Nonlexical Chart Parsing for TAG

Alexis Nasr and Owen Rambow

Abstract

Bangalore and Joshi (1999) investigate supertagging as "almost parsing". In this paper we explore this claim further by replacing their Lightweight Dependency Analyzer with a nonlexical probabilistic chart parser. Our approach is still in the spirit of their work in the sense that lexical information is only used during supertagging; the parser and its probabilistic model only see supertags.

1 Introduction: Supertags and Parsing

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% to 92% when measuring just the accuracy of choosing these dependencies (Collins, 1997; Chiang, 2003; Charniak, 2000; Clark et al., 2002; Hockenmaier and Steedman, 2002), also see (Yamada and Matsumoto, 2003). 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, with various types of backoff.

An alternative approach has been proposed in the literature: supertagging followed by "lightweight" 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 for verbs. If words in a string can be tagged with this rich syntactic information, 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 tagged string of words. It uses no information gleaned from corpora at all, and performs with an (unlabeled) accuracy of about 95%, given the correct supertag. While the supertagging only requires a notion of syntactically relevant features, the stage of determining a syntactic structure requires a grammar that uses these syntactically relevant features; Bangalore and Joshi (1999) use Tree Adjoining Grammar (TAG) as a bridge between the features and the actual syntactic combinations. The approach does not rely on TAG, however,[1] and any lexicalized[2] (or lexicalist, in a wider sense) grammar formalism could be used.

There are several reasons why it is worthwhile pursuing an approach to parsing in which the sentence is first supertagged and then analyzed syntactically. The main point is that the models involved are potentially simpler than those in bilexical parsing. More precisely, the probabilistic models of the parser define a smaller number of parameters and are therefore less prone to data sparseness.[3] In particular, no bilexical or monolexical information is used in the parsing model. This holds the promise that when porting a supertagger-based parser to a new domain, a nonlexical 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. For example, suppose that an application of a parser such as that of Collins (1997) trained on a news corpus is applied to rather different genre, and that sentences such as *John put the book on the table*, unseen in training, are mostly analyzed with the PP attached to the noun, not to the verb (as is always required in the new domain). In the application, this would need to be fixed by writing special post-processing code to rearrange the output of the parser since there is no way to change the parser's behavior short of retraining it; in our approach, we could simply state that *put* should always (or with greatest probability) have a PP argument. And finally, we point out that 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 this paper, we follow the line of research started by Bangalore and Joshi (1999). Like their work, we in fact use a less powerful tree-rewriting formalism than TAG, namely TIG. We depart from their work in two ways.

- First, we use a chart parser with a statistical model derived from a corpus, which however is entirely nonlexical and just uses the supertags, not the words or lexemes found in the corpus. As we will see, when supertagging

[1]The morpheme *tag* in *supertagging* is of course not the TAG of Tree Adjoining Grammar, but the English word that means 'label'.

[2]We use the term *lexicalized* to refer to a grammar formalism such that each of its elementary structures is associated with one lexical item (i.e., terminal symbol in the sense of formal grammar theory); we use the terms *nonlexical*, *monolexical* and *bilexical* to refer to probabilistic models that take no, one or two lexical heads into account, respectively. The more common terms in the literature for these notions, *unlexicalized*, *lexicalized*, and *bilexical* are confusing and inconsistent.

[3]Gildea (2001) shows that in fact the bilexical dependencies contribute little to performance of bilexical parsers, with lexical-structural dependencies being more important. While that finding is compatible with our results, we structure the parsing process in a completely different manner.

is combined with a full chart parser, the dependency accuracy is about 98% when given correct (gold-standard) supertags. We thus cut the error rate of the heuristic LDA by more than half. Our approach is still in the spirit of supertagging, as the parser has no access to lexical information, only to information about the supertags. We differ from Bangalore and Joshi (1999) only in that we use, in addition to structural information contained in the supertag, probabilistic information about the relation between supertags, as derived from a corpus.

- Second, we use a grammar extracted automatically from the Penn Treebank (PTB) rather than a hand-crafted one mapped to the corpus (Chen, 2001; Xia et al., 2000; Chiang, 2003). We use a grammar derived from a treebank in order to achiever greater empirical coverage.

The overall goal of this paper is to show that the use of a full probabilistic chart parser can improve on the results of the LDA, while retaining the intuition of localizing all lexical information in the supertagging stage. The specific goal of this paper is to present a nonlexical probabilistic model based on supertags alone; more specifically, we want to investigate different probabilistic models of adjunction at the same node.

We present results using the approach of (Nasr and Rambow, 2004b), but using actual supertagging (while (Nasr and Rambow, 2004b) uses gold-standard supertags). We explore using n-best results from two supertaggers, namely a standard n-gram supertagger, and a maximum entropy tagger which performs better (Bangalore et al., 2005). (Note that the LDA can only take one supertag per word in input, so we are here making use of the chart parser.) We achieve about 85% accuracy on dependency arcs. While this performance is below the state of the art using other methods,[4] we believe that the work reported here can serve as a starting point for more research into this kind of parsing. In particular, in our architecture we can explore supertagging and parsing as separate processes, finding ways to improve the performance of either, or as interdependent processes.

The paper is structured as follows. In Section 2, we discuss related work. We concentrate on two lines of research which are very similar to ours in certain respects: the work on parsing CCG by Clark and Curran (2004), and the work on nonlexical CFG parsing by Klein and Manning (2003). These comparisons further help clarify the goals of our research. We present the underlying formalism in Section 3. In Section 4 we present the parser we use. We discuss the probabilistic models encoded in the automaton representation of the grammar

[4]However, we point out that unlike many other dependency structures, our dependency structure is very "semantic", in that it directly represents predicate-argument structure. We give several examples: in the presence of an auxiliary, the subject still depends on the main verb, as does the auxiliary (which has no dependents); nominal, adjectival, and prepositional predicates are analyzed with the copula as auxiliary (i.e., the subject depends directly on the predicate, as does the auxiliary); a strongly governed preposition (for example, *to* in *give books to Mary*) does not govern a verbal argument (*Mary* in the example), instead the verbal argument depends directly on the verb. Thus our accuracy figures may not be comparable directly to other published results.

in Section 5, and how to extract a grammar which encodes such models from a corpus in Section 6. In Section 7, we present and discuss results. We conclude in Section 8 with a summary of future avenues of research based on the preliminary study presented here.

2 Related Work

The work we describe in this paper is closely related in different ways to the work of Bangalore and Joshi (1999), Clark and Curran (2004), and Klein and Manning (2003). We have discussed the relation to the work of Bangalore and Joshi (1999) in Section 1; we discuss the other two in turn in this section. In addition, there is work on probabilistic TAG parsing which does not use supertagging, see (Sarkar and Joshi, 2003) for a general overview. The advantage of using probabilistic TAG is that the bilexical model can be expressed very naturally. Our (nonlexical) work follows most current probabilistic TAG work in being based on the (very similar) models of Resnik (1992) and Schabes (1992). Of particular relevance to our work is (Chiang, 2003), who, while using a bilexical model, uses a TAG variant similar to the one we use, and also uses a similar horizontal markovization (Section 5.3). We omit a general comparison to the growing literature on non-TAG dependency parsing, and refer to (Nivre, 2006) for a general overview.

2.1 CCG Parsing

Clark and Curran (2004) present a CCG parser which uses a supertagger. It is the first full parser of any kind to successfully use a supertagger in conjunction with a parser. In CCG, the supertags correspond to lexical categories. The corpus used is the CCGbank (Hockenmaier and Steedman, 2002), which fixes the set of lexical categories to about 1,200, of which 400 occur more than 10 times. (Note that for our purposes, the CCGbank should be compared to a particular way of extracting a TAG from the PTB and the resulting annotated corpus, not directly to the PTB.) Clark and Curran (2004) use a maximum-entropy tagger which produces more than one result. The parser (described in (Clark and Curran, 2004)) is a discriminative log-likelihood chart parser, which uses lexical information as well, unlike our parser, for which all lexical information is relegated to the supertagging phase. There are several important results. First, they obtain an oracle parser results of almost 98%, which is comparable to our oracle result, i.e., our result using the gold supertag. (But note that their parser is a bilexical parser while ours is nonlexical.) Second, the performance of their n-best supertagger varies from 96.4% to 98.6%, depending on two parameters for choosing the n (which differs from word to word), which gives an average ambiguity per tag ranging from 1.4 to 3.5. This result is better than that of the supertagger we use, presumably because of their smaller tag set. Finally, using this supertagger in a parser, the best performance (unordered, unlabeled dependency recall/precision on the CCG derivation) is 92.5% (precision) and

91.1% (recall). Again, this performance is better than our parser, presumably because their supertagger performs better.

2.2 Nonlexical CFG Parsing

Our approach is similar to that of Klein and Manning (2003) in that we do not use a bilexical or monolexical probability model. Rather, the generative probabilistic model that we use only models the relation between the elementary structures in the grammar. The second important similarity is that the grammars are extracted from the Penn Treebank, and the probability model is estimated at the same time using a maximum likelihood estimation. The principal difference is that Klein and Manning (2003) use a CFG as their underlying grammar formalism, while we use TAG. This difference has important repercussions: on the one hand, the extracted TAG is much bigger than an extracted CFG, and the lexicon much more ambiguous (i.e., the tagging task is harder). On the other hand, the extended domain of locality of TAG makes the grammar more expressive. As a result, many of the techniques discussed in Klein and Manning (2003), which are aimed at changing the CFG in order to boost performance, are not relevant. Specifically:

- In vertical markovization, ancestor nodes are annotated in nonterminal nodes. In TAG (but not in CFG), each elementary structure contains the entire syntactic projection from the head. This is different from a fixed-length "vertical" history, but captures the same kind of information.
- There is no need for special modeling of unary rules. This is because unary context-free expansions will be part of a larger structure. Furthermore, many of the cases in which Klein and Manning (2003) deal with unary rules are in fact not unary because empty categories are explicitly included in the elementary structures of the grammar. This includes the traces left by *wh*-movement, by argument movement (for passive), and empty arguments (including pro and PRO).
- Many groups of words which receive the same PTB part-of-speech tag are differentiated by their supertag. To take the example discussed in Klein and Manning (2003), demonstrative pronouns and demonstrative determiners have the same tag DT in the PTB, but they have different elementary trees in our grammar (an initial tree and an auxiliary tree, respectively). In general, a TAG subsumes the annotation of preterminals, as no elementary tree has only a preterminal and a terminal symbol.

However, the following techniques are also relevant to nonlexical TAG parsing:

- Horizontal markovization refers to the decomposition of flat rules into a sequence of rules conditioned on the context, and was first proposed by Collins (1997) and subsequently used by Klein and Manning (2003) and by Chiang (2003) for a TAG model, among others. We use horizontal markovization as well (see Section 5.3). However, in our model (as in that of Chiang (2003), but unlike CFG-based models), the adjunction operation allows for explicitly modeling the argument/adjunct distinction, and the markovization only applies to adjuncts, and only to adjuncts adjoined at the same node, not to any expansion of the derived or derivation tree.

- Attachment errors and conjunction scope are problems that also affect our approach, though note that attachment errors include only those that are attachments to the same category (for example, attachment of a VP modifier to a higher or a lower clause), but not the classical NP-VP attachment ambiguities (which are disambiguated by the supertag). The technique proposed by Collins (1997) which Klein and Manning (2003) investigate could also be used in TAG parsing (in appropriately modified form), but we do not investigate this question in this paper.

In summary, we could consider our approach as an alternate form of nonlexical parsing.

3 Generative Dependency Grammar (gdg)

The formalism we use can be presented in several different ways:

- As a generative string-rewriting system, gdg (Generative Dependency Grammar). We present this formalism in detail in (Nasr and Rambow, 2004a).
- As an implementation of Recursive Transition Networks (rtn).
- As a tree-rewriting formalism, namely Tree Insertion Grammar (Schabes and Waters, 1995).

In this paper, we choose the presentation as a rtn, and refer to the cited papers for the other views.

3.1 Informal Definition

A gdg is a set of finite-state automata (fsms) of a particular type, namely *lexicalized automata*. A lexicalized automaton with the anchor (word) m describes all possible dependents of m. Each automaton has a name, which defines not only the part-of-speech of m, but also the active valency of m (i.e., all word classes that can depend on it), as well as their linear order. Thus this name can be thought of as a *supertag* in the sense of (Bangalore and Joshi, 1999), and we will adopt the name "supertag" here to avoid confusion with simple part-of-speech tags. A sample lexicalized automaton is shown in Figure 1.[5] The transitions of the automaton are labeled with pairs $\langle f, c\rangle$, where f is a grammatical function

[5]The initial state of an automaton is labeled 0 while its accepting states are indicated in boldface. The empty transitions are represented in dotted lines.

(subject, object, different types of adjuncts, etc.), and c is a supertag, or by pairs $\langle \texttt{LEX}, m\rangle$, where m is an anchor of the automaton. For expository purposes, in these examples, the supertags c are simply standard part-of-speech tags, but one should think of the symbol N in Figure 1, for example, as representing an initial tree whose anchor is of category N. This automaton indicates that the verb *eat* has a dependent which is its subject, obligatory and non-repeatable, and whose category is noun or pronoun; a dependent which is its object which is optional and non-repeatable; and an adjunct prepositional phrase which is optional and repeatable.

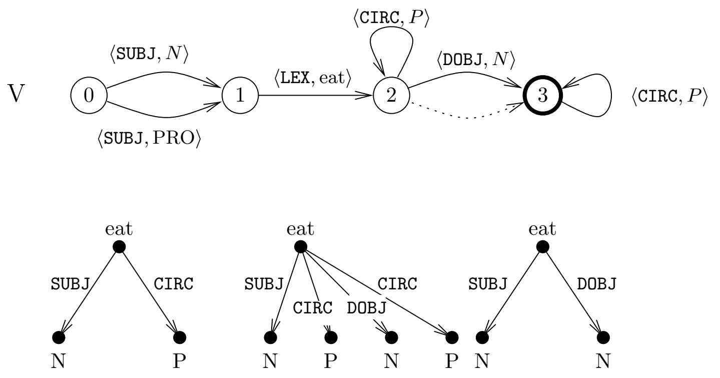

Figure 1: A lexicalized automaton and three elementary dependency trees that can be derived from it

Each word (in the formal language theory sense), i.e., each sentence (in the linguistic sense) accepted by an automaton is a sequence of pairs $\langle f, c\rangle$. Each such sequence corresponds to a dependency tree of depth one, which we will call an *elementary dependency tree* of the grammar. Three sample elementary dependency trees can be seen in the lower part of figure 1. The word corresponding to the leftmost tree is: $\langle \texttt{SUBJ}, N\rangle$ $\langle \texttt{LEX}, \text{eat}\rangle$ $\langle \texttt{CIRC}, P\rangle$.

A gdg derivation is defined like a derivation in an rtn (Woods, 1970). It uses a stack, which contains pairs $\langle c, e\rangle$ where c is the name of an automaton from the grammar, and e is a state of c. When $\langle c, e\rangle$ is on the top of the stack, and a transition of type $\langle f, c'\rangle$ goes from state e to state e' in automaton c, $\langle c, e\rangle$ is popped and $\langle c, e'\rangle$ is pushed as well as the machine c' in its initial state ($\langle c', q\rangle$). When we reach an accepting state q' in c', the pair $\langle c', q'\rangle$ is popped, uncovering $\langle c, e'\rangle$, and the traversal of automaton c resumes. We need to use a stack because, as we saw, during a derivation, several automata can be traversed in parallel, with one invoking the next recursively.

Since our automata are lexicalized, each traversal of a non-lexical arc (i.e., an arc of the form $\langle f, c\rangle$) corresponds to the establishment of a dependency between the lexical anchor of the automaton we are traversing and which we then put on the stack (as governor), and the lexical anchor of the new automaton which we start upon traversing the arc (as dependent). Thus, the result of a derivation can be seen as a sequence of transitions, which can be bijectively mapped to a dependency tree.

A probabilistic gdg, pgdg, is a gdg in which the automata of the grammar are weighted finite state automata. For each state in an automaton of the grammar, the weights of the outgoing arcs represent a probability distribution over possible transitions out of that state.

3.2 The Sites of an Automaton

The transitions of a lexicalized automaton do not all play the same role. We have already seen the lexical transitions which provide the words that anchor the automaton. In addition, we will distinguish the *argument transitions* which attach an argument as a dependent to the lexical anchor. All argument transitions which share the same grammatical function label constitute an *argument site* of the automaton. An example can be seen in Figure 2, where site 1 is the subject site, while site 4 is the object site. Note that since we consider in this example the grammatical object of *eat* to be optional, the attachment in site 4 can be skipped using its ε -transition.

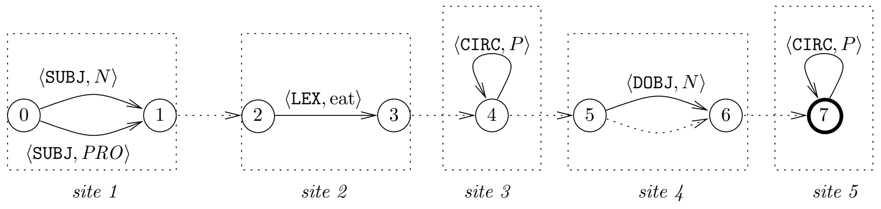

Figure 2: Sites of the automaton in figure 1

The transitions associated with adjuncts are called *adjunct transitions*. They are grouped into *adjunct sites*, such as sites 3 and 5 in figure 2. Each adjunct site corresponds to all adjunctions that can be made at one node of the tree, from one side. Some adjunct sites are repeatable, while others (such as determiners in some languages) are not. When several dependencies are generated by the same repeatable adjunct site, we distinguish them by their *position*, which we mark with integers. The argument and adjunct sites are distinguished from the lexical transitions, which are called *lexical sites*.

4 Parsing with a gdg

The parsing algorithm is a simple extension of the chart parsing algorithm for context-free grammar (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 its supertag is t. If T is the parse table for input sentence $W = w_1 \cdots w_n$ 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 $w_i \cdots w_j$ such that the root of the corresponding dependency tree is the lexical anchor of M with supertag t. The main operation we use to fill the table is the following. If $T_{i,j}$ contains (M, q_1), if there is a transition in M from q_1 to q_2 labeled t, and if $T_{j+1,k}$ contains (M', q') where M' is a t-FSM and q' is a final state, then we add (M, q_2) to $T_{i,k}$. Note that because our grammars are lexicalized, this operation corresponds to establishing a dependency between the lexical anchor of M (as head) and the lexical anchor of M' (as dependent). The algorithm is extended to lattice input following (Chappelier et al., 1999).

Before starting the parse, we create a tailored grammar by selecting those automata associated with the words in the input sentence. (Note that the crucial issue is how to associate automata with words in a sentence, which is the job of the supertagger; we do not discuss this issue in this paper, and refer to the literature on supertagging (for example, (Bangalore and Joshi, 1999)). 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 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. An extended description of this algorithm can be found in (Nasr, 2004), and a more compact description in English can be found in (Nasr and Rambow, 2004b).

5 Probabilistic Models for gdg

The parser introduced in Section 4 associates one or several analyses to a supertag sequence $S = S_1 \ldots S_n$. Each analysis $\mathcal{A}$ can be seen as a set of $n - 1$ attachment operations of one lexical node as an immediate dependent of another lexical node, 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, and $\mathcal{A}$ is therefore of cardinality n. In the following, for an attachment operation A, $O(A)$ returns its type (argument, adjunct, root), which in the case of argument and adjuncts is determined by the site at which it takes place. *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 $A_1, \ldots, A_n$. 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 describe in this section vary in the way they model multiple adjuncts attaching at the same adjunct site. Put differently, the internal structure of repeatable adjunct sites is the only difference between the models. The three models described below consider that attachments at argument sites are independent of all the other attachments that make up an analysis. The general model (following (Resnik, 1992; Schabes, 1992)) is therefore:

$$\begin{aligned} P(\mathcal{A}) &= P(\mathit{Root}) \\ &\times \prod_{A \in \mathcal{A} | O(A) = argument} P(A) \\ &\times \prod_{A \in \mathcal{A} | O(A) = adjunct} P(A) \end{aligned}$$

What is important is that the three models we present in this section change the automata, but the changes are fully within sites; if we abstract to the level of sites, the automata are identical. Furthermore, while the automata implement different probabilistic models, the same parser described in Section 4 can of course be used in conjunction with all of them.

The three models for adjunction will be illustrated on a simple example where two automata c_1 and c_2 are candidates for attachment at a given repeatable adjunct site (which we will simply refer to as a "site"). In the following models, we estimate parameters from the corpus obtained by running the TAG extraction algorithm over the PTB training corpus (see Section 6). We can then easily count the relevant events.

5.1 Model 1: Independent Attachments

In this model, an attachment at a site is considered independent from the other attachments that can take place at the same site. The probability of each attachment depends on the dependent automaton, on the governor automaton, and on the site of the governor automaton at which the attachment takes place. However, it is independent of the order of the attachments. The model does therefore not distinguish between attachments that only differ in their order. For example, the probability of the sequence $c_1c_2c_1c_2$ being adjoined is modeled as follows (we use here and subsequently a simplified notation where $P(c_1)$ designates the probability of the attachment of c_1 at the relevant site in the relevant automaton):

$$P(c_1c_2c_1c_2) = P(c_1)P(c_2)P(c_1)P(c_2)$$

It is clear that this probability is the same as that of the sequence $c_1c_1c_2c_2$ adjoining, or of any other permutation.

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 automaton c_1 attaches at a given site in order i, it does not take into account the attachments that happened for $\mathit{order} < i$. Such models also add a new parameter which is the maximum number of attachment that are distinguished (from a probabilistic point of view).

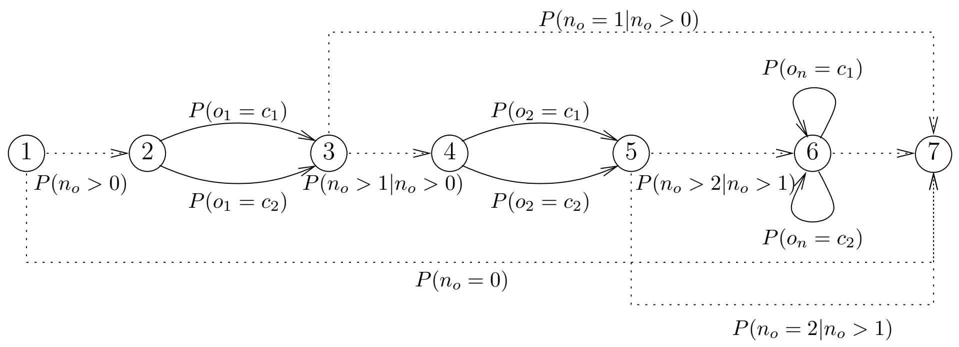

Figure 3: Positional model: a repeatable site with two positions

The automaton for a repeatable site with two positions is shown in Figure 3. It consists of a series of transitions between consecutive pairs of states. The first "bundle" of transitions models the first attachment at the site, 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 6 of the automaton. ϵ-transitions allow the attachments to stop at any moment by transitioning to state 7. (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 $c_1c_2c_1c_2$ being adjoined is:

$$P(c_1c_2c_1c_2) = P(o_1 = c_1) \times P(o_2 = c_2) \times P(o_n = c_1) \times P(o_n = c_2) \times P(n_o > 2)$$

Here, variables o_1 and o_2 represent the first and second order adjunctions. Variable o_n represents adjunctions of order higher than 2. Variable n_o represents the total number of adjunctions.

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 the frequently used horizontal markovization (see Section 2.2). Horizontal markovization is, in a sense, complementary to the positional model 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 Figure 4. The transitions with probabilities

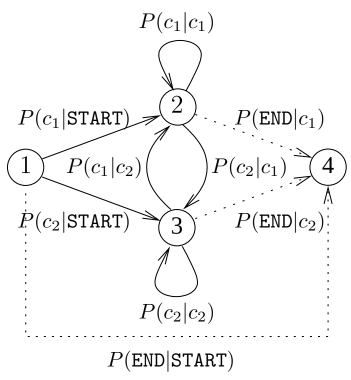

Figure 4: N-Gram model: repeatable site with bigram modeling

$P(x|\texttt{START})$ (respect. $P(\texttt{END}|x)$) correspond to the occurrence of automaton x as the first (respectively the last) attachment at this node and the transition with probability $P(\texttt{END}|\texttt{START})$ corresponds to the null adjunction (the probability that no adjunction occurs at a node). The probability of the sequence $c_1c_2c_1c_2$ being adjoined is now:

$$\begin{aligned} P(c_1c_2c_1c_2) &= P(c_1|START) \\ &\times P(c_2|c_1) \\ &\times P(c_1|c_2) \\ &\times P(c_2|c_1) \\ &\times P(END|c_2) \end{aligned}$$

5.4 Finding the n-best parses

We extend our parser by augmenting entries in the parse table with probabilities. 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 Extracting a gdg from a Corpus

We first describe the basic approach, we then show how we use the corpus to estimate probabilities, and finally we discuss the more complex models of adjunction we introduced in Section 5.

6.1 Basic Approach

To extract a gdg (i.e., a lexicalized rtn) from the Penn Treebank (PTB), we first extract a tag, and then convert it to a gdg. We make the detour via tag for the following reason: we must extract an intermediate representation first in any case, as the automata in the gdg may refer in their transitions to any other automaton in the grammar. Thus, we cannot construct the automata until we have done a first pass through the corpus. We use tag as the result of the first pass because this work has already been done, and we can reuse previous work, specifically the approach of (Chen, 2001) (which is similar to (Xia et al., 2000) and (Chiang, 2003)).

We first briefly describe the work on tag extraction, but refer the reader to the just cited literature for details. 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 as secondary lexical heads. Generally, function words such as auxiliaries or determiners are dependents of the lexical head,[6] 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.

In the second step, we directly compile this TAG grammar into a set of FSMs which constitute the gdg and which are used in the parser. To derive a set of FSMs from a tag, we do a depth-first traversal of each elementary tree in the grammar to obtain a sequence of nonterminal nodes. We exclude the root and foot nodes of adjunct auxiliary trees (its "passive valency structure"), because this structure merely tells us where and from which direction this tree can be adjoined, and we represent this information differently, namely in the structures into which this tree can be adjoined. As usual, the elementary trees are tree schemas, with positions for the lexical heads. Substitution nodes are represented by obligatory transitions, and in the basic model which assumes independent attachments, adjunction nodes are represented by optional transitions (self-loops). Adjunction nodes are represented by more complex structures in the other two models; we return to them below. Each node in the TAG tree 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 (and only for leaf nodes), the two states immediately follow one another. The states are connected with transitions as described in the next paragraph, with the left node state of the root node the start state, and its right node state

[6]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.

the final state (except for predicative auxiliary trees – see below).

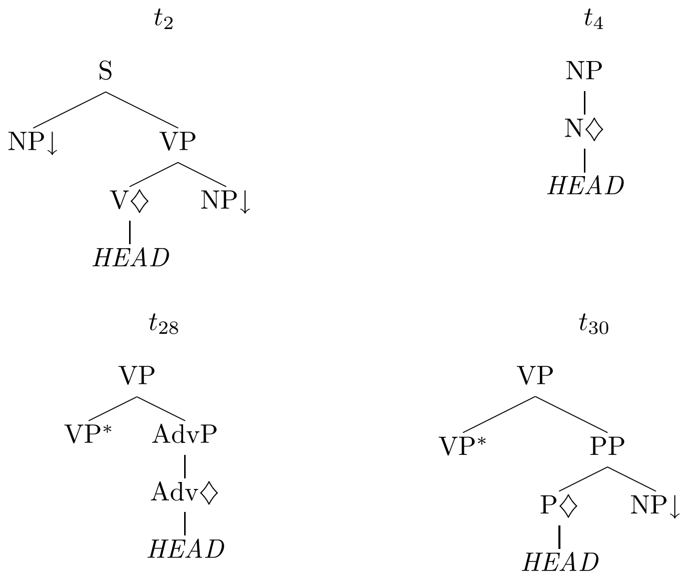

Figure 5: Sample small grammar: trees for a transitive verb, a nominal argument, and two VP adjuncts from the right

For each pair of adjacent states representing a substitution node, we add transitions between them labeled with the names of all 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 and it therefore adjoins from the left), we take the left node state as the final state. Finally, in the basic model in which adjunctions are modeled as independent (we return to the other models below), we proceed as follows for non-leaf nodes. 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 be a left auxiliary tree and 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. We give a sample grammar in Figure 5 and the result of converting it to FSMs in Figure 6.

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. As a result, the grammar is weakly equivalent to

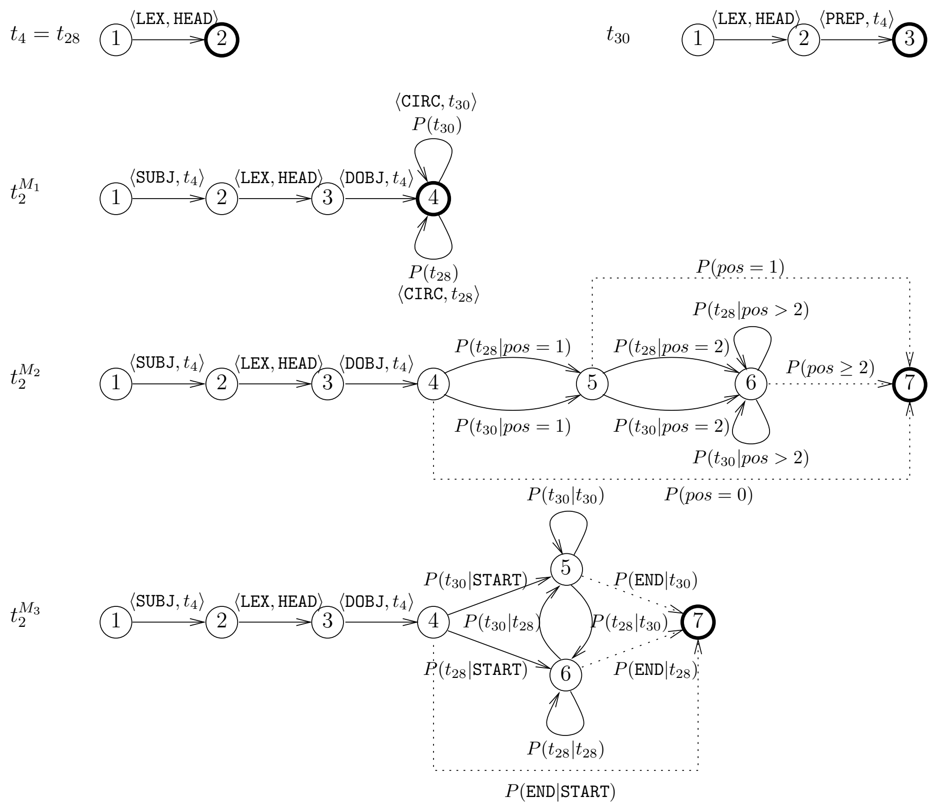

Figure 6: FSMs for trees t_4 and t_{30} (top row) from the grammar in Figure 5 (t_{28} is similar to t_4), and three FSMs for tree t_2 derived according to the basic model (second row), the positional model (third row), and the N-gram model (bottom row)

a CFG. In fact, the construction treats a TAG as if it were a Tree Insertion Grammar (TIG, (Schabes and Waters, 1995)).

6.2 The Basic Probabilistic Model

In the basic model, each event in constructing the derived tree is modeled as independent. To determine the weights in the FSM, we use a maximum likelihood estimation on the Penn Treebank with add-X smoothing.[7] We make use of a representation of the PTB which we obtain from the TAG extraction process, which provides the derivation tree for each sentence, using the extracted TAG. The derivation tree shows not only which trees are associated with each word in the sentence, but also into which other tree they are substituted or adjoined, and at which node (and, in the case of adjunction, from which direction). Given the construction above, we can always map between tree location and corresponding substitution and adjunction sites in the FSM that corresponds to the tree. In the case of adjunction, we distinguish between left and right adjunction, which correspond to self-loops on the left and right node states, respectively.

Recall that there are three types of nodes in our FSMs, which we group into sites. In this model, each site consists of two nodes. The sites are connected by ε-transitions. The lexical sites contain a single obligatory transition on the lexical head (which is instantiated when an FSM is chosen for a particular word in the input sentence). This transition is of course given a weight of 1. In the case of substitution sites, we count all cases of substitution into the corresponding tree location, and use these counts to estimate the probabilities of the transition between the left node state and the right node state of the substitution node. We perform smoothing, in order for formally possible substitutions that have not been seen in the corpus to have a non-zero probability. Since substitution is obligatory, there is no special case to consider. Adjunction is optional, so in the case of adjunction sites, we must also take into account the cases in which no adjunction occurred at that node. In the following, we assume we are considering a particular tree, a particular node in that tree, and either adjunction from the left or from the right to that node. We count the number of times each tree has been adjoined at that node from the relevant direction, and also how many times there is a "no adjunction" event. A "no adjunction" event means that no *further* adjunction occurs, it marks the end of every sequence of adjunctions (including the empty sequence of adjunctions). Thus, for every instance in the corpus of the governing supertag, and for each of its adjunction nodes, there is exactly one "no adjunction" event for adjunction from the left, and one from the right, independently of how many adjunctions occurred at that node. We do not record how many adjunctions take place at a given node and from a given direction, but rather consider the total number of events — adjunctions and "no adjunction" events — for the node and direction. We then use counts of the adjunctions to estimate the probability of the adjunction self-loops on the first state of the adjunction site, while the probability of the ε-transition to

[7] After tuning on the development corpus we chose $X = 0.00001$.

Boys/t4 like/t2 cakes/t4
Parents/t4 bake/t2 cakes/t4 daily/t28 in/t30 kitchens/t4 with/t30 gusto/t4
Boys/t4 eat/t2 cakes/t4 beside/t30 dogs/t4 after/t30 snowstorms/t4
Parents/t4 allow/t2 binges/t4 reluctantly/t28

Figure 7: Sample corpus

the second state is estimated based on the number of "no adjunction" events.

We illustrate this estimation with a simple (obviously made-up) example corpus, shown in Figure 7, in which a supertag is associated to every word. The elementary trees corresponding to the supertags are represented in Figure 5. We omit the tree addresses at which the operations occur, as the grammar is so simple. We assume there are no other trees in the grammar. For the two substitution nodes in t_2 and the one in t_{30}, there is only one possible tree that can substitute, t_4, so its probability is 1. The interesting case are the arcs emerging from the fourth state of $t_2^{M_1}$, which is the first state of its only adjunction site. There are six adjunctions at this node in our corpus, two of t_{28} and four of t_{30}, as well as four "no-adjunction" events (since there are four instances of t_2). We get probabilities of 0.2 for t_{28}, and of of 0.4 for both t_{30} and the ε-transition to the next (and final) state without smoothing.

6.3 The Positional and N-Gram Probabilistic Models

In Section 4, we discussed two other models that treat non-leaf of extracted elementary trees nodes in a more complex manner. For the positional model, we create a new distribution for each position, i.e., for the first adjunct in that position, for the second adjunct, and so on. Specifically, we count how often a particular tree was adjoined in position n, and how often there were exactly $n-1$ adjunctions (to estimate the probability of no adjunction at position n). The positional model is parametrized for the number of positions explicitly modeled; beyond this value of this parameter, we use the basic model. (In fact, the basic model is the same as the positional model with no positions explicitly modeled.) We show the probabilities for the middle model in Figure 4 in Figure 8.

In the N-Gram model, we estimate the probabilities of bigrams for those trees that can be adjoined at the same node. Note that we do not use bigrams to model the probabilities of *all* sister nodes in a dependency tree (as do some models), only those sister nodes that result from adjunctions at the same node in the governing tree. We estimate the probabilities by counting the number of bigrams found for a given tree, a given node in that tree, and a given adjunction direction. If fewer than N cases of the tree were found in the corpus, we use the category of the node instead. The probabilities are smoothed using linear interpolation of the unigram and bigram probabilities.

$P(pos = 0)$	0.250,001
$P(t_{28} \| pos = 1)$	0.499,999
$P(t_{30} \| pos = 1)$	0.250,001
$P(pos = 1)$	0.333,333
$P(t_{28} \| pos = 2)$	0.000,003
$P(t_{30} \| pos = 2)$	0.666,663
$P(pos \geq 2)$	0.666,663
$P(t_{28} \| pos > 2)$	0.000,003
$P(t_{30} \| pos > 2)$	0.333,333

Figure 8: Transition probabilities on automaton $T_2^{M_2}$ for the positional model (see Figure 6), given sample corpus in Figure 7; figures may not add up to 1 due to rounding

7 Results

In this section, we present results using three types of supertagged input: the gold supertag (i.e., the correct supertag); the output of a trigram HMM supertagger; the (better) output of a maximum entropy supertagger. All results reported in this section are based on unlabeled evaluation. (In our dependency representation, the only relevant labels are the arc labels, and we use labels that identify the deep subject, the deep object, the deep indirect object, and a single label for all adjuncts.)

7.1 Using the Gold Supertag

In this evaluation, 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 depend on itself (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 (should a node remain unattached in a parse, it is given itself as governor). 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

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 9: 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)

is about 94.35%, on Section 23 95.14%. 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 94.73% on Section 00, 94.69% on Section 23. As we can see, the supertags provide sufficient information to result in high baselines. The results are summarized in Figure 9.

There are several clear conclusions to be drawn from Figure 9. 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 using gold supertags. On the test corpus, the error rate increases to 2.39%.

Number of paths	Stag. Acc.	Dep. Acc.
1	81.3%	71.1%
10	85.6%	76.4%
20	88.2%	77.9%
50	89.2%	78.4%
100	89.9%	79.4%
500	90.7%	79.9%
1000	90.8%	79.3%
1500	90.9%	79.5%
2000	90.9%	79.7%

Figure 10: Accuracy of the trigram tagger and the parser on the development corpus (first 800 sentences from Section 00) as a function of the number of supertagging paths taken into account: accuracy of the supertags of the best path among the n-best paths ("Stag. Acc.") and accuracy of the best dependency parse among the n-best paths ("Dep. Acc.")

7.2 Using Supertags Predicted by an HMM Tagger

In this section, we present results on an initial subsection of Section 00 of the PTB (800 sentences), using a standard trigram HMM tagger with backoff, where the parameters were calculated using the CMU Language Modeling Toolkit. An HMM tagger used in conjunction with a Viterbi decoder can be used to obtain n-best paths. Since our parser was extended to take lattices as input, we give the parser as input a lattice representing the n-best paths.

We show in Figure Figure 10 the results as a function of the number of paths. As we can see, beyond 100 paths, the results of both the supertagger and the parser fluctuate a bit, but presumably not at a statistically significant level. We assume (backed by the analysis of some cases) that this is due to the fact that the paths start to differ only in choices which do not affect the parse. Note that the LDA obtains a score of about 75% dependency accuracy on the 1-best path; to process multiple paths, it must be run multiple times, and there is no way to choose among multiple results.

7.3 Using Supertags Predicted by a Maximum Entropy Tagger

In this subsection, we report results using two innovations: we use a better-performing tagger, and we use a much faster parser. We discuss these in turn.

Given the non-local nature of supertag dependencies, we move from an trigram-based HMM tagger to a tagger based on a standard maximum entropy (maxent) model (Bangalore et al., 2005). The features used for the maxent model include: the lexical and part-of-speech attributes from the left and right context in a six-word window and the lexical, orthographic (e.g. capitalization, prefix, is digit) and part-of-speech attributes of the word being supertagged.

Category	Dependency Accuracy	Number
Det	94.9%	3,989
Adj	91.8%	3,190
N	88.6%	15,675
P (no *to*)	73.6%	3677
Adv	81.5%	1554
V	83.2%	6262
Conj	68.3%	1124

Figure 11: Accuracy of choice of governor for words of different categories in the best parse on Section 00; the third column shows the number of cases in the test corpus

Note there is no use of preceding supertags, which permits very fast (parallel) decoding of a sentence. However, as a result, we do not obtain n-best paths, but only n-best supertags (for each word). The 1-best accuracy of this supertagger is 85.7% and 85.5% on Sections 00 and 23, respectively.

The use of a new tagger has an important consequence: since the tagger no longer emits n-best paths of supertags but n-best supertags per word, the number of possible paths increases exponentially with the sentence length, resulting in a performance problem for the original implementation of our parser. We therefore reimplemented the parser using the parsing approach of (Boullier, 2003), in which a context-free grammar is compiled into an Earley-type parser. In this approach, the plain gdg is transformed into a CFG, from which the Earley parser is compiled. The parser outputs a shared parse forest from which the search algorithm extracts the best parse, according to a given probabilistic model. Of course, we chose our best-performing model (the N-gram model). The main advantage of this parser is a significant speed-up, which allows us to take more supertags into account when parsing, and to use a wider beam. All results that we report in this subsection were obtained using this reimplementation of the parser. These results represent the best performance of our parser at the present moment.

The new parser has two parameters: the beam width, and the number of supertags taken into account in the input. Suppose α is the beam width. Then if the highest confidence score for all supertags of a given word is s, we eliminate all supertags with a confidence score less than s/α. Using Section 00, we determine the optimal values of these parameters to be 9 input supertags per word, and a beam width of 224. The tagger takes about 8.2 seconds for 1,000 words (when tagging an entire section of the Penn treebank), while the parser takes about 7 seconds at these parameter settings, for a total of about 15.2 seconds per 1,000 words end-to-end. A smaller beam or a smaller number of input supertags results in a faster parse, at the expense of accuracy.

Overall we obtain a dependency accuracy score of 84.7% and 84.8% on Section 00 and 23, respectively, of the Penn Treebank. This result is worse than that

Length ≤ %	Corr. Sent.	Dep. Accuracy	Stag acc.	Number
5	8194%	8643%	7868%	72
10	7181%	8949%	8251%	259
15	5728%	8956%	8278%	604
20	4611%	8855%	8235%	989
25	3920%	8756%	8162%	1306
30	3397%	8700%	8112%	1572
35	3069%	8637%	8053%	1756
40	2937%	8611%	8021%	1835
45	2868%	8593%	8008%	1883
50	2839%	8571%	7987%	1902
80	2817%	8545%	7959%	1917
300	2811%	8467%	7884%	1921

Figure 12: Using Maxent supertagger, percentage of completely correct sentences, accuracy of choice of governor, and accuracy of supertag in best parse for sentences of different maximum length in Section 00: sentences with completely correct parses ("corr. Sent."), accuracy of the best dependency parse among the n-best paths ("Dep. Acc."), accuracy of the supertags of the best path among the n-best paths ("Stag. Acc."), and number of sentences of the specified length ("Number")

obtained by Clark and Curran (2004) (92.5% recall, 92.1% precision). However, their measure does not take into account the direction of the dependency arc, while our accuracy figure does. Furthermore, the gold dependency structures are different (ours are oriented towards the predicate-argument structure), so a direct comparison is difficult. (Clark and Curran (2004) use recall/precision as some nodes may not be assigned governors. We use accuracy as all nodes are assigned governors, so for us recall, precision, and accuracy all have the same value.)

We give some additional data for the results for Section 00. First, we observe that the root is chosen correctly in 90.0% of cases. 99.5% of sentences have a complete analysis (i.e., analyses in which all nodes except a single root node are dependent on another node in the sentence, with no unattached subtrees), and 28.1% have a fully correct analysis. Figure 11 shows the distribution of attachment error rate for different part-of-speech categories. We see that determiners and adjectives, both nominal dependents with restricted syntax in English (and a simplified analysis in the PTB) do best, while conjunctions do worst. The relatively low performance on verbs reflects the difficulty of determining proper attachment for relative clauses.

Finally, Figure 12 shows the results as a function of maximum sentence length. We see the expected deterioration with sentence length, except that very short sentences do not perform as well as slightly longer sentences. We also note that the supertag accuracy on the best parses is worse than the accuracy

Type	Corpus	Stag accuracy	Parsing accuracy
HMM	Sec. 00 (800 sentences)	81.3%	79.7%
Maxent	Sec. 00	85.7%	84.7%
Maxent	Sec. 23	85.5%	84.8%
Gold	Sec. 00 (seen stags)	100.0%	97.7%

Figure 13: Summary of results for the 1-best supertagger and the resulting parser (which, except for the Gold case, is based on n-best supertags or supertag paths)

of the supertagger.

7.4 Summary of Results

We summarize the results in Figure 13. As we can see, increased performance in supertagging leads to increased performance in parsing, as we would expect. Given only three data points, with rather different characteristics, we do not wish to hazard generalizations, but the data suggests strongly that further research into improving supertagging will also directly improve parsing.

8 Conclusion

We have presented a nonlexical probabilistic chart parser for TIG which works in conjunction with a supertagger. It is implemented using finite-state automata. We have shown that this parser improves on the results of the Lightweight Dependency Analyzer, while retaining the intuition of localizing all lexical information in the supertagging stage. Furthermore, we have shown that the parser performance is roughly a linear function of the supertagger performance. This implies that research in further increasing the performance of a supertagger will also directly benefit parsing.

Acknowledgments

We would like to thank John Chen for letting us use his extracted grammar; Srinivas Bangalore, Patrick Haffner, and Gaël Emamifor the use of their supertagger; Pierre Boullier for for his patient and efficient assistance during the development of the new parser; and an anonymous reviewer for helpful comments.

Bibliography

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

Srinivas Bangalore, Patrick Haffner, and Gaël Emami. 2005. Factoring global inference by enriching local representations. Technical report, AT&T Labs – Reserach.

Pierre Boullier. 2003. Guided Earley parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT03)*, pages 43–54, Nancy, France, April.

Jean-Cédric Chappelier, Martin Rajman, and Antoine Rozenknop. 1999. Lattice parsing for speech recognition. In *Traitement Automatqiues du Langage Naturel (TALN'99)*, Cargèse, France.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *1st Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL'00)*, pages 132–139.

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

David Chiang. 2003. Statistical parsing with an automatically extracted tree adjoining grammar. In Rens Bod, Remko Scha, and Khalil Sima'an, editors, *Data-Oriented Parsing*. CSLI Publications, Stanford.

Stephen Clark and James R. Curran. 2004. Parsing the WSJ using CCG and log-linear models. In *42nd Meeting of the Association for Computational Linguistics (ACL'04)*, Barcelona, Spain.

Stephen Clark, Julia Hockenmaier, and Mark Steedman. 2002. Building deep dependency structures with a wide-coverage CCG parser. In *40th Meeting of the Association for Computational Linguistics (ACL'02)*, 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.

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)*.

Alexis Nasr and Owen Rambow. 2004a. A simple string-rewriting formalism for dependency grammar. In *Recent Advances in Dependency Grammar: Proceedings of the Coling Worshop*.

Alexis Nasr and Owen Rambow. 2004b. Supertagging and full parsing. In *Proceedings of the Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+7)*, Vancouver, BC, Canada.

Alexis Nasr. 2004. Analyse syntaxique probabiliste pour grammaires de dépendances extraites automatiquement. Habilitation à diriger des recherches, Université Paris 7, December.

Joakim Nivre. 2006. *Inductive Dependency Parsing*. Text, Speech, and Language Technology. Springer.

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.

Anoop Sarkar and Aravind Joshi. 2003. Tree-adjoining grammars and its application to statistical parsing. In Rens Bod, Remko Scha, and Khalil Sima'an, editors, *Data-Oriented Parsing*. CSLI Publications, Stanford.

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 15th [sic] International Conference on Computational Linguistics (COLING'92)*.

William A. Woods. 1970. Transition network grammars for natural language analysis. *Commun. ACM*, 3(10):591–606.

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.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop of Parsing Technologies (IWPT2003)*, Nancy, France.