SuperTagging and Full Parsing

Alexis Nasr
Lattice
Université Paris 7
Paris, France
alexis.nasr@linguist.jussieu.fr

Owen Rambow
Department of Computer Science
Columbia University
New York, NY, USA
rambow@cs.columbia.edu

Abstract

We investigate an approach to parsing in which lexical information is used only in a first phase, supertagging, in which lexical syntactic properties are determined without building structure. In the second phase, the best parse tree is determined without using lexical information. We investigate different probabilistic models for adjunction, and we show that, assuming hypothetically perfect performance in the first phase, the error rate on dependency arc attachment can be reduced to 2.3% using a full chart parser. This is an improvement of about 50% over previously reported results using a simple heuristic parser.

1 Introduction

Over the last ten years, there has been a great increase in the performance of parsers. Current parsers use the notion of lexical head when generating phrase structure parses, and use bilexical dependencies – probabilities that one particular head depends on another – to guide the parser. Current parsers achieve an score of about 90% when measuring just the accuracy of choosing these dependencies (Collins, 1997; Chiang, 2000; Clark et al., 2002; Hockenmaier and Steedman, 2002). Interestingly, the choice of formalism (headed CFG, TAG, or CCG) does not greatly change the parsers' accuracy, presumably because in all approaches, the underlying information is the same – word-word dependencies.

Supertagging followed by "lightweight" parsing has been proposed as an alternative to full parsing. The idea behind supertagging (Bangalore and Joshi, 1999) is to extend the notion of "tag" from a part of speech or a part of speech including morphological information to a tag that represents rich syntactic information as well, in particular active valency including subcategorization (who can/must be my dependents?), passive valency (who can be my governor?), and notions specific to particular parts of speech, such as voice. If words in a string can be tagged with this rich syntactic information in a supertag, then, Bangalore and Joshi (1999) claim, the remaining step of determining the actual syntactic structure is trivial. They propose a "lightweight dependency parser" (LDA) which is a heuristically-driven, very simple program that creates a dependency structure from the sequence of supertags. It uses no information gleaned from corpora at all, and performs with an (unlabeled) accuracy of about 95%, given the correct supertag.

The question arises how much better we can do if we use a more sophisticated way of determining the parse from the supertags, such as a chart parser. Of course, we do not want to give up the notion of a parsing stage which is relatively simple. In this paper, we extend the parsing stage by using a chart parser and probabilistic models, but we use only models that relate supertags to each other. In fact, such models are also used during supertagging, except that in supertagging, the only relation between supertags we are interested in modeling probabilistically is linear precedence, while for parsing we will use structural dependency as well. Thus, our approach conservatively extends the supertagging-and-LDA approach, and remains quite different from the current work on parsing based on bilexical probability models following (Collins, 1997). A secondary question we investigate in this paper is the issue of how best to model multiple adjunctions at a same node.

The paper is structured as follows. In Section 2, we provide some more motivation for this work. We present our formalization of TAG and discuss how to derive such a grammar formalized in that way from a corpus in Section 3. We present the parser in Section 4. In Section 5 we discuss three different ways in which we we estimate parameters for the statistical models. In Section 6, we present two baselines, and our main results. We discuss related work in Section 7, and conclude in Section 8.

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 56-63.

2 Motivation

In this paper, we assume we have the correct supertag and we investigate the quality of the resulting parse. The state of the art in supertagging is currently in the 80%-85% range, depending on the grammar (see for example (Chen, 2001)). Thus, the task we set out to examine is not a realistic "real-world" task, and the question arises why we should be interested in the question at all. In this section, we try to motivate our research agenda by first arguing why supertag-based non-lexical parsing is interesting, and then by arguing why we need to show it is feasible.

2.1 Supertag-Based Parsing Is Interesting

There are several reasons to investigate supertag-based parsing. The main point is that the models involved are potentially simpler than those in bilexical parsing: no bilexical structural information is needed for deriving the parsing model. This holds the promise that when porting a supertagger-based parser to a new domain, a non-lexical structural model can be reused from a previous domain, and only a supertagged corpus in the new domain is needed (to train the supertagger), not a structurally annotated corpus.

Furthermore, this approach uses an explicit lexicalized grammar. As a consequence, when porting a parser to a new domain, learned parser preferences in the supertagger can be overridden explicitly for domain-idiosyncratic words before the parse happens. This overriding can happen through manually written or learned rules. By way of anecdotal example, in a recent application of the parser of Collins (1997) in which the WSJ-trained parser was applied to rather different text, sentences such as *John put the book on the table* were mostly analyzed with the PP attached to the noun, not the verb (as was always required in that domain). In the application, this had to be fixed by writing special post-processing code to rearrange the output of the parser; in our approach, we could simply state that *put* should always have a PP argument *before* the parse, and correct any output of the supertagger using simple hand-written rules.

And finally, we point out that is is a different approach from the dominant bilexical one, and it is always worthwhile to pursue new approaches, especially as the performance of the bilexical parsers seems to be plateauing. In fact recent work has questioned to what extent bilexical parsers even profit from bilexical information that they use (Gildea, 2001; Klein and Manning, 2003).

2.2 But Is Supertag-Based Parsing Feasible?

Bangalore and Joshi (1999) claim that supertagging is "almost parsing". What this means is that the syntactic information provided by supertags is so rich that there is little structural ambiguity left and the parse is almost entirely determined by the supertags. In fact, since the supertags determine both active and passive valency, the only remaining ambiguity is related to attachment of trees to nodes with the same label; for example, the standard PP-attachment ambiguity of *see a man with a telescope* is resolved in the supertags, as the tag for *with* specified adjunction to a VP or an NP. The remaining issues of structural ambiguity which are not resolved by supertags include conjunctions, noun-noun compounds (which however are not given meaningful analyses in the PTB), attachment of adjuncts in sentences with several clauses (such as *John told Mary to leave today*), and so on.

There is thus both a practical and a theoretical interest in knowing how much ambiguity remains after supertagging, or, put differently, to what extent supertagging is in fact "almost parsing". Practically, the performance of a parser with correct supertags as input gives us an upper bound on supertag-based parsing. The current figure of 95% (using the heuristic LDA) may seem a bit low as an upper bound. Theoretically, it is interesting to know to what extent, in a corpus, syntactic structure is disambiguated by specifying both active and passive valency of words.

3 Representing a TAG as a Set of FSMs

For the purpose of our parser, we represent a Tree Adjoining Grammar as a set of finite-state machines (FSMs). The FSMs form a (lexicalized) Recursive Transition Network (RTN). To extract an RTN from the Penn Treebank (PTB), we first extract a TAG, and then convert it to an RTN. This first step does not represent the research reported in this paper, and we describe it only for the sake of clarity. We use the approach of (Chen, 2001) (which is similar to (Xia et al., 2000) and (Chiang, 2000)). We use sections 02 to 21 of the Penn Treebank. However, we optimize the head percolation in the grammar extraction module to create meaningful dependency structures, rather than (for example) maximally simple elementary tree structures. For example, we include long-distance dependencies (*wh*-movement, relativization) in elementary trees, we distinguish passive transitives without *by*-phrase from active intransitives, and we include strongly governed prepositions (as determined in the PTB annotation, including passive *by*-phrases) in elementary verbal trees. Generally, function words such as auxiliaries or determiners are dependents of the lexical head,[1] conjunctions (including punctuation functioning as conjunction) are dependent on the first conjunct and take the second conjunct as their argument, and conjunction chains are represented as right-branching rather than flat.

[1]This is a linguistic choice and not forced by the formalism or the PTB. We prefer this representation as the resulting dependency tree is closer to predicate-argument structure.

In the second step, we directly compile a set of FSMs which are used by the parser. To derive a set of FSMs from a TAG, we do a depth-first traversal of each elementary tree in the grammar (but excluding the root and foot nodes of adjunct auxiliary trees) to obtain a sequence of nonterminal nodes. As usual, the elementary trees are tree schemas, with positions for the lexical heads. Substitution nodes are represented by obligatory transitions, adjunction by optional transitions (self-loops). (Note that in this paper, we assume adjunction as defined by Schabes and Shieber (1994).) Each node becomes two states of the FSM, one state representing the node on the downward traversal on the left side (the **left node state**), the other representing the state on the upward traversal, on the right side (the **right node state**). For leaf nodes, the two states immediately follow one another. The states are linearly connected with ϵ-transitions, with the left node state of the root node the start state, and its right node state the final state (except for predicative auxiliary trees – see below). We give a sample grammar in Figure 1 and the result of converting one of its trees to an FSM in Figure 2.

For each pair of adjacent states representing a substitution node, we add transitions between them labeled with the names of the trees that can substitute there. For the lexical head, we add a transition on that head. For footnodes of predicative auxiliary trees which are left auxiliary trees (in the sense of Schabes and Waters (1995), i.e., all nonempty frontier nodes are to the left of the footnode), we take the left node state as the final state. Finally, in the basic model in which adjunctions are modeled as independent, we proceed as follows for non-leaf nodes. (In Section 5, we will see two other models that treat non-leaf nodes in a more complex manner.) To each non-leaf state, we add one self loop transition for each tree in the grammar that can adjoin at that state from the specified direction (i.e., for a state representing a node on the downward traversal, the auxiliary tree must adjoin from the left), labeled with the tree name. There are no other types of leaf nodes since we do not traverse the passive valency structure of adjunct auxiliary tees. The result of this phase of the conversion is a set of FSMs, one per elementary tree of the grammar, whose transitions refer to other FSMs.

Note that the treatment of footnodes makes it impossible to deal with trees that have terminal, substitution or active adjunction nodes on both sides of a footnode. It is this situation (iterated, of course) that makes TAG formally more powerful than CFG; in linguistic uses, it is very rare, and no such trees are extracted from the PTB.[2] As a result, the grammar is weakly equivalent to a CFG. In fact, the construction treats a TAG as if were a Tree Insertion Grammar (TIG, Schabes and Waters (1995)), or rather, it coerces a TAG to be a TIG: during the traversal, both terminal nodes and nonterminal (i.e., substitution) nodes between the footnode and the root node are ignored (because the traversal stops at the footnode), thus imposing the constraint that the trees may not be wrapping trees and that no further adjunction may occur to the right of the spine in a left auxiliary tree.

4 Parsing with FSMs

The parsing algorithm is a simple extension of the chart parsing algorithm for CFG. The difference is in the use of finite state machines in the items in the chart. In the following, we will call t-**FSM** an FSM M if it is derived from tree t in the original TAG (or TIG) grammar G. If T is the parse table for input sentence W and GDG G, then $T_{i,j}$ contains (M, q) where M is a t-FSM, and q is one of the final states of M, iff we have a complete derivation of substring $w_i \cdots w_j$ in G such that the root of the derivation tree is labeled t.

Before starting the parse, we create a tailored grammar by selecting those trees associated with the words in the input sentence, and substituting the actual words for the positions of the lexical head. (Note that the crucial issue is how to associate trees with words in a sentence; in this paper, we assume that the correct tree is used.)

Initialization: We start by adding, for each i, $1 \leq i \leq n$, w_i to $T_{i,i}$.

Completion: If $T_{i,j}$ contains either the input symbol w or an item (M, q) such that q is a final state of M, and M is a t-FSM, then add to $T_{i,j}$ all (M', q') such that M' is a FSM which transitions from a start state to state q' on input w or t.

Add a single backpointer from (M', q') in $T_{i,j}$ to (M, q) or w in $T_{i,j}$.

Scanning: If (M_1, q_1) is in $T_{i,k}$, and $T_{k+1,j}$ contains either the input symbol w or the item (M_2, q_2) where q_2 is a final state and M_2 is a t-FSM, then add (M_1, q) to $T_{i,j}$ (if not already present) if M_1 transitions from q_1 to q on either w or t.

Add a double backpointer from (M_1, q) in $T_{i,j}$ to (M_1, q_1) in $T_{i,k}$ (left backpointer) and to either w or (M_2, q_2) in $T_{k+1,j}$ (right backpointer).

Note that because we are using a dependency grammar, each scanning step corresponds to one attachment of a lexical head to another. At the end of the parsing process, a packed parse forest has been built. The nonterminal nodes are labeled with pairs (M, q) where M is an FSM and q a state of this FSM. Obtaining the dependency trees from the packed parse forest is performed

[2] Our construction cannot handle Dutch cross-serial dependencies (not surprisingly), but it can convert the TAG analysis of *wh*-movement in English and similar languages, because the predicative auxiliary verbal trees do not have terminal or substitution nodes on both sides of the foot node.

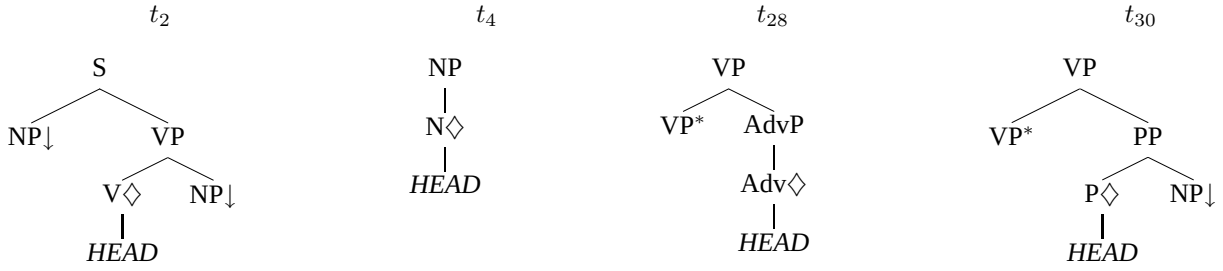

Figure 1: Sample small grammar: trees for a transitive verb, a nominal argument, and two VP adjuncts from the right

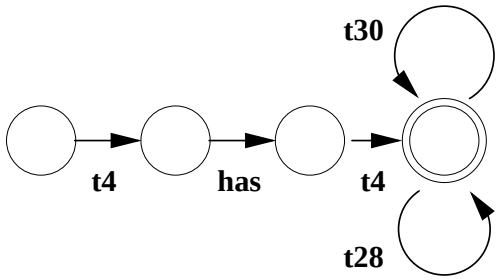

Figure 2: FSM derived according to Model 1 for tree t_2 from the grammar in Figure 1, instantiated for the verb *has*

in two stages. In a first stage, a forest of binary phrase-structure trees is obtained from the packed forest and in a second stage, each phrase-structure tree is transformed into a dependency tree.

5 Probabilistic Models

The parser introduced in Section 4 associates to a supertag sequence $S = S_1 \dots S_n$ one or several analyses. Each analysis $\mathcal{A}$ can be seen as a set of $n-1$ attachment operations and the selection of one supertag token as the root of the analysis (the single supertag that is not attached in another supertag). For the sake of uniformity, we will consider the selection of the root as a special kind of attachment, $\mathcal{A}$ is therefore of cardinality n. In the following, $LEFT(x, y)$ (respect. $RIGHT(x, y)$) denotes the set of attachments that occurred on the left (respect. right) side of node y of supertag x. For an attachment operation A, $O(A)$ returns its type (adjunction, substitution, root). *Root* designates the unique event in $\mathcal{A}$ that selects the root.

From a probabilistic point of view, each attachment operation is considered as an event and an analysis $\mathcal{A}$ as the joint event $A_1, \dots, A_n$. A large range of different models can be used to compute such a joint probability, from the simplest which considers that all events are independent to the model that considers that they are all dependent. The three models that we descibe in this section vary in the way they model multi-adjunction (when several auxiliary trees are attached to a single node from the same direction). The reason to focus on this phenomenon comes from the fact that it is precisely at this level that much of the structural ambiguity occurs. For example, in a sentence containing three or more conjoined NPs (such as *dogs, hamsters, cats, and bats*), there is massive ambiguity of attachment, as each conjunction can attach to any of the preceding nouns. However, only one structure (in our corpus, the right-branching one) is correct. Thus, a precise model is needed. The three models described below consider that substitution operations are independent of all the other attachments that make up an analysis. The general model is therefore:

$$\begin{aligned} P(\mathcal{A}) &= P(Root) \\ &\times \prod_{A \in \mathcal{A} | O(A) = subst} P(A) \\ &\times \prod_{s \in S, i \in nodes(s)} P(LEFT(s,i)) \\ &\times \prod_{s \in S, i \in nodes(s)} P(RIGHT(s,i)) \end{aligned}$$

This basically follows (Resnik, 1992; Schabes, 1992). The models we discuss here differ in how to compute the terms $P(RIGHT(s,i))$ and $P(LEFT(s,i))$.

The probability of each attachment is estimated by maximum likelihood (the counts are obtained in the same step as the grammar extraction), and are added to the corresponding transition in the governor's automaton as its weight. When the probabilistic model associates different

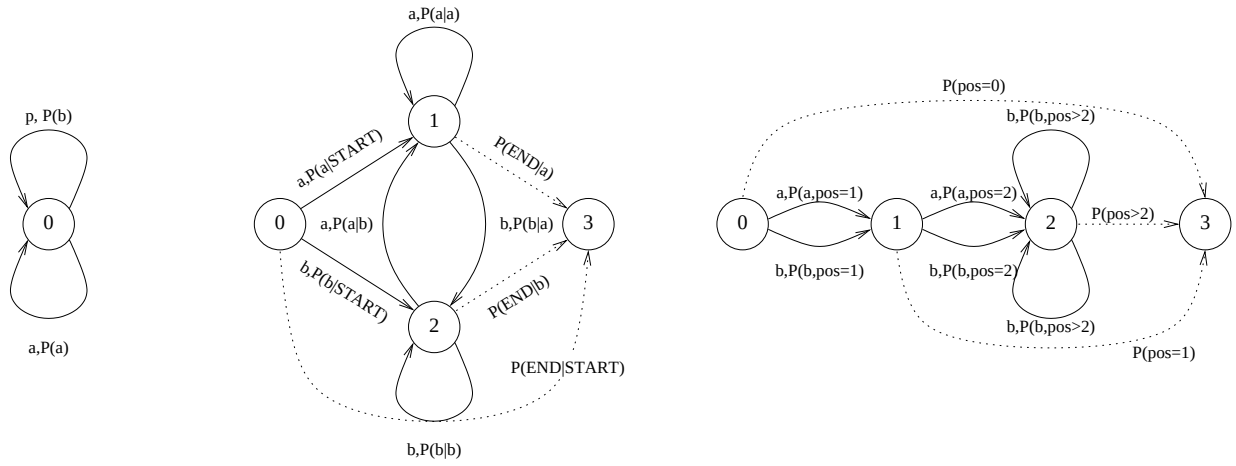

Figure 3: Three models of adjunction; these correspond to the last node of the FSM in Figure 2, with the model on the left exactly as shown in Figure 2; here, a represents t_{30} of Figure 2 and b, t_{28}

probabilities to attachments that were not distinguished in an automaton, the structure of the latter will be changed in order to account for this difference. This change in the structure will, of course, leave unchanged the language recognized by the automaton . The three models for adjunction will be illustrated on a simple example where two supertags a and b are candidate for adjunction at a given node. In the following models, we estimate parameters from the corpus obtained by running the TAG extraction algorithm over the PTB training corpus. We can then easily count the relevant events.

5.1 Model 1: Independent Adjunctions

In this model, an adjunction on one node is considered independent from the other adjunctions that can take place on the same node. The probability of each adjunction depends on the dependent supertag, on the governor supertag, and on the node of the governor supertag at which the attachment takes place. However, it is independent of the order of the attachment. The model does therefore not distinguish between attachments that only differ in their order. This model corresponds to the left part of figure 3, the attachment of an a, for example, does not depend on what was attached before and how many attachment took place. For example, the probability of the sequence $abab$ being adjoined is modeled as follows (we use here and subsequently a simplified notation where $P(a)$ designates the adjunction of a at the relevant node in the relevant tree):

$$P(abab) = P(a)P(b)P(a)P(b)$$

5.2 Model 2: Positional Model

This model adds to the first one the knowledge of the *order* of an attachment. But when modeling the probability that supertag a attaches on a given node at order i, it does not take into account the attachments that happened for $order < i$. Such models also add a new parameter which is the maximum number of attachment that are distinguished. The graphical representation of the model as a finite state automaton, as it appears to the right in Figure 3, gives an intuitive account of the nature of the model. It is made of a series of transitions between consecutive pairs of nodes. The first "bundle" of transitions models the first attachment on the node, the second bundle, the second attachment, and so on, until the maximum number of attachments is reached. This limit on the number of attachments concerns only the probabilistic part of the automaton, more attachment can occur on this node, but their probabilities will not be distinguished. These attachments correspond to the loops on state 2 of the automaton. ϵ-transitions allow the attachments to stop at any moment by transitioning to state 3. (The ϵ-transitions are shown as dotted lines for reading convenience, they are formally regular transitions in the FSM.) Under Model 2, the probability of the sequence $abab$ being adjoined is:

$$\begin{aligned} P(abab) &= P(a, pos = 1) \\ &\times P(b, pos = 2) \\ &\times P(a, pos > 2) \\ &\times P(b, pos > 2) \end{aligned}$$

5.3 Model 3: N-Gram Model

The previous model takes into account the order of an attachment and disregards the nature of the attachments that happened before (or after) a given attachment. The model described here is, in a sense, complementary to

the previous one since it takes into account, in the probability of an attachment, the nature of the attachment that occurred just before and ignores the order of the current attachment. The probability of a series of attachments on the same side of the same node will be computed by an order-1 Markov chain, represented as a finite state automaton in the central part of Figure 3. The transitions with probabilities $P(x|START)$ (respect. $P(END|x)$) correspond to the occurrence of supertag x as the first (respect. the last) attachment at this node and the transition with probability $P(END|START)$ corresponds to the null adjunction (the probability that no adjunction occurs at a node). The probability of the sequence $abab$ being adjoined is now:

$$\begin{aligned} P(abab) &= P(a|START) \\ &\times P(b|a) \\ &\times P(a|b) \\ &\times P(b|a) \\ &\times P(END|b) \end{aligned}$$

5.4 Finding the n-best parses

We extend our parser by augmenting entries in the parse table with probabilities. As usual, only the highest probability is retained for a given analysis. The algorithm for extracting parses is augmented to choose the best parse (or n-best parses) in the usual manner. Note that the different models discussed in this section only affect the manner in which the TAG grammar extracted from the corpus is converted to an FSM; the parsing algorithm (and code) is always the same.

6 Results

In this study, we are interested in exploring how parsing performs in the presence of the correct supertag. As a result, in the following, we report on data which has been correctly supertagged. We used Sections 02 to 21 of the Penn Treebank for training, the first 800 sentences of Section 00 for development, and Section 23 for testing only. The figures we report are accuracy figures: we evaluate how many dependency relations have been found. The root node is considered to have a special dependency relation. There is no need to report recall and precision, as each sentence always has a number of dependency relations which is equal to the number of words. In the evaluation, we disregard true (non-conjunction) punctuation. The figures for the LDA are obtained by using the LDA as developed previously by Bangalore Srinivas, but using the same grammar we used for the full parser. Note that none of the numbers reported in this section can be directly compared to any numbers reported elsewhere, as this task differs from the tasks discussed in other research on parsing.

We use two different baselines. First, we use the performance of the LDA of (Bangalore and Joshi, 1999). The performance of the LDA on Section 00 is about 94.3%, on Section 23 95.1%. Second, we use the full chart parser, but randomly choose a parse from the parse forest. This baseline measures to what extent using a probabilistic model in the chart parser actually helps. The performance of this baseline is 94.7% on Section 00, 94.6% on Section 23. As we can see, the supertags provide sufficient information to result in high baselines. The results are summarized in Figure 4.

There are several clear conclusions to be drawn from Figure 4. First, a full parse has advantages over a heuristic parse, as even a random choice of a tree from the parse forest in the chart (i.e., without use of a probabilistic model) performs nearly as well as the heuristic LDA. Second, the use of even a simple probabilistic model using no lexical probabilities at all, and modeling adjunctions as entirely independent, reduces the error rate over the non-probabilistic baseline by 22.8%, to 4.04%. Third, the modeling of multiple adjunctions at one node as independent is not optimal, and two different models can further reduce the error rate substantially. Specifically, we can increase the error reduction to 53.0% by modeling the first adjunction (from left to right) separately from all subsequent ones. However, presumably due to sparseness of data, there is no major advantage to using more than one position (and modeling the first and second adjunction separately). Furthermore, switching to the n-gram model in which an adjunction is conditioned on the previously adjoined supertag as well as the governing supertag, the error reduction is further increased slightly to 56.6%, with an error rate of 2.27%. This is the best result obtained on the development corpus.

7 Related Work

We are not aware of any other work that directly investigates the extent to which supertagging determines parsing. Chiang (2000) also parses with an automatically extracted TIG, but unlike our approach, he uses standard TAG/TIG parsing techniques (i.e., he reconstructs the derived tree in the chart, not the derivation tree). Rogers (1994) proposes a different context-free variant, "regular-form TAG". The set of regular-form TAGs is a superset of the set of TIGs, and our construction cannot capture the added expressive power of regular-form TAG. Our conversion to FSMs is very similar to that of Evans and Weir (1997). One important difference is that they model TAG, while we model TIG. Another difference is that they use FSMs to encode the sequence of actions that need to be taken during a standard TAG parse (i.e., reconstructing the derived tree), while we encode

Method	Accuracy on Sec 00	Accuracy on Sec 23
Baseline: LDA	94.35%	95.14%
Baseline: full parse with random choice	94.73%	94.69%
Model 1 (Independent Adjunction)	95.96%	
Model 2 (Positional Model): 1 position	97.54%	
Model 2 (Positional Model: 2 position	97.49%	
Model 2 (Positional Model: 3 position	97.57%	
Model 3 (N-Gram Model), using Supertag	97.73%	97.61%
Model 3 (N-Gram Model), using Category	97.29%	

Figure 4: Results (accuracy) for different models using the Gold-Standard supertag on development corpus (Section 00, first 800 sentences) with add-0.001 smoothing, and for the best performing model as well as the baselines on the test corpus (Section 23)

the active valency of the lexical head in the FSM. A result, in retrieving the derivation tree, each item in the parse tree corresponds to an attachment of one word to another, and there are fewer items. Furthermore, our FSMs are built left-to-right, while Evans and Weir only explore FSMs constructed bottom-up from the lexical anchor of the tree (not unlike (Eisner, 2000)). As a result, we can perform a strict left-to-right parse, which is not straightforwardly possible in standard TAG parsing using FSMs.

Our parsing algorithm is similar to the work of Alshawi et al. (2000). They use cascaded head automata to derive dependency trees, but leave the nature of the cascading under-formalized. Eisner (2000) provides a formalization of a system that uses two different automata to generate left and right children of a head. His formalism is very close to the one we present, but we use a single automaton. Also, the relation to an independently proposed syntactic formalism such as TAG is less obvious.

In related work (Rambow et al., 2002), we have used the same automata constructed from an extracted TAG for parsing, but instead of using them in a chart parser, we have used them to construct a single large FSM that produces a dependency tree. Needless to say, the number of embeddings allowed by such an approach is limited.

8 Conclusion

We have provided further evidence for the claim of Bangalore and Joshi (1999) that supertagging is "almost parsing", and we have quantified the "almost" to be 97.7%.[3] This figure represents the dependency accuracy that can be obtained when the input is represented as a sequence of supertags, with no lexical information used in the parse (and hence not in the training of the parser, either). This shows that an architecture is viable in which *all* information related to the specific lexemes is assigned in a first pass before structure is constructed, and structure is constructed only in a second pass in which *no* lexical information is used (other than the lexical emit probability for supertags). This result motivates further research into supertagging accuracy. If supertagging accuracy is improved, a lightweight parser in conjunction with supertagging may perform as well as a full bilexical parser, or even better. Furthermore, for certain applications, a lightweight parser may be appealing because only the supertagger needs to be retrained which can be done with less effort. Finally, the explicit and declarative nature of the grammar used makes it easy to write hand-written rules to override the supertagger in cases in which the application designer wishes to correct a systematic parser error.

[3] We note that this figure holds for the particular grammar that we used; other grammars may result in different figures.

References

Hiyan Alshawi, Srinivas Bangalore, and Shona Douglas. 2000. Learning dependency translation models as collections of finite-state head transducers. *cl*, 26(1):45–60.

Srinivas Bangalore and Aravind Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–266.

John Chen. 2001. *Towards Efficient Statistical Parsing Using Lexicalized Grammatical Information*. Ph.D. thesis, University of Delaware.

David Chiang. 2000. Statistical parsing with an automatically-extracted tree adjoining grammar. In *38th Meeting of the Association for Computational Linguistics (ACL'00)*, pages 456–463, Hong Kong, China.

Stephen Clark, Julia Hockenmaier, and Mark Steedman. 2002. Building deep dependency structures with a wide-coverage ccg parser. In *acl02*, pages 327–334.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, Madrid, Spain, July.

Jason Eisner. 2000. Bilexical grammars and their cubic-time parsing algorithms. In Harry C. Bunt and Anton Nijholt, editors, *New Developments in Natural Language Parsing*. Kluwer Academic Publishers.

Roger Evans and David Weir. 1997. Automaton-based parsing for lexicalized grammars. In *5th International Workshop on Parsing Technologies (IWPT97)*, pages 66–76.

Daniel Gildea. 2001. Corpus variation and parser performance. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing (EMNLP01)*, pages 167–202, Pittsburgh, PA.

Julia Hockenmaier and Mark Steedman. 2002. Generative models for statistical parsing with combinatory categorial grammar. In *acl02*, pages 335–342.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *41st Meeting of the Association for Computational Linguistics (ACL'03)*.

Owen Rambow, Srinivas Bangalore, Tahir Butt, Alexis Nasr, and Richard Sproat. 2002. Creating a finite-state parser with application semantics. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002)*, Taipei, Republic of China.

Philip Resnik. 1992. Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *Proceedings of the Fourteenth International Conference on Computational Linguistics (COLING '92)*, Nantes, France, July.

James Rogers. 1994. Capturing cfls with Tree Adjoining Grammars. In *32nd Meeting of the Association for Computational Linguistics (ACL'94)*. ACL.

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

Yves Schabes and Richard C. Waters. 1995. Tree Insertion Grammar: A cubic-time, parsable formalism that lexicalizes Context-Free Grammar without changing the trees produced. *Computational Linguistics*, 21(4):479–514.

Yves Schabes. 1992. Stochastic lexicalized tree-adjoining grammars. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING'92)*.

Fei Xia, Martha Palmer, and Aravind Joshi. 2000. A uniform method of grammar extraction and its applications. In *Proc. of the EMNLP 2000*, Hong Kong.