbers that is already completely sorted has time complexity $\Theta(n^2)$.

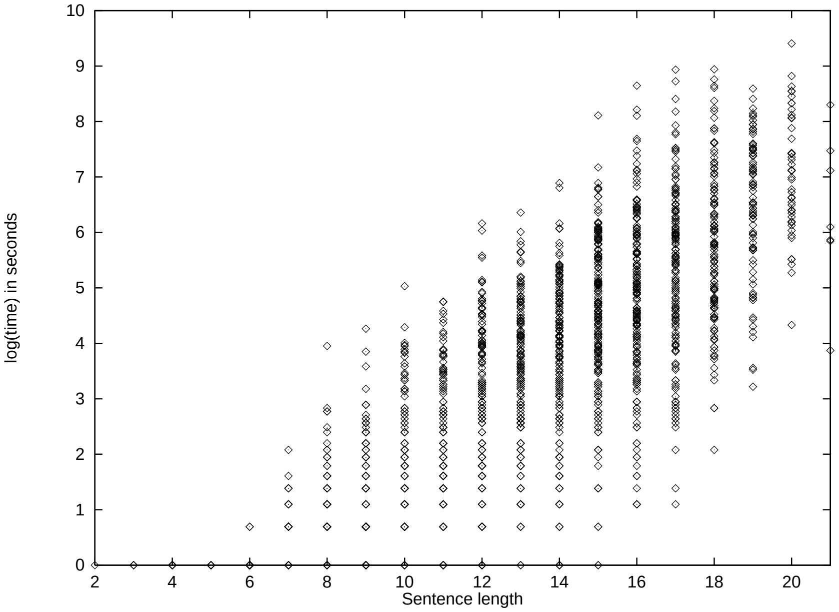


Figure 3: Parse times plotted against sentence length. (x-axis: Sentence length; y-axis: log(time) in seconds)

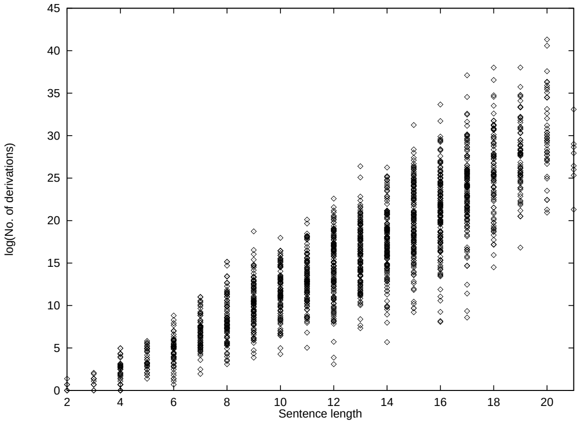


Figure 6: Log of number of derivations produced by the parser plotted against sentence length. (x-axis: Sentence length; y-axis: log(No. of derivations))

Figure 6 shows a plot of the number of derivations reported by the parser for each sentence plotted against sentence length. These derivations were never enumerated by the parser – the total number of derivations for each sentence was computed directly from the shared derivation forest reported by the parser. The size of the grammar has a direct relation to the large number of derivations reported by the parser. However, in this figure just as in the figure for the parsing times while there is an overall increase in the number of derivations as the sentence length increases, there is also a large variation in this number for identical sentence lengths. We wanted to discover the relevant variable other than sentence length which would be the right predictor of parsing time complexity.[5] As our analysis of the lexicon showed us (see Figure 1), there can be a large variation in syntactic lexical ambigu-

[5] Note that this variable cannot be the number of active or passive edges proposed by the parser since these values can only be computed at run-time.

ity which might be a relevant factor in parsing time complexity. To draw this out, in Figure 7 we plotted the number of trees selected by a sentence against the time taken to parse that sentence. From this graph we see that the number of trees selected is a better predictor than sentence length of increase in parsing complexity. Based on the comparison of the graph in Figure 7 with Figure 3, we assert that it is the syntactic lexical ambiguity of the words in the sentence which is the major contributor to parsing time complexity. One might be tempted to suggest that instead of number of trees selected, the number of derivations reported by the parser might be a better predictor of parsing time complexity.[6] We tested this hypothesis by plotting the number of derivations reported for each sentence plotted against the time taken to produce them (shown in Figure 8). The figure shows that the final number of derivations reported is not a valid predictor of parsing time complexity.

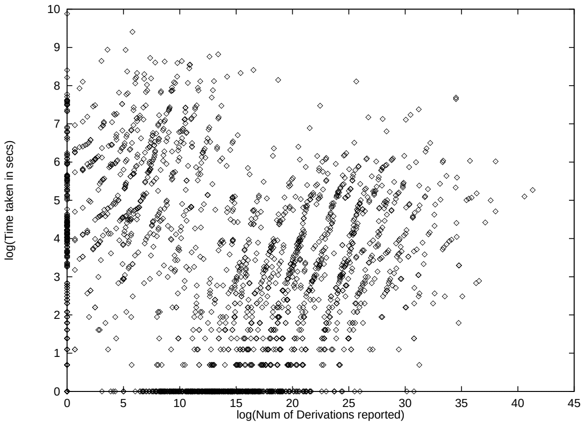


Figure 8: Number of derivations reported for each sentence plotted against the time taken to produce them (both axes are in log scale). (x-axis: log(Number of derivations reported); y-axis: log(Time taken) in seconds)

6. Conclusion

In this paper, we described an implementation of a chart-based head-corner parser for LTAGs. We ran some empirical tests by running the parser on 2250 sentences from the Wall Street Journal. We used a large Treebank Grammar to parse these sentences. We showed that the observed time complexity of the parser on these sentences does not increase predictably with longer sentence lengths. Looking at the derivations produced by the parser, we see a similar variation in the number of derivations for the same sentence length. We presented evidence that indicates that the number of trees selected by the words in the sentence (a measure of the syntactic lexical ambiguity of a sentence) is a better predictor of complexity in LTAG parsing.

References

JOSHI A. K., LEVY L. & TAKAHASHI M. (1975). Tree Adjunct Grammars. *Journal of Computer and System Sciences*.

KAY M. (1989). Head driven parsing. In *Proc. of IWPT '89*, p. 52–62, Pittsburgh, PA.

LANG B. (1994). Recognition can be harder than parsing. *Computational Intelligence*, **10** (4).

LAVELLI A. & SATTA G. (1991). Bidirectional parsing of Lexicalized Tree Adjoining Grammars. In *Proc. 5th EACL*, Berlin, Germany.

MARCUS M., SANTORINI B. & MARCINKIEWIECZ M. (1993). Building a large annotated corpus of english. *Computational Linguistics*, **19** (2), 313–330.

SCHABES Y. (1994). Left to right parsing of lexicalized tree adjoining grammars. *Computational Intelligence*, **10** (4).

SIKKEL K. (1997). *Parsing Schemata*. EATCS Series. Springer-Verlag.

VAN NOORD G. (1994). Head-corner parsing for TAG. *Computational Intelligence*, **10** (4).

[6] One of the anonymous reviewers of this paper suggested that this might be a useful indicator.

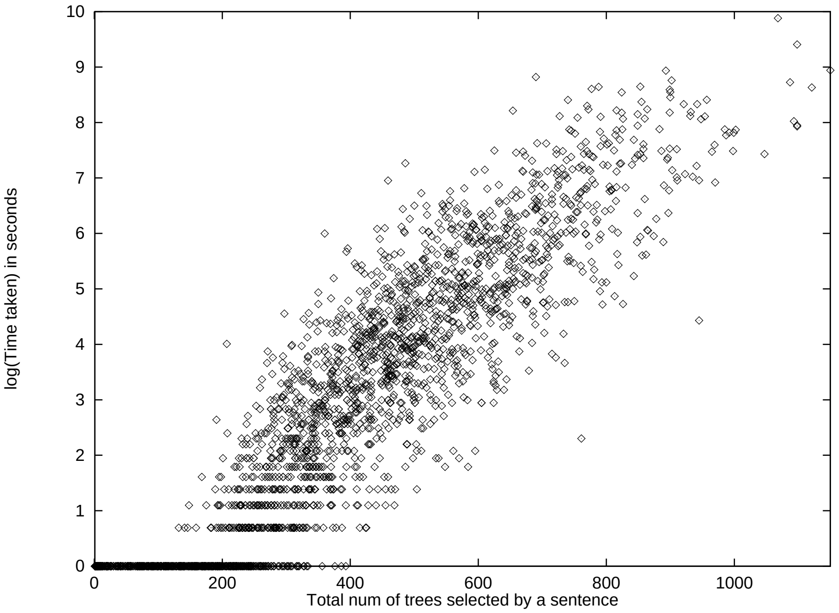


Figure 7: The impact of syntactic lexical ambiguity on parsing times. Log of the time taken to parse a sentence plotted against the total number of trees selected by the sentence. (x-axis: Total number of trees selected by a sentence; y-axis: log(Time taken) in seconds). Compare with Figure 3.

VIJAY-SHANKER K. & WEIR D. J. (1993). The use of shared forests in TAG parsing. In *Proc of 6th Meeting of the EACL*, p. 384–393, Utrecht, The Netherlands.

XIA F. (1999). Extracting tree adjoining grammars from bracketed corpora. In *Proceedings of 5th Natural Language Processing Pacific Rim Symposium (NLPRS-99)*, Beijing, China.