Combining Supertagging and Lexicalized Tree-Adjoining Grammar Parsing*

Anoop Sarkar
School of Computing Science,
Simon Fraser University
Burnaby, BC V5A 1S6, Canada
anoop@cs.sfu.ca

Abstract

In this paper we study various reasons and mechanisms for combining Supertagging with Lexicalized Tree-Adjoining Grammar (LTAG) parsing. Because of the highly lexicalized nature of the LTAG formalism, we experimentally show that notions other than sentence length play a factor in observed parse times. In particular, *syntactic lexical ambiguity* and *sentence complexity* (both are terms we define in this paper) are the dominant factors that affect parsing efficiency. We show how a Supertagger can be used to drastically reduce the syntactic lexical ambiguity for a given input and can be used in combination with an LTAG parser to radically improve parsing efficiency. We then turn our attention to from parsing efficiency to parsing accuracy and provide a method by which we can effectively combine the output of a Supertagger and a statistical LTAG parser using a co-training algorithm for bootstrapping new labeled data. This combination method can be used to incorporate new labeled data from raw text to improve parsing accuracy.

1 Introduction

In this paper we study two particular applications for the idea that it is possible to combine Supertagging with Lexicalized Tree-Adjoining Grammar (LTAG) parsing. A Supertagger (Srinivas and Joshi, 1999) is a statistical model that can be used to probabilistically assign to the word sequence (which makes up the input string)

* This research was partially supported by NSERC, Canada (RGPIN: 264905). Thanks to Aravind Joshi and Fei Xia for their collaboration on some aspects of this paper.

a sequence of elementary trees from some LTAG grammar. An LTAG parser on the other hand also has to attach the various possible elementary trees for each word in the input string to provide a complete derivation tree for the input (please see (Joshi and Schabes, 1992) for more information on the LTAG formalism and LTAG parsing).

The first argument to combine Supertagging with LTAG parsing comes from parsing efficiency. Because of the highly lexicalized nature of the grammar formalism we experimentally show that notions other than sentence length play a factor for speeding up observed parse times when finding all derivations for a given input string. In particular, *syntactic lexical ambiguity* and *sentence complexity* (both are terms we define more precisely later on in this paper) are the dominant factors that affect parsing efficiency. We then show how a Supertagger can be used to drastically reduce the syntactic lexical ambiguity for a given input and so can be used in combination with an LTAG parser to radically improve parsing efficiency. All of these ideas are tested out experimentally by parsing sentences from Penn TreeBank Wall Street Journal corpus (Marcus, Santorini, and Marcinkiewiecz, 1993).

The second argument to combine Supertagging with LTAG parsing comes from the need for two distinct statistical models that use (preferably) conditionally independent features in assigning probabilities to their input, while at the same time producing the same output. This enables the use of the co-training algorithm (Blum and Mitchell, 1998) to bootstrap new labeled data instances from raw unlabeled data. In this case, however, the combination of a Supertagger and an LTAG parser permits the use of the co-training algorithm to assign a complex label, a structured description to the input, rather than the typical case of a binary or multi-class classifier.

There is a third alternative perspective on combining a Supertagger with a parser that uses the same lexicalized structures and that is to improve the efficiency of a statistical parser. However this idea has been thoroughly explored in (Clark, 2002; Clark and Curran, 2004) which uses this same idea in a very related area of parsing for the Combinatory Categorial Grammar (CCG) formalism which also can exploit highly lexicalized structures as in the LTAG formalism. Because this perspective is discussed in these related publications, we will not discuss it here. One thing to note is that a statistical parser can use tricks such as beam search to improve parsing efficiency not available to a parser that finds all derivations. However based on the experimental evidence presented in (Clark, 2002; Clark and Curran, 2004) it seems as if the notions of syntactic lexical ambiguity and sentence complexity apply to this case as well.

2 Parsing Efficiency

The time taken by a parser to produce derivations for input sentences is typically associated with the length of those sentences. The longer the sentence, the more time the parser is expected to take. However, complex algorithms like parsers are typically affected by several factors. A common experience is that parsing algorithms differ in the number of edges inserted into the chart while parsing. In this paper, we explore some of these constraints from the perspective of lexicalized grammars and explore how these constraints might be exploited to improve parser efficiency.

We will show how Supertagging can effectively cut down on the observed time taken by an LTAG parser. First we show that various factors other than sentence length affect parsing complexity and then we will show that based on this hypothesis Supertagging can be used to effectively speed up an all-paths LTAG parser that produces a derivation forest. We show this experimentally using parsing experiments on the Penn Treebank corpus. We also describe how this approach can be useful for bootstrapping an LTAG statistical parser from labeled and unlabeled data.

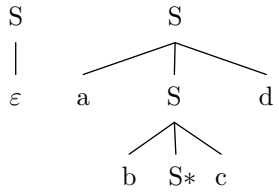
We concentrate on the problem of parsing using *fully* lexicalized grammars by looking at parsers for Lexicalized Tree Adjoining Grammar (LTAG). By a fully lexicalized grammar we mean a grammar in which there are one or more syntactic structures associated with each lexical item. In the case of LTAG each structure is a tree (or, in general, a directed acyclic graph). For each structure there is an explicit structural slot for each of the arguments of the lexical item. The various advantages of defining a lexicalized grammar formalism in this way are discussed in (Joshi and Schabes, 1991).

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):

$O(n^6)$ - arbitrary TAGs

$O(n^4)$ - unambiguous TAGs

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



The grammar factors are as follows: Schabes' Earley-style algorithm takes:

$O(|A| \cdot |I \cup A| \cdot N \cdot n^6)$ worst case time, and

$O(|A \cup I| \cdot N \cdot n^4)$ 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 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 (Caraballo and Charniak, 1998).

An example LTAG is shown in Figure 1. To parse the sentence *Ms. Haag plays Elianti* the parser has to combine the trees selected by each word in the sentence by using the operations of substitution and adjunction (the two composition operations in LTAG) producing a valid derivation for the sentence. The result is shown in Figure 2 where one possible derivation is shown. The derived tree or parse tree contains the constituency information (in practice, when using the Penn Treebank annotation style, we use an operation called sister-adjunction to generate the relatively flat trees from the Treebank, please see (Schabes and Shieber, 1994; Chiang, 2000) for more information on this slight modification to adjunction). The derivation tree is the history of how the elementary trees were combined to form the derived tree and hence is the more basic representation of an LTAG parse. For all-paths parsing, all possible derivation trees for an input can be compactly encoded in polynomial space as a derivation forest (Vijay-Shanker and Weir, 1993).

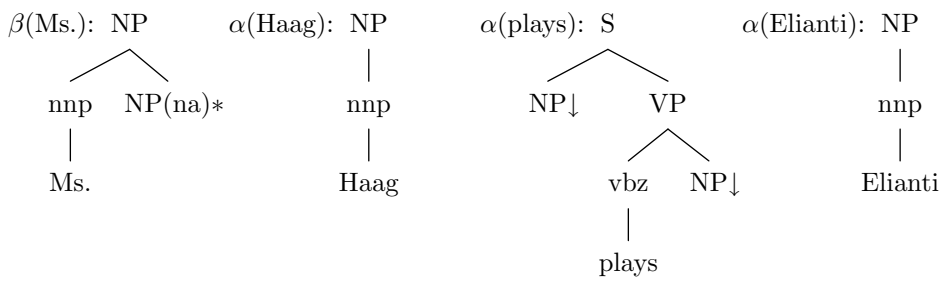


Figure 1: Example lexicalized elementary trees taken from a fragment of a lexicalized Tree Adjoining Grammar extracted from the Penn Treebank. They are shown using the usual notation: $\downarrow$ denotes a *substitution* node; $*$ denotes a footnode in an auxiliary tree which has the same root label and footnode label enabling it to rewrite an internal node in any elementary tree (called the *adjunction* operation); and *na* is the *null adjunction* constraint which states that no adjunction is permitted at that node.

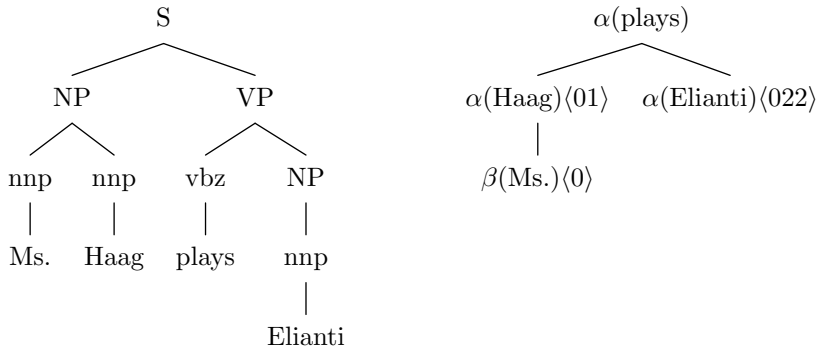


Figure 2: The parse tree and the derivation tree produced for the sentence *Ms. Haag plays Elianti*. The Gorn tree address of the attachment point in the parent node is provided within $\langle\rangle$ in the derivation tree. Note that the parser suppresses the printing of certain nodes in the elementary trees, for example, the node marked *na* in this grammar to enable the parser to produce parse trees that are identical to the trees in the Treebank.

Notice that as a consequence of this kind of lexicalized grammatical description there might be several different factors that affect parsing complexity. Each word can select many different trees; for example, the word *plays* in Figure 1 might select several trees for each syntactic context in which it can occur. The verb *plays* can be used in a relative clause, a wh-extraction clause, among others. While grammatical notions of argument structure and syntax can be processed in abstract terms just as in other kinds of formalisms, the crucial difference in LTAG is that all of this information is compiled into a finite set of trees *before* parsing. Each of these separate lexicalized trees is now considered by the parser. This compilation is repeated for other argument structures, e.g. the verb *plays* could also select trees which are intransitive thus increasing the set of lexicalized trees it can select. The set of trees selected by different lexical items is what we term in this paper as *lexical syntactic ambiguity*.

The importance of this compilation into a set of lexicalized trees is that each predicate-argument structure across each syntactic context has its own lexicalized tree. Most grammar formalisms use feature structures to capture the same grammatical and predicate-argument information. In LTAG, this larger set of lexicalized trees directly corresponds to the fact that recursive feature structures are not needed for linguistic description. Feature structures are typically atomic with a few instances of re-entrant features.

Thus, in contrast with LTAG parsing, parsing for formalisms like HPSG or LFG concentrates on efficiently managing the unification of large feature structures and also the packing of ambiguities when these feature structures subsume each other (see (Oepen and Carroll, 2000) and references cited there). We argue in this paper that the result of having compiled out abstract grammatical descriptions into a set of lexicalized trees allows us to predict the number of edges that will be proposed by the parser even before parsing begins. This allows us to explore novel methods of dealing with parsing complexity that are difficult to consider in formalisms that are not fully lexicalized.

Furthermore, as the sentence length increases, the number of lexicalized trees increase proportionally increasing the attachment ambiguity. Each sentence is composed of several clauses. In a lexicalized grammar, each clause can be seen as headed by a single predicate tree with its arguments and associated adjuncts. We shall see that empirically the number of clauses grow with increasing sentence length only up to a certain point. For sentences greater than a certain length the number of clauses do not keep increasing.

Based on these intuitions we identify the following factors that affect parsing complexity for lexicalized grammars:

Syntactic Lexical Ambiguity The number of trees selected by the words in the

sentence being parsed. We show that this is a better indicator of parsing time than sentence length. This is also a predictor of the number of edges that will be proposed by a parser, allowing us to better handle difficult cases *before* parsing.

Sentence Complexity The clausal complexity in the sentences to be parsed. We observe that the number of clauses in a sentence stops growing in proportion to the sentence length after a point. We show that before this point parsing complexity is related to attachment of adjuncts rather than attachment of arguments.

3 LTAG Treebank Grammar

The grammar we used for our experiments was a LTAG Treebank Grammar which was automatically extracted from Sections 02–21 of the Wall Street Journal Penn Treebank II corpus (Marcus, Santorini, and Marcinkiewiecz, 1993). The extraction tool (Xia, 1999) converted the *derived* trees of the Treebank into *derivation* trees in LTAG 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 type is 2.78. However, this average is misleading since it does not consider the frequency with which words that select a large number of trees occur in the corpus. In Figure 3 we see that many frequently seen words can select a large number of trees.

Another objection that can be raised against a Treebank grammar which has been automatically extracted is that any parsing results using such a grammar might not be indicative of parsing using a hand-crafted linguistically sophisticated grammar. To address this point (Xia and Palmer, 2000), compares this Treebank grammar with the XTAG grammar (XTAG-Group, 1998), a large-scale hand-crafted LTAG grammar for English. The experiment shows that 82.1% of template tokens in the Treebank grammar matches with a corresponding template in the XTAG grammar; 14.0% are covered by the XTAG grammar but the templates in two grammars look different because the Treebank and the XTAG grammar have adopted different analyses for the corresponding constructions; 1.1% of template tokens in the Treebank grammar are not linguistically sound due to annotation errors in the original Treebank; and the remaining 2.8% are not currently covered by the XTAG grammar. Thus, a total of 96.1% of the structures in the Treebank grammar match up with structures in the XTAG grammar.

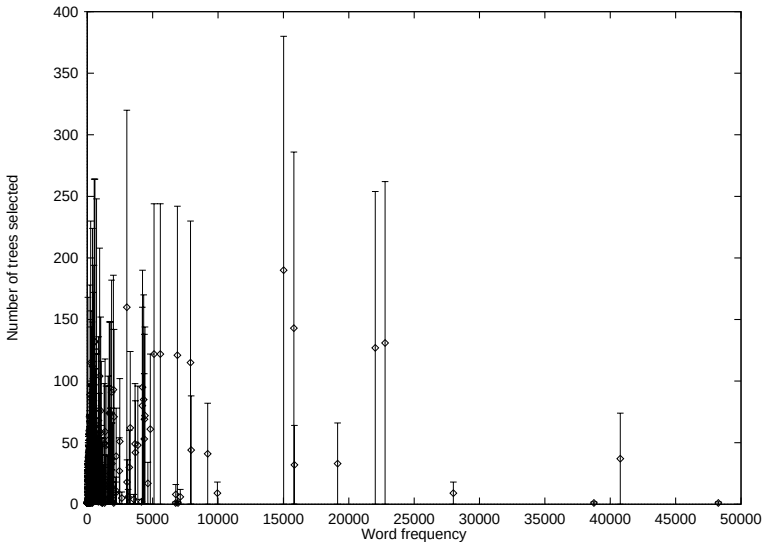


Figure 3: Number of trees selected plotted against words with a particular frequency. (x-axis: words of frequency x; y-axis: number of trees selected, error bars indicate least and most ambiguous word of a particular frequency x)

4 Experimental Setup

4.1 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 and 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 and 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 description 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.

4.2 Parser Implementation

The parser is implemented in ANSI C. 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 and Weir, 1993; Lang, 1994).

4.3 Input Data

To test the performance of LTAG parsing on a realistic corpus using a large grammar (described above) we parsed 2250 sentences from the Wall Street Journal using the lexicalized grammar described in Section 3.[1] 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 (02-21) of the Treebank from which the original grammar was extracted. This was done to avoid the complication of using default rules for unknown words.

4.4 Parsing Setup

In this section we examine the performance of the parser on the input data (described in Section 4.3).[2] Figure 4 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

[1] Some of these results appear in (Sarkar, 2000). In this section we present some additional data on the previous results and also the results of some new experiments that do not appear in the earlier work.

[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.

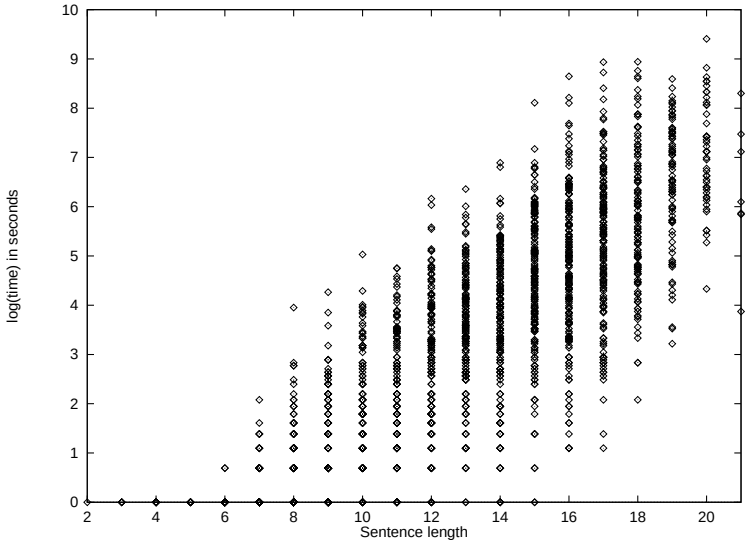


Figure 4: Parse times plotted against sentence length. Coefficient of determination: $R^2 = 0.65$. (x-axis: Sentence length; y-axis: log(time in seconds))

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 5 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 6 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.

[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 $\Theta(n^2)$.

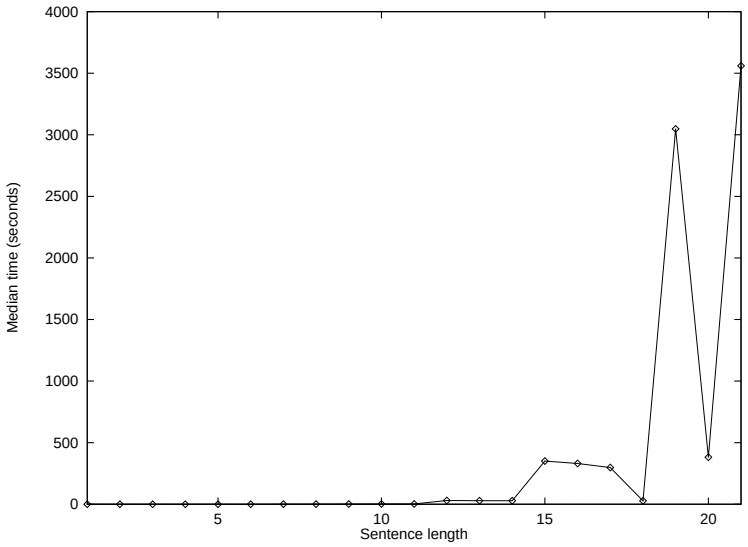


Figure 5: Median running times for the parser. (x-axis: Sentence length; y-axis: Median time in seconds)

5 Syntactic Lexical Ambiguity

In a fully lexicalized grammar such as LTAG the combinations of trees (by substitution and adjunction) can be thought of as *attachments*. It is this perspective that allows us to define the parsing problem in two steps (Joshi and Schabes, 1991):

1. Assigning a set of lexicalized structures to each word in the input sentence.
2. Finding the correct attachments between these structures to get all parses for the sentence.

In this section we will try to find which of these factors determines parsing complexity when finding all parses in an LTAG parser.

In all of the experiments reported here, the parser produces all parses for each sentence. It produces a shared derivation forest for each sentence which stores, in compact form, all derivations for each sentence.

We found that the observed complexity of parsing for LTAG is dominated by factors other than sentence length.[5] Figure 4 shows the time taken in seconds by

[5]Note that the precise number of edges proposed by the parser and other common indicators of complexity can be obtained only while or after parsing. We are interested in *predicting* parsing complexity.

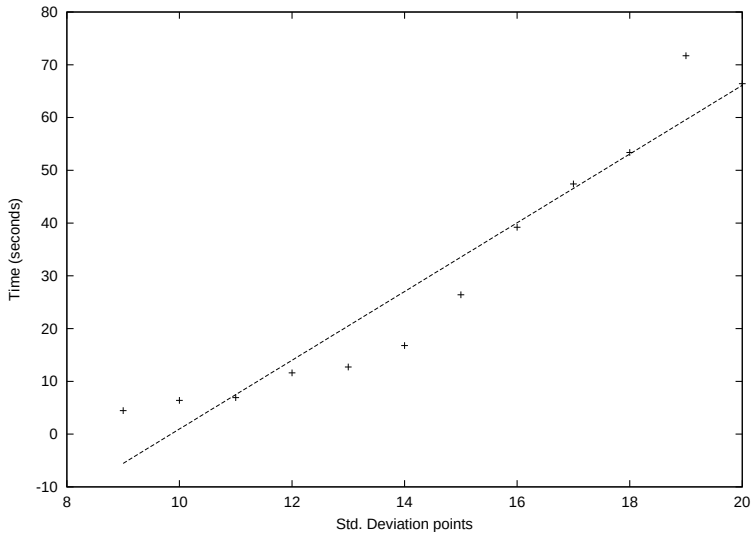


Figure 6: Least Squares fit over standard deviation points for each sentence length. Error was 9.078% and 13.74% for the slope and intercept respectively. We ignored sentences shorter than 8 words due to round-off errors; cf. Figure 4 (x-axis: Std. deviation points; y-axis: Time in seconds)

the parser plotted against sentence length. We see a great deal of variation in timing for the same sentence length, especially for longer sentences.

We wanted to find the relevant variable other than sentence length which would be the right predictor of parsing time complexity. There can be a large variation in syntactic lexical ambiguity 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. By examining this graph we can visually infer that the number of trees selected is a better predictor of increase in parsing complexity than sentence length. We can also compare numerically the two hypotheses by computing the coefficient of determination (R^2) for the two graphs. We get a R^2 value of 0.65 for Figure 4 and a value of 0.82 for Figure 7. Thus, we infer that it is the syntactic lexical ambiguity of the words in the sentence which is the major contributor to parsing time complexity. Note that Supertagging directly addresses this issue of syntactic lexical ambiguity and we will show how Supertagging can help parsing efficiency in Section 7.

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

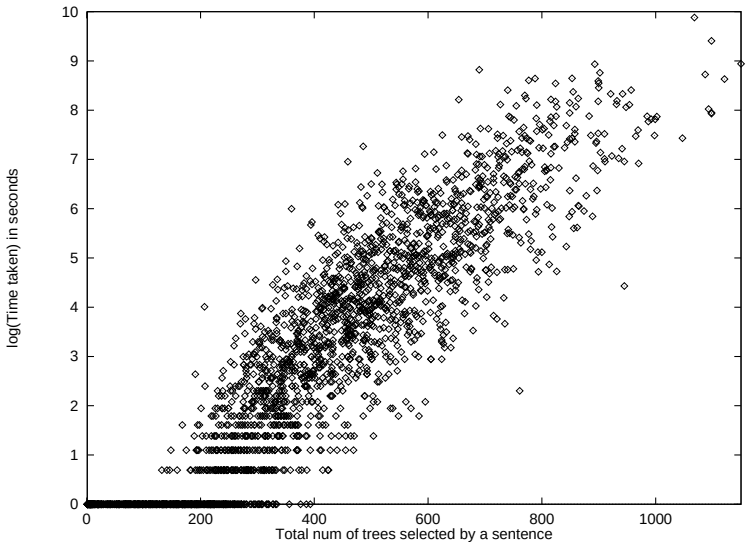


Figure 7: The impact of syntactic lexical ambiguity on parsing times. Log of the time taken to parse a sentence plotted against the total number of trees selected by the sentence. Coefficient of determination: $R^2 = 0.82$. (x-axis: Total number of trees selected by a sentence; y-axis: log(time) in seconds).

time complexity. 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 clearly not a valid predictor of parsing time complexity.

6 Sentence Complexity

There are many ways of describing sentence complexity[6], which are not necessarily independent of each other. In the context of lexicalized tree-adjoining grammar (and in other lexical frameworks, perhaps with some modifications) the complexity of syntactic and semantic processing is related to the number of predicate-argument structures being computed for a given sentence.

In this section, we explore the possibility of characterizing sentence complexity in terms of the number of clauses which is used as an approximation to the number of predicate-argument structures to be found in a sentence.

[6]This section is based on some joint work with Fei Xia and Aravind Joshi, previously published as a workshop paper (Sarkar, Xia, and Joshi, 2000).

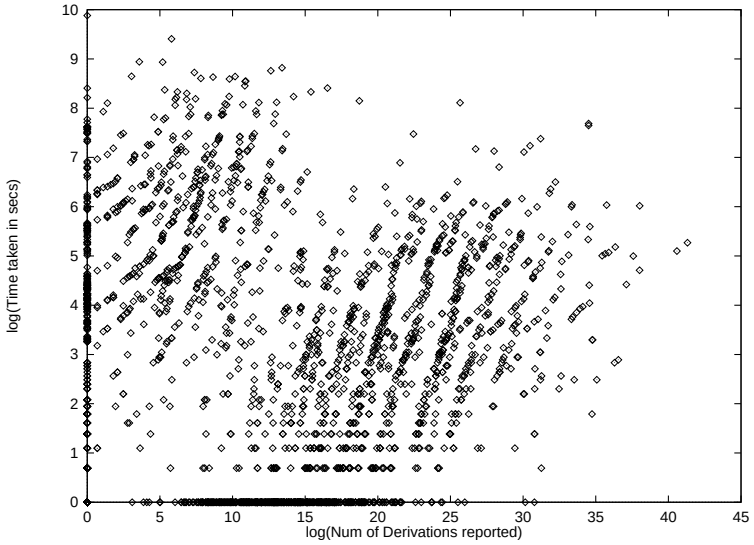


Figure 8: Number of derivations reported for each sentence plotted against the time taken to produce them (both axes are in log scale). (x-axis: log(Number of derivations reported); y-axis: log(Time taken) in seconds)

The number of clauses of a given sentence in the Penn Treebank is counted using the bracketing tags. The count is computed to be the number of S/SINV/SQ/RRC nodes which have a VP child or a child with -PRD function tag. In principle number of clauses can grow continuously as the sentence length increases. However it is interesting to note that 99.1% of sentences in the Penn Treebank contain 6 or fewer clauses.

Figure 9 shows the average number of clauses plotted against sentence length. For sentences with no more than 50 words, which accounts for 98.2% of the corpus, we see a linear increase in the average number of clauses with respect to sentence length. But from that point on, increasing the sentence length does not lead to a proportional increase in the number of clauses. Thus, empirically, the number of clauses is bounded by a constant. For some very long sentences, the number of clauses actually decreases because these sentences include long but flat coordinated phrases.

Figure 10 shows the standard deviation of the clause number plotted against sentence length. There is an increase in deviation for sentences longer than 50 words. This is due to two reasons: first, quite often, long sentences either have many embedded clauses or are flat with long coordinated phrases; second, the data

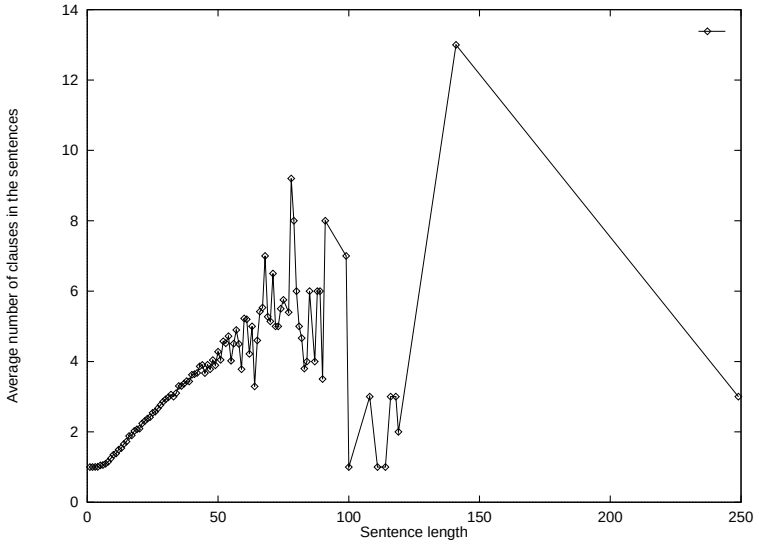


Figure 9: Average number of clauses plotted against sentence length.

become sparse as the sentence length grows, resulting in high deviation.[7]

In Figure 11 and Figure 12 we show how parsing time varies as a function of the number of clauses present in the sentence being parsed. The figures are analogous to the earlier graphs relating parsing time with other factors (see Figure 4 and Figure 7). Surprisingly, in both graphs we see that when the number of clauses is small (in this case less than 5), an increase in the number of clauses has no effect on the parsing complexity. Even when the number of clauses is 1 we find the same pattern of time complexity that we have seen in the earlier graphs when we ignored clause complexity. Thus, when the number of clauses is small parsing complexity is related to attachment of adjuncts rather than arguments. It would be interesting to continue increasing the number of clauses and the sentence length and then compare the differences in parsing times.

We have seen that beyond a certain sentence length, the number of clauses do not increase proportionally. We conjecture that a parser can exploit this observed constraint on clause complexity in sentences to improve its efficiency. In a way similar to methods that account for low attachment of adjuncts while parsing, we can introduce constraints on how many clauses a particular node can dominate in a parse. By making the parser sensitive to this measure, we can prune out unlikely

[7]For some sentence lengths (e.g., length $= 250$), there is only one sentence with that length in the whole corpus, resulting in zero deviation.

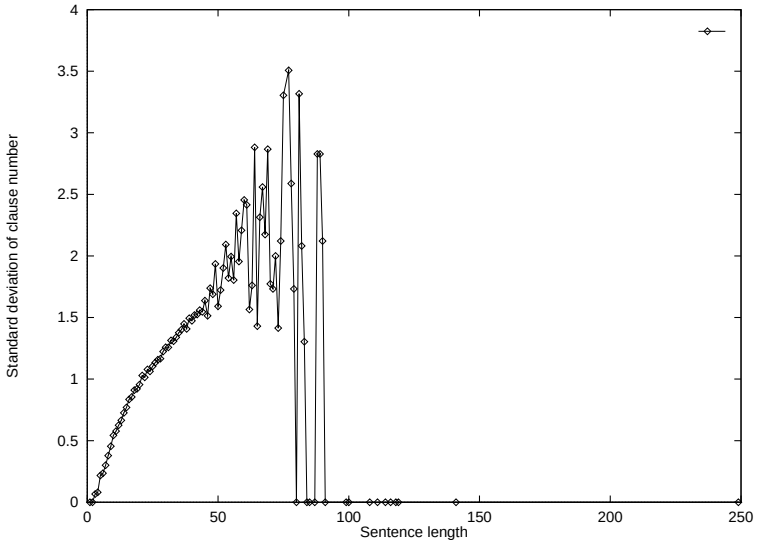


Figure 10: Standard deviation of number of clauses against sentence length.

derivations previously considered to be plausible by the parser. There is also an independent reason for pursuing this measure of clausal complexity. It can be extended to a notion of syntactic and semantic complexity as they relate to both the representational and processing aspects (Joshi, 2000). The empirical study of clausal complexity described in this section might shed some light on the general issue of syntactic and semantic complexity.

7 Supertagging helps Parsing Efficiency

Since we can easily determine the number of trees selected by a sentence before we start parsing, we can use this number to predict the number of edges that will be proposed by a parser when parsing this sentence, allowing us to better handle difficult cases *before* parsing.

We test the above hypothesis further by parsing the same set of sentences as above but this time using an oracle which tells us the correct elementary lexicalized structure for each word in the sentence. This eliminates lexical syntactic ambiguity but does not eliminate attachment ambiguity for the parser. The graph comparing the parsing times is shown in Figure 13. As the comparison shows, the elimination of lexical ambiguity leads to a drastic increase in parsing efficiency. The total time taken to parse all 2250 sentences went from 548K seconds to 31.2 seconds.

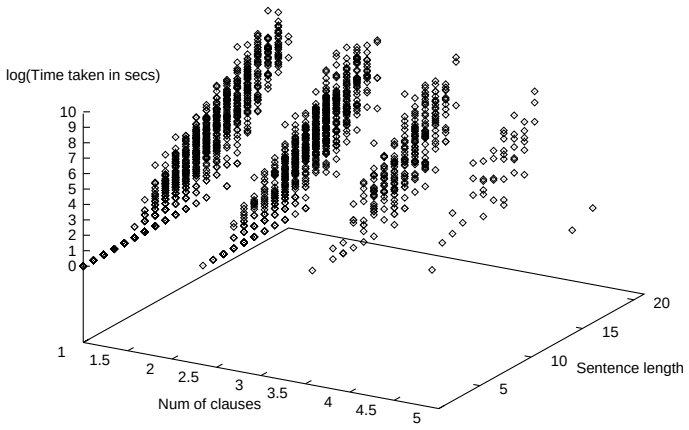


Figure 11: Variation in times for parsing plotted against length of each sentence while identifying the number of clauses.

Figure 13 shows us that a model which disambiguates syntactic lexical ambiguity can potentially be extremely useful in terms of parsing efficiency. Thus disambiguation of tree assignment or Supertagging (Srinivas, 1997) of a sentence before parsing it might be a way of improving parsing efficiency. This gives us a way to reduce the parsing complexity for precisely the sentences which were problematic: the ones which selected too many trees. To test whether parsing times are reduced after Supertagging we conducted an experiment in which the output of an n-best Supertagger was taken as input to the parser. In our experiment we set n to be 60.[8] The time taken to parse the same set of sentences was again dramatically reduced (the total time taken was 21K seconds). However, the disadvantage of this method was that the coverage of the parser was somewhat reduced. This was because some crucial tree was missing in the n-best output. Such coverage issues can be fixed fairly easily by having a LTAG grammar with sister adjunction, so that even with a small set of trees in the initial set, sister adjunction can be used to create parse trees for all input strings, with only a slight penalty in accuracy. The results are graphed in Figure 14. The total number of derivations for all sen-

[8](Chen, Bangalore, and Vijay-Shanker, 1999) show that to get greater than 97% accuracy using Supertagging the value of n must be quite high ($n > 40$). They use a different set of Supertags and so we used their result simply to get an approximate estimate of the value of n.

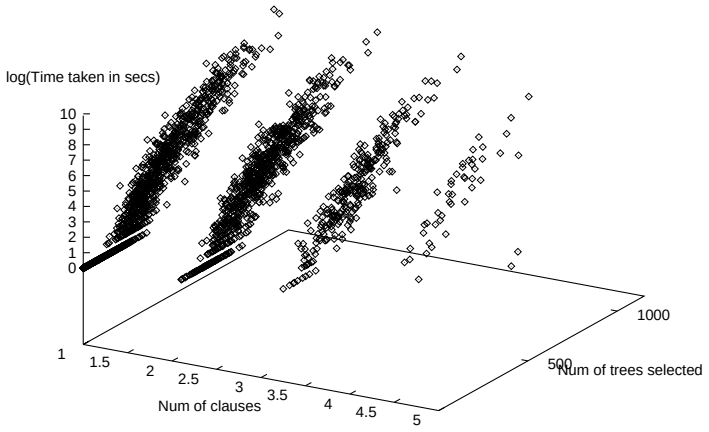


Figure 12: Variation in times for parsing plotted against the number of trees selected by each sentence while identifying the number of clauses.

tences went down to 1.01e+10 (the original total number was 1.4e+18) indicating (not surprisingly) that some attachment ambiguities persist although the number of trees are reduced.

8 Bootstrapping an LTAG Parser using a Supertagger

In this section we provide another application to the ideas presented in Section 7. Consider the derivation tree in Figure 2. The probability of a sequence of elementary trees $t_0, t_1, \ldots, t_n$ assigned to the n words in the sentence $S = w_0, \ldots, w_n$ is computed using a statistical model for Supertagging (Srinivas and Joshi, 1999) which is the probability of generating a Supertag sequence and the input word sequence. We can think of the probability distribution over Supertag sequences given the input string as a particular instantiation of a scoring function f_A over such sequences.

$$f_A = P(t_0, t_1, \ldots, t_n, S)$$

Models for Supertagging are described in great detail elsewhere in this book.

In a similar fashion, the derivation tree itself can be assigned a probability. Let S be the input sentence and T be the derivation tree for that sentence. A statistical

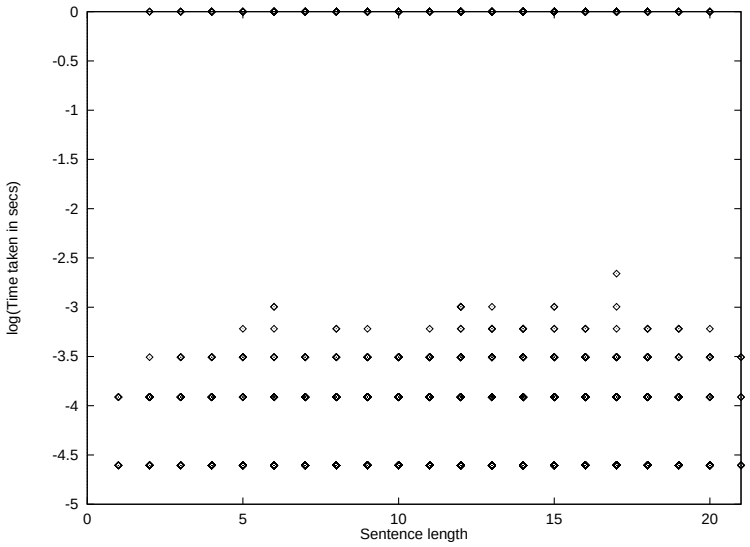


Figure 13: An oracle experiment: parse times when the parser gets the correct tree for each word in the sentence (eliminating any syntactic lexical ambiguity). The parsing times for all the 2250 sentences for all lengths never goes above 1 second. Total time was reduced from 548K seconds to 31.2 seconds. (x-axis: Sentence length; y-axis: log(time) in seconds)

parsing model defines $P(T \mid S)$. Using this model we can find the best parse T^*:

$$T^* = \underset{T}{\operatorname{argmax}}\, P(T \mid S) = \frac{P(T,S)}{P(S)} \approx P(T,S)$$

So the best parse will be $T^* = \operatorname{argmax}_T P(T,S)$. A statistical model for LTAG (Schabes, 1992) computes $P(T,S)$. As before, we can think of the probability distribution over derivations given the input string as an instantiation of a scoring function over derivations which we will call f_B.

$$f_B = P(T,S) = P(\alpha_0) \cdot \prod_{i=1\ldots n-1} P(\beta_i \mid \gamma, \eta)$$

for all the n trees in the derivation tree, where either the tree is at the root of the derivation and generated with probability $P(\alpha_0)$ or it is not the root in which case the tree β_i is generated with probability $P(\beta_i \mid \gamma, \eta)$ which is the probability of β_i substituting or adjoining into some tree γ at node η. This is a generative model, so any tree γ previously generated can be used in the conditioning context to generate

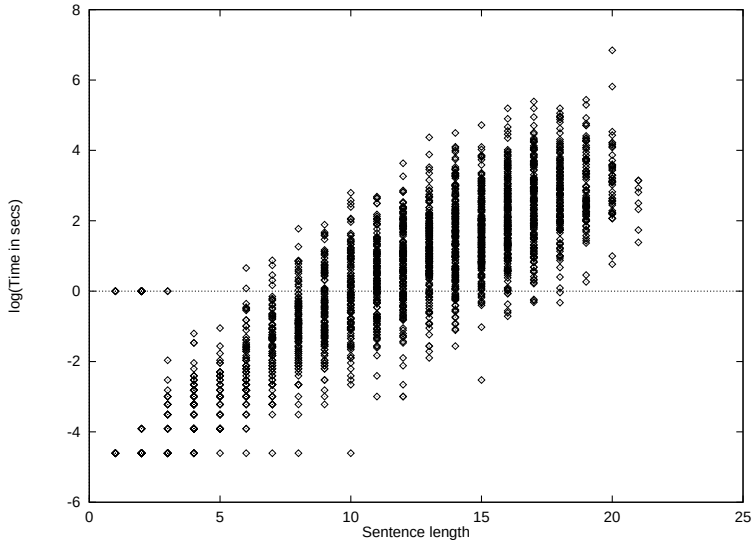


Figure 14: Time taken by the parser after n-best Supertagging ($n = 60$). Total time was reduced from 548K seconds to 21K seconds. (x-axis: Sentence length; y-axis: log(time) in seconds)

another tree in the derivation until all the trees in the LTAG derivation tree are generated. This is analogous to the probability distribution over parse trees defined in a probabilistic context-free grammar.

A key issue is how to estimate all of the above probability distributions. The supervised learning solution is to estimate these probability distributions using maximum likelihood estimation from hand annotated data: the correct Supertag sequence for each sentence in training data to estimate probabilities for the Supertagger, and the correct LTAG derivation for each sentence in the training data to estimate probabilities for the LTAG statistical parser. There are standard heuristic methods to convert the Treebank parse trees into LTAG derivations (Xia, 1999; Chiang, 2000; Chen and Vijay-Shanker, 2000) and also see Section 3. Training in the supervised setting simply amounts to relative frequency counts which are normalized to provide the probabilities needed[9].

Instead of supervised learning, we consider here the setting of a model trained on small amount of seed labeled data combined with a model trained on larger amounts of unlabeled or raw data, where the correct annotations are not provided.

[9]There are issues about smoothing these distributions which we do not cover here as they are not pertinent to our discussion here.

A is a statistical n-best Supertagger, and
B is a statistical LTAG parser.
M_A^i and M_B^i are models of A and B at step i.
U is a large pool of unlabelled sentences.
U^i is a small cache holding subset of U at step i.
L is the manually labelled seed data.
L_A^i and L_B^i are the labelled training examples for A and B at step i.

Initialise:

$L_A^0 \leftarrow L_B^0 \leftarrow L$
$M_A^0 \leftarrow \mathit{Train}(A, L_A^0)$
$M_B^0 \leftarrow \mathit{Train}(B, L_B^0)$

Loop:

$U^i \leftarrow$ Add unlabeled sentences from U.
M_A^i and M_B^i parse each sentences in U^i and assign scores to n-best Supertags and most likely LTAG derivation using scoring functions f_A and f_B.
For each sentence in U_i, parse the n-best Supertags using the LTAG parser (cf.§7)
For each sentence in U_i, select most likely "parse" for Supertagger by choosing the parse from the LTAG parser output forest that has the highest f_A score.
Select new parses $\{P_A\}$ and $\{P_B\}$ from parsed U_i according to selection method S which uses scores from f_A and f_B.
L_A^{i+1} is L_A^i augmented with $\{P_B\}$
L_B^{i+1} is L_B^i augmented with $\{P_A\}$
$M_A^{i+1} \leftarrow \mathit{Train}(A, L_A^{i+1})$
$M_B^{i+1} \leftarrow \mathit{Train}(B, L_B^{i+1})$

Return:

M_A^T and M_B^T after T rounds of co-training.

Figure 15: Pseudo-code for the co-training algorithm

This setting is attractive for several reasons:

- Annotating sentences is an expensive process. A parser that can be trained on a small amount of labeled data will reduce this annotation cost.
- Creating statistical parsers for novel domains and new languages will become easier.
- Combining labeled data with unlabeled data allows exploration of unsupervised methods which can now be tested using evaluations compatible with supervised statistical parsing.

The idea presented in this section is to combine a Supertagger and an LTAG statistical parser. However, instead of feeding the output of the Supertagger to an LTAG parser as we did in Section 7, we instead use the each model to create newly parsed data for the other model. This newly parsed data is treated as new labeled data, which is used to retrain each model in an iterative augmentation process.

The pair of probabilistic models, a Supertagger and an LTAG statistical parser, can be exploited to bootstrap new information from unlabeled data. Since both of these steps ultimately have to agree with each other, we can utilize an iterative method called co-training (Blum and Mitchell, 1998) that attempts to increase agreement between a pair of statistical models by exploiting mutual constraints between their output. Co-training has some good generalization properties when the features or views used by the two models are conditionally independent of each other (Dasgupta, Littman, and McAllester, 2002) and co-training can be formulated to be a greedy search for agreement in the feature space of the two models, even if the two models are not conditionally independent (Abney, 2002). In the experiments reported here however we will use the simple greedy algorithm provided by (Blum and Mitchell, 1998) extended to handle the multiple views provided by a Supertagger and an LTAG parser.

Previous approaches that use co-training were on tasks that involved identifying the right label from a small set of labels (typically 2–3), and in a relatively small parameter space. Compared to these earlier models, a statistical parser has a very large parameter space and the labels that are expected as output are parse trees which have to be built up recursively. Co-training can be informally described in the following manner:

- Pick two (or more) "views" of a classification problem.
- Build separate models for each of these "views" and train each model on a small set of labeled data.

- Sample an unlabeled data set and to find examples that each model independently labels with high confidence. (Nigam and Ghani, 2000)
- Confidently labeled examples can be picked in various ways. (Collins and Singer, 1999; Goldman and Zhou, 2000)
- Take these examples as being valuable as training examples and iterate this procedure until the unlabeled data is exhausted.

Effectively, by picking confidently labeled data from each model to add to the training data, one model is labeling data for the other model.

The pseudo-code for the co-training algorithm we used to bootstrap an LTAG parser using a Supertagger is shown in Figure 15. Note that f_A is the probability model for a Supertag sequence used by a statistical Supertagger, f_B is the probability model for LTAG derivations used by the LTAG parser, and the selection method S used here is simple *top-k* selection where the k labeled outputs from U_i with the highest scores are selected. Other variants are explored in (Steedman et al., 2003b; Steedman et al., 2003a; Hwa et al., 2003).

Experiments using this approach were reported in (Sarkar, 2001) The experiments used the Penn Treebank WSJ Corpus (Marcus, Santorini, and Marcinkiewiecz, 1993). The various settings for the co-training algorithm (cf. Figure 15) are as follows:

- L was set to Sections 02-06 of the Penn Treebank WSJ (9625 sentences)
- U was 30137 sentences (Section 07-21 of the Treebank stripped of all annotations).
- We used a tag dictionary of all lexicalized trees from *labeled* and *unlabeled* for unseen words.
- Novel trees were treated as unknown tree tokens.
- The cache U^i was set to be 3000 sentences.

While it might seem expensive to run the parser over the cache multiple times, we use the pruning capabilities of the parser to good use here. During the iterations we set the beam size to a value which is likely to prune out all derivations for a large portion of the cache except the most likely ones. This allows the parser to run faster, hence avoiding the usual problem with running an iterative algorithm over thousands of sentences. In the initial runs we also limit the length of the sentences entered into the cache since we select parses based on their probability and shorter

sentences tend to receive higher probability than longer sentences. The beam size is reset when running the parser on the test data to allow the parser a better chance at finding the most likely parse.

We scored the output of the parser on Section 23 of the Wall Street Journal Penn Treebank. The following are some aspects of the scoring that might be useful for comparision with other results: No punctuations are scored, including sentence final punctuation. Empty elements are not scored. We used EVALB (written by Satoshi Sekine and Michael Collins) which scores based on PARSEVAL (Black et al., 1991); with the standard parameter file (as per standard practice, part of speech brackets were not part of the evaluation). Also, we used Adwait Ratnaparkhi's part-of-speech tagger (Ratnaparkhi, 1996) to tag unknown words in the test data.

We obtained 80.02% and 79.64% labeled bracketing precision and recall respectively (as defined in (Black et al., 1991)). The baseline model which was only trained on the 9695 sentences of labeled data performed at 72.23% and 69.12% precision and recall. These results show that training a statistical parser using our co-training method to combine labeled and unlabeled data strongly outperforms training only on the labeled data.

It is important to note that unlike previous studies which aim to infer a grammar in an unsupervised setting (Marcken, 1995), our method is a step towards unsupervised parsing but the output results are directly compared to the output of supervised parsers.

9 Conclusion

In this paper, we identified syntactic lexical ambiguity and sentence complexity as factors that contribute to parsing complexity in fully lexicalized grammars. We conducted various experiments by parsing the Penn Treebank WSJ corpus using an LTAG grammar extracted from the annotated corpus. The experiments led to make the following conclusions.

We showed that lexical syntactic ambiguity has a strong effect on parsing time and that a model which disambiguates syntactic lexical ambiguity can potentially be extremely useful in terms of parsing efficiency. By assigning each word in the sentence with the correct elementary tree showed that parsing times were reduced by several orders of magnitude.

We showed that even as sentence length increases the number of clauses is empirically bounded by a constant. The number of clauses in 99.1% of sentences in the Penn Treebank was bounded by 6. We discussed how this finding affects parsing efficiency and showed that for when the number of clauses is smaller than 4, parsing efficiency is dominated by adjunct attachments rather than argument

attachments. We conducted an experiment in which the output of an n-best Supertagger was taken as input to the parser. The time taken to parse the same set of sentences was again dramatically reduced.

We also showed how the combination of a Supertagger with a LTAG parser can be effectively used in a co-training approach for situations in which there is insufficient training data.

References

Abney, Steven. 2002. Bootstrapping. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 360–367, Philadelphia, PA.

Black, E., S. Abney, D. Flickinger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of english grammars. In *Proc. DARPA Speech and Natural Language Workshop*, pages 306–311. Morgan Kaufmann.

Blum, Avrim and Tom Mitchell. 1998. Combining Labeled and Unlabeled Data with Co-Training. In *In Proc. of 11th Annual Conf. on Comp. Learning Theory (COLT)*, pages 92–100.

Caraballo, Sharon A. and Eugene Charniak. 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24(2):275–298.

Chen, John, Srinivas Bangalore, and K. Vijay-Shanker. 1999. New models for improving supertag disambiguation. In *Proceedings of the 9th Conference of the European Chapter of the Association for Computational Linguistics*, Bergen, Norway.

Chen, John and K. Vijay-Shanker. 2000. Automated Extraction of TAGs from the Penn Treebank. In *Proc. of the 6th International Workshop on Parsing Technologies (IWPT-2000), Italy*.

Chiang, David. 2000. Statistical Parsing with an Automatically-Extracted Tree Adjoining Grammar. In *Proc. of ACL-2000*.

Clark, Stephen. 2002. Supertagging for combinatory categorial grammar. In *Proceedings of the 6th International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6)*, pages 19–24, Venice, Italy.

Clark, Stephen and James R. Curran. 2004. The importance of supertagging for wide-coverage ccg parsing. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING-04)*, pages 282–288, Geneva, Switzerland.

Collins, Michael and Yoram Singer. 1999. Unsupervised Models for Named Entity Classification. In *In Proc. of WVLC/EMNLP-99*, pages 100–110.

Dasgupta, Sanjoy, Michael Littman, and David McAllester. 2002. PAC generalization bounds for co-training. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA. MIT Press.

Goldman, Sally and Yan Zhou. 2000. Enhancing supervised learning with unlabeled data. In *Proceedings of ICML'2000*, Stanford University, June 29–July 2.

Hwa, Rebecca, Miles Osborne, Anoop Sarkar, and Mark Steedman. 2003. Corrected co-training for statistical parsers. In *In Proceedings of the ICML Workshop on The Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining at the 20th International Conference on Machine Learning (ICML-2003)*, Washington DC, August 21–24.

Joshi, A. and Y. Schabes. 1991. Tree adjoining grammars and lexicalized grammars. In M. Nivat and A. Podelski, editors, *Tree automata and languages*. North-Holland.

Joshi, A. K. and Y. Schabes. 1992. Tree-adjoining grammar and lexicalized grammars. In M. Nivat and A. Podelski, editors, *Tree automata and languages*. Elsevier Science, pages 409–431.

Joshi, Aravind K. 2000. Some aspects of syntactic and semantic complexity and underspecification. Talk given at *Syntactic and Semantic Complexity in Natural Language Processing Systems, Workshop at ANLP-NAACL 2000, Seattle*, May.

Joshi, Aravind K., L. Levy, and M. Takahashi. 1975. Tree Adjunct Grammars. *Journal of Computer and System Sciences*.

Kay, Martin. 1989. Head driven parsing. In *Proc. of IWPT '89*, pages 52–62, Pittsburgh, PA.

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

Lavelli, Alberto and Giorgio Satta. 1991. Bidirectional parsing of Lexicalized Tree Adjoining Grammars. In *Proc. 5th EACL*, Berlin, Germany, April.

Marcken, C. de. 1995. Lexical heads, phrase structure and the induction of grammar. In D. Yarowsky and K. Church, editors, *Proceedings of the Third Workshop of Very Large Corpora*, pages 14–26, MIT, Cambridge, MA.

Marcus, M., B. Santorini, and M. Marcinkiewiecz. 1993. Building a large annotated corpus of english. *Computational Linguistics*, 19(2):313–330.

Nigam, Kamal and Rayid Ghani. 2000. Analyzing the effectiveness and applicability of co-training. In *Proc. of Ninth International Conference on Information and Knowledge (CIKM-2000)*.

Oepen, Stephan and John Carroll. 2000. Ambiguity packing in constraint-based parsing – practical results. In *Proceedings of the 1st Meeting of the North American ACL, NAACL-2000*, Seattle, Washington, Apr 29 – May 4.

Ratnaparkhi, A. 1996. A Maximum Entropy Part-Of-Speech Tagger. In *Proc. of the Empirical Methods in Natural Language Processing Conference*, University of Pennsylvania.

Sarkar, Anoop. 2000. Practical experiments in parsing using tree adjoining grammars. In *Proceedings of the Fifth Workshop on Tree Adjoining Grammars*, Paris, France, May 25–27.

Sarkar, Anoop. 2001. Applying co-training methods to statistical parsing. In *Proceedings of NAACL 2001*, Pittsburgh, PA, June.

Sarkar, Anoop, Fei Xia, and Aravind Joshi. 2000. Some experiments on indicators of parsing complexity for lexicalized grammars. In *In Efficiency in Large-Scale Parsing Systems Workshop held at COLING 2000*, Luxembourg, Aug 5.

XTAG-Group, The. 1998. A Lexicalized Tree Adjoining Grammar for English. Technical Report IRCS 98-18, University of Pennsylvania.

Schabes, Y. 1992. Stochastic lexicalized tree-adjoining grammars. In *Proc. of COLING '92*, volume 2, pages 426–432, Nantes, France.

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

Schabes, Yves and Stuart Shieber. 1994. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 20(1):91–124.

Sikkel, Klaas. 1997. *Parsing Schemata*. EATCS Series. Springer-Verlag.

Srinivas, B. 1997. Performance Evaluation of Supertagging for Partial Parsing. In *Proceedings of Fifth International Workshop on Parsing Technology*, Boston, USA, September.

Srinivas, B. and Aravind Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2).

Steedman, Mark, Rebecca Hwa, Stephen Clark, Miles Osborne, Anoop Sarkar, Julia Hockenmaier, Paul Ruhlen, Steve Baker, and Jeremiah Crim. 2003a. Example selection for bootstrapping statistical parsers. In *In the Human Language Technology Conference and the 4th Meeting of the North American Association for Computational Linguistics: HLT-NAACL 2003*, Edmonton, Canada, May 27–June 1.

Steedman, Mark, Miles Osborne, Anoop Sarkar, Stephen Clark, Rebecca Hwa, Julia Hockenmaier, Paul Ruhlen, Steve Baker, and Jeremiah Crim. 2003b. Bootstrapping statistical parsers from small datasets. In *In the 11th Conference of the European Association for Computational Linguistics: EACL 2003*, Budapest, Hungary, April 12–17.

van Noord, Gertjan. 1994. Head-corner parsing for TAG. *Computational Intelligence*, 10(4).

Vijay-Shanker, K. and D. J. Weir. 1993. The use of shared forests in TAG parsing. In *Proc of 6th Meeting of the EACL*, pages 384–393, Utrecht, The Netherlands.

Xia, Fei. 1999. Extracting tree adjoining grammars from bracketed corpora. In *Proc. of NLPRS-99*, Beijing, China.

Xia, Fei and Martha Palmer. 2000. Evaluating the Coverage of LTAGs on Annotated Corpora. In *Proceedings of LREC satellite workshop Using Evaluation within HLT Programs: Results and Trends*.