Natural Language Engineering **12** (3): 251–299. © 2005 Cambridge University Press 251
doi:10.1017/S1351324905003943 First published online 22 December 2005 Printed in the United Kingdom

Automated extraction of Tree-Adjoining Grammars from treebanks

JOHN CHEN†
Microsoft Research Asia, No. 49 Zhichun Road, Haidian District, Beijing 100080, China
e-mail: t-Johnc@microsoft.com

SRINIVAS BANGALORE
AT&T Labs–Research, P.O. Box 971, 180 Park Avenue, Florham Park, NJ 07932, USA
e-mail: srini@research.att.com

K. VIJAY-SHANKER
Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716, USA
e-mail: vijay@cis.udel.edu

(*Received 30 April 2003; revised 20 December 2004*)

Abstract

There has been a contemporary surge of interest in the application of stochastic models of parsing. The use of tree-adjoining grammar (TAG) in this domain has been relatively limited due in part to the unavailability, until recently, of large-scale corpora hand-annotated with TAG structures. Our goals are to develop inexpensive means of generating such corpora and to demonstrate their applicability to stochastic modeling. We present a method for automatically extracting a linguistically plausible TAG from the Penn Treebank. Furthermore, we also introduce labor-inexpensive methods for inducing higher-level organization of TAGs. Empirically, we perform an evaluation of various automatically extracted TAGs and also demonstrate how our induced higher-level organization of TAGs can be used for smoothing stochastic TAG models.

1 Introduction

It has been found that inclusion of lexical information confers advantages to both theoretical and statistical models of natural language. This kind of information is naturally encoded in a lexicalized grammatical formalism such as TAG. In the case of TAG, however, interesting work has been done for a long time in the linguistic domain (e.g. Frank 2002), but until recently there has been less work done in stochastic modeling. In our opinion, this has been because of the lack of large-scale corpora hand-annotated with TAG structures. It is not surprising that such corpora have been unavailable until recently because of the considerable time and effort

† The majority of this work was done while the author was studying at the University of Delaware. This work was supported by NSF grants #SBR-9710411, #GER-9354869, and #CDA-9703088.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

that is required to annotate such corpora with the relatively rich annotation that TAG provides and because existing manually designed TAG grammars do not have sufficient coverage. In this work, we introduce techniques that significantly reduce the labor that is required to build wide-coverage grammars by extracting TAGs from pre-existing, widely available treebanks. In this process, we automatically get a corpus annotated with TAG structures which has contributed to the recent activity in building stochastic TAG models (cf. Bangalore, Chen and Rambow 2001; Chen, Bangalore, Collins and Rambow 2002a; Chen, Bangalore, Rambow and Walker 2002b).

We would like to extract a TAG from a treebank such that the resulting TAG grammar can derive the original unlabeled treebank bracketings of words exactly. Furthermore, we make the assumption that this TAG should conform to the linguistic principles that govern the extraction of hand-written linguistically plausible TAGs. We make this assumption because it is our purpose to see how these principles, including the factoring of recursion and the localization of dependencies, which play such a central role in TAG grammars, affect a stochastic model of TAG. It is the case that these principles do not fully dictate the form of an extracted TAG given an input treebank. We are thus able to parameterize our extraction procedure to extract different kinds of TAGs from the same treebank while still being faithful to linguistic principles.

Although at its core a handwritten TAG is a set of lexicalized trees, this set is also often characterized manually in terms of a large-scale organization (Xia, Palmer, Vijay-Shanker and Rosenzweig 1998; XTAG-Group 2001). The latter has many uses. Linguistically, it generalizes distinct grammatical structures as grammatical processes. Computationally, it provides a succinct representation of TAG that is useful for debugging the grammar and potentially smoothing statistical TAG models. For these reasons, we introduce both semi-automated and fully automated techniques that extract this large-scale organization from an automatically extracted TAG grammar.

Now, given these procedures that extract both a TAG grammar and its large-scale organization, it is interesting to evaluate the quality of these resources. In accordance with our goal of investigating the use of TAG in stochastic modeling, we evaluate these resources on the supertagging task (Bangalore and Joshi 1999).[1] We also evaluate according to other metrics such as grammar size and ability to localize dependencies properly. Also note that because the TAG extraction procedure is parameterized, we are in the position to compare and contrast a set of automatically extracted TAGs. Furthermore, we perform experiments that

[1] We choose supertagging over stochastic TAG parsing in part because we are mainly interested in how different extracted grammars affect a statistical model. Differences in parsing performance between stochastic TAG parsing models based on different grammars may be due to differences between the grammars. However, because of the relative complexity of stochastic TAG parsing, including its backoff model, there may be other reasons. For example, perhaps the backoff model or the kind of beam search used is optimal for one kind of grammar but not another.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

show how the automatically-induced large-scale organization of TAGs can be used for smoothing supertagging models.

Others have performed work that is similar to our work in extraction of TAGs from the Penn Treebank, mostly in parallel with or after our work. A brief description of related work is presented here, while a more detailed exposition is reserved for section 7. Neumann (1998) presents a method for extracting a lexicalized tree grammar from a treebank. Xia (1999) presents a method for extracting a lexicalized TAG from a treebank. Xia, Palmer and Joshi (2000) compare it to a hand-crafted TAG and do supertagging experiments. Xia, Han, Palmer and Joshi (2000) compare TAGs extracted from treebanks of different languages using this method. Chiang (2000) presents a method for extracting a lexicalized tree insertion grammar (LTIG) from a treebank and applies it to full stochastic LTIG parsing. Hockenmaier and Steedman (2002b) present a method for extracting a combinatory categorial grammar (CCG) from a treebank. It is applied to CCG supertagging (Clark 2002) and full CCG parsing (Hockenmaier and Steedman 2002b).

In the following sections, we first review TAG and crucial principles that guide the formation of hand-written TAGs. Subsequently, we introduce the automated TAG extraction procedure and explain how it conforms to these principles. Next, we describe the automated procedures for inducing a large-scale organization of TAG. Following this, we evaluate automatically extracted TAGs corresponding to different parameterizations with each other before also evaluating an automatically extracted TAG against a hand-written TAG. We then describe and evaluate smoothing models for supertagging based on the large-scale organization of TAG. Finally, we discuss the results in relation to related work and summarize the salient aspects of this work.

2 Extraction of a TAG from the Penn Treebank

This section describes the TAG extraction procedure. It was devised to extract TAGs that conform to the general linguistic principles that guide the formation of handwritten TAGs. These principles will be discussed first. These principles will be insufficient to determine fully how a TAG should be extracted from a particular treebank. We will therefore discuss in this section not only the basic treebank extraction procedure but also some parameterizations of this procedure.

2.1 Guidelines for TAG formation

Restrictions on the shapes and sizes of the elementary trees which compose a TAG for a particular language may be divided into two types: those that are imposed by the TAG formalism and those that are imposed by linguistic theory and custom. After reviewing the restrictions imposed by the former, we will review those imposed by the latter.

A TAG G can be defined as the tuple $\langle N, T, I, A \rangle$. N is a set of nonterminal labels, T is a set of terminal labels, I is a set of *initial trees*, and A is a set of *auxiliary trees* (Schabes, Abeillé and Joshi 1988). The set $I \cup A$ is called the set of *elementary trees*

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

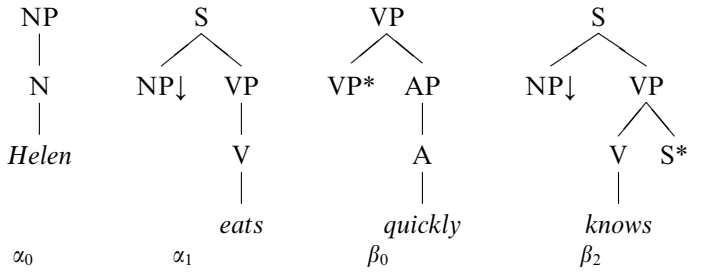


Fig. 1. α_0 and α_1 represent initial trees. β_0 represents a modifier auxiliary tree. β_2 represents a predicative auxiliary tree.

or *lexicalized trees* of grammar G. The frontier of each elementary tree $\gamma \in I \cup A$ contains a *lexical anchor* η_a; other nonterminal nodes on the frontier are *substitution nodes*, and, in the case of an auxiliary tree, one node on the frontier will be a *foot node* η_f whose label is the same as the root η_r of the tree. The *spine* of an auxiliary tree is the path from its root η_r to its foot node η_f. It is to be distinguished from the *trunk* of an elementary tree, initial or auxiliary, which is the path from η_r to η_a. Deletion of the lexical anchor from an elementary tree yields a *tree frame* or a *supertag*.

Figure 1 contains examples of lexicalized trees from a TAG grammar. Each substitution node is marked with ↓. Each foot node is marked with *. The substitution node represents the location where the TAG operation of substitution occurs. Conversely, the foot node is involved with the TAG operation of adjoining. These nodes represent locations where the TAG operations of substitution and adjoining, respectively, occur. α_0 and α_1 are initial trees. β_0 represents an auxiliary tree where VP* is the foot node. Trees α_0, α_1, β_0, and β_2 are lexicalized as can be seen by the presence of an anchor in each one of these trees. Note also that dependencies are localized with respect to these lexical items. For example, the substitution node which is labeled NP↓ in the tree α_1 represents the fact that the verb *eats* takes an NP subject.

The TAG formalism allows wide latitude in how trees in G may be defined. Nevertheless, several linguistic principles guide their formation. First, dependencies, including long distance dependencies, are typically localized in the same elementary tree by appropriate grouping of syntactically or semantically related elements; i.e. complements of a lexical item are included in the same tree as is the case of α_1. Second, recursion is factored into separate auxiliary trees. Schabes and Shieber (1994) note that for natural language grammars these recursive elements may be divided into two types: those that represent modification, and those containing clausal complements out of which extraction can take place. According to this classification, they proceed to divide auxiliary trees into *modifier auxiliary trees* and *predicative auxiliary trees*. In Figure 1, β_0 represents a modifier auxiliary tree and β_2 represents a predicative auxiliary tree.

Conjunction is commonly represented in TAG by the use of auxiliary trees (XTAG-Group 2001). Such an auxiliary tree is anchored by the conjunction. It is

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

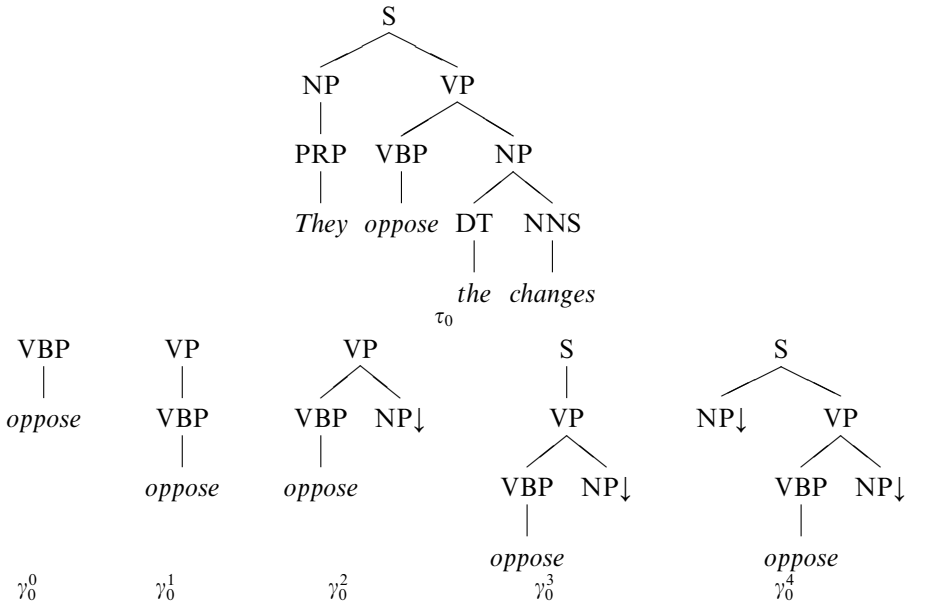


Fig. 2. τ_0 is a Penn Treebank bracketing. γ_0^0 through γ_0^4 represent partial trees on the way to generating the elementary tree γ_0^4 anchored by the verb "oppose."

meant to adjoin onto the first conjunct and also contains a substitution node where the second conjunct should attach.

2.2 Basic recursive tree extraction procedure

We develop a procedure to decompose a parse tree from the Penn Treebank into a set of elementary trees. These automatically generated trees are linguistically well-motivated in the sense that this procedure follows the linguistic guidelines that constrain handwritten TAGs. In this section, we describe one approach for extracting a TAG. In a later section, we discuss alternative parameterizations.

The outline of this section is as follows. To provide an overview of the tree extraction procedure, we first present the basic steps that are necessary to extract exactly one elementary tree from a Penn Treebank bracketing. Next, we enumerate the steps that comprise one call to the procedure, which is recursive. Finally, we discuss at greater length certain key aspects of this procedure, including determination of the head of a phrase and whether a given node is a complement or an adjunct.

The tree extraction procedure is applied recursively and bottom-up to an input bracketing. During its application, there are typically multiple elementary trees in various stages of being generated. We start by describing how one particular elementary tree is generated from the input bracketing. In particular, we will describe how the elementary tree γ_0^4 is extracted from the bracketing τ_0 in Figure 2.

The procedure begins by associating the anchor word "oppose" and its corresponding preterminal with partial tree γ_0^0. A *partial tree* is either an incompletely-specified or a completely-specified elementary tree. The full elementary tree is

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

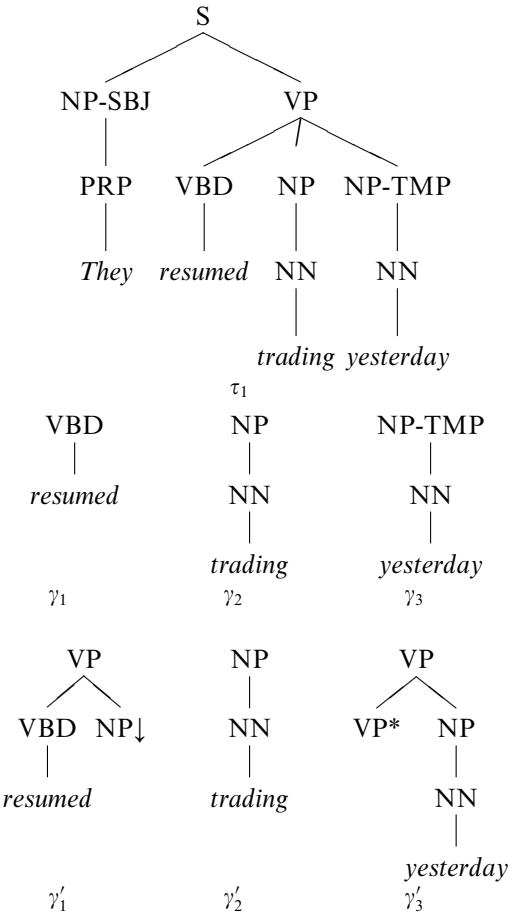


Fig. 3. τ_1 is a Penn Treebank bracketing. γ_1, γ_2, and γ_3 are partial trees at one stage in the tree extraction procedure. γ'_1, γ'_2, and γ'_3 are the same partial trees at the next stage.

constructed bottom up from γ_0^0, alternating between two steps: first, extending the trunk of the partial tree by one level and second, adding to the root of the resulting partial tree as daughters substitution nodes whose corresponding nodes in τ_0 represent complements. Results of the first step are shown as γ_0^1 and γ_0^3. Results of the second step are shown as γ_0^2 and γ_0^4.

Rather than being serially extracted from an input bracketing, elementary trees are extracted in parallel by the tree extraction procedure. Consider the bracketing τ_1 in Figure 3. At one stage in the tree extraction procedure, partial trees γ_1, γ_2, and γ_3 have been extracted. Let η_1, η_2, and η_3 be the root nodes of γ_1, γ_2, and γ_3, respectively. η_1, η_2, and η_3 correspond to sibling nodes labeled VBD, NP, and NP-TMP in bracketing τ_1. Let η^p be a new node corresponding to these siblings' parent, the node labeled VP.

One partial tree is selected to be the *head partial tree* based on the roots of the partial trees and their common parent. In this case, γ_1 is selected out of γ_1, γ_2 and γ_3 based on the identities of η_1, η_2, η_3, and η^p. Roots of the remaining partial trees

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

Table 1. *Excerpt from a head percolation table*

Label of η^p	Direction	Priority list of labels of η
SBAR	right	S SQ SINV SBAR FRAG X
S	right	VP SBAR ADJP UCP NP
VP	right	VP NP-PRD ADJP-PRD ADVP-PRD VBD VBN VBP ...
NP	right	(NN NNS NNP NNPS) NP NAC EX $ CD QP ...
...	...	...

are then labeled as either complement or adjunct. In this case η_2 is labeled as a complement and η_3 is labeled as an adjunct. The head partial tree is then extended one level. Here, γ_1 projects up to η^p (the node labeled VP). Duplicates are made of complement nodes such that they become substitution nodes in the head partial tree. In our example, the duplicate of η_2 becomes an NP substitution node in γ_1'.

Now consider a partial tree whose root is labeled as an adjunct. It is factored into a modifier auxiliary tree that adjoins onto the head partial tree. For example, since η_3 is an adjunct node, γ_3 becomes the modifier auxiliary tree γ_3'. Note that a hierarchical structure in the resulting extracted TAG elementary trees has been induced that is not present in the relatively flat Penn Treebank input bracketing.[2]

This sketch of the tree extraction procedure leaves out several vital components. One component is a method for determining the head child η_h of a phrase when given a parent node η^p and its children $\eta_1, \ldots, \eta_m$. In order to accomplish this task, we use a slight variant of what is called a *head percolation table*, which was introduced by Magerman (1994). An excerpt of a head percolation table is shown in Table 1, while a complete version along with a detailed discussion may be found in the appendix. The first column is the label of the parent node η^p; the third column is used to assign priorities to various possible child labels; the second column determines the priorities of children with the same child label. Sets of labels in parentheses have equal priority. η_h is the child of η^p with the highest priority according to the table. For example, given η^p = VP, η_1 = VBD, η_2 =NP, and η_3 =NP-TMP, as in Figure 3, η_h =VBD since VBD has higher priority than either NP or NP-TMP.

The particular variety of head percolation table being used is the one used by Collins (1999), with several modifications. These modifications are generally employed so that semantic dependencies are localized in accordance with TAG formation principles.[3] For example, in Table 1 the head child of SBAR is preferentially an S node with the result that a wh-extracted complement is in a position to be localized into the same tree with their corresponding trace. This is unlike Collins

[2] Such flat structures are not possible with TAG if nodes are distinguished to be either complement or adjunct. It is possible, however, to further annotate the extracted TAG trees so that the flat structure of the Penn Treebank is recoverable from the Penn Treebank (Chen2001). Also note that extracted grammars based on the D-Tree grammar formalism (Rambow, Vijay-Shanker and Weir 1995), a relation of TAG, is able to keep the complement/adjunct distinction while preserving the flatness of the Penn Treebank.

[3] It is possible to modify the table so as to meet other goals, such as optimizing the accuracy of a stochastic TIG parser. This alternative has been pursued in Chiang and Bikel (2002).

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

Table 2. *The complement-adjunct table is used to distinguish complements from adjuncts*

Label of η	Label of η^p if η is complement	Label of η^p if η is adjunct
PP-CLR	VP	–
NP	VP PP PP-CLR PP-EXT ...	ADJP NP NP-SBJ
NP-SBJ	S	–
WHADVP	SBARQ SBAR SBAR-LOC ...	–
...	...	...

(1999), where the head child of SBAR is preferentially the extracted complement, e.g. WHNP or WHPP; if this preference were kept, the extracted complement would eventually head its own elementary tree, separate from the elementary tree where its trace resides.

Aside from the ability to determine the head child η_h among a set of daughters $\eta_1, \ldots, \eta_m$, given their parent η^p, the tree extraction procedure requires a method of determining whether each sister $\eta_i, 1 \leq i \leq m, i \neq h$ should be regarded as either a complement or an adjunct.

As XTAG-Group (2001) points out, the precise definition of complement and adjunct is a matter of debate. XTAG-Group themselves rely on the intuition of grammaticality judgments in order to distinguish between complements and adjuncts. This is only natural considering that their task focuses on creating a hand-written grammar. In terms of an experiment on large scale corpora, this approach is difficult because it involves a high cost in manual labor. For this kind of task, a different approach is required. For example, Briscoe and Carroll (1997) automatically detect subcategorization frames by first parsing raw text and then applying various filters to the results. For our approach, we assume treebanked text instead of raw text as input. Another approach (Buchholz 1998; Xia *et al.* 1998; Collins 1999; Chiang 2000; Merlo and Leybold 2001; Hockenmaier and Steedman 2002b) is to automatically determine complement and adjunct status based on information found in the Penn Treebank. We pursue this alternative here.

Our method for distinguishing complements from adjuncts relies on heuristics that are based on local structural information. In particular, these heuristics use the label, semantic tags, and grammatical tags of a node and its parent. Exactly these characteristics are used as an index into a manually constructed table to determine complement or adjunct status. Most of the time, this table is sufficient to distinguish complements from adjuncts. We first describe use of the table before discussing the exceptional situations.

A portion of this table, called the *complement adjunct table*, is shown in Table 2. The first column specifies characteristics of the node to be categorized.[4] The second column specifies the characteristics that its parent node must have when the node is

[4] In the table, the node's label appears before the first dash, while the node's semantic or grammatical tags appear after the first dash.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

Table 3. *A description of the semantic and grammatical tags that are found in the Penn Treebank to which we refer in this paper*

ADV	Marks a constituent other than ADVP or PP when it is used adverbially: "Airplanes crash *if they run out of fuel*."
CLR	Marks constituents that occupy some middle ground between argument and adjunct of the verb phrase: "Migrant laborers donate money *to impoverished CEOs*."
DTV	Marks the dative object in the unshifted form of the double object construction: "The organ grinder gives free concerts *to disabled veterans*."
EXT	Marks adverbial phrases describing the spatial extent of an activity: "The Dow Jones plunged *1003 points*."
LGS	Marks the logical NP subject in passives: "The harp seal was eaten by *the polar bear*."
LOC	Marks adverbials that indicate the place or setting of an event: "Neil put the flag *on the moon*."
NOM	Marks free ("headless") relatives and gerunds when they act nominally: "*Baking pies* leaves me exhausted."
PRD	Marks any predicate that is not VP: "UFOs are *figments of your imagination*."
PUT	Marks the locative complement of *put*: "The cop put the revolver *in the baby stroller*."
SBJ	Marks the surface subject: "*The harp seal* was eaten by the polar bear."
VOC	Marks nouns of address: "*Phil*, bludgeon the harp seal with the truncheon please."
PRP	Marks purpose or reason clauses or PPs: "*For the love of God*, take the eskimo out of the freezer."
TMP	Marks adverbials that answer the questions *when*, *how often*, or *how long*: "We will return *after a word from our sponsor*."

a complement. Likewise, the third column specifies the characteristics that the parent node must have when the node is an adjunct. Meanings of semantic or grammatical tags as defined by the Penn Treebank annotation guide (Bies *et al.* 1995) are shown in Table 3.

Use of this table is straightforward. For example, suppose the node is labeled NP-SBJ and its parent is labeled S. The node is then labeled as a complement because S is found in the second column of the row whose first column is labeled NP-SBJ.

There are several exceptions to the above use of the table. One exception has to do with the semantic tag CLR. According to Marcus, Kim, Marcinkiewicz, MacIntyre, Bies, Ferguson, Katz, and Schasberger (1994), CLR stands for "closely related." It is used for elements that the annotators of the Penn Treebank believed to be not quite arguments and not quite adjuncts. We treat a node η_i that contains the label CLR as an argument if and only if the complement-adjunct table deems η_i to be an argument and there is no other argument node intervening between η_i and the head child η_h. When at least one argument node intervenes between a head and a CLR-marked node, the latter is less likely to be an argument because of the observation that arguments generally appear quite close to their heads (Radford 1988). For example, the PP-CLR phrase "as a nonexecutive director" in the sentence "Pierre Vinken will join the board as a nonexecutive director" will be marked as an adjunct because there is an NP argument "the board" intervening between the PP-CLR and its verbal head "join." The other exception to the use of the complement-adjunct table treats as arguments those PP nodes that represent passive "by" phrases. In the Penn

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

Treebank, the labeling of these PP nodes is indistinguishable from the labeling of PP adjunct nodes. In order to make the distinction, it is necessary to determine whether the PP's object is labeled with -SBJ, which would be true only in the former case.

The complement-adjunct table is sparse; should the labels and associated tags of a node and its parent not be found in the table then the following procedure is invoked:

1. Nonterminal PRN is an adjunct.
2. Nonterminals with semantic tags NOM, DTV, LGS, PRD, PUT, SBJ are complements.
3. Nonterminals with semantic tags ADV, VOC, LOC, PRP are adjuncts.
4. If none of the other conditions apply, the nonterminal is an adjunct.

The design of the complement-adjunct table is based on assumptions that we make about the Penn Treebank labeling as well as assumptions about linguistic principles guiding the formation of TAG grammars (Frank 2002). For example, according to the principle of localizing dependencies and making the assumption that a complementizer is a dependent of the predicate, we stipulate in the complement-adjunct table that labels of nodes in complementizer position are complements, not adjuncts. Hence, the WHNP complementizer in Figure 6 is localized in elementary tree α_1. Although this is the methodology that we use to design our particular instantiation of complement-adjunct table, note that it can be instantiated in other ways that do not necessarily follow these linguistic principles.

2.3 Extensions to the tree extraction procedure

The basic tree extraction procedure suffices to extract a TAG from the Penn Treebank. It is also deterministic in that only one multiset of TAG elementary trees is output given a particular bracketed sentence as input. However, there are deficiencies in the procedure insofar as some of the extracted trees do not quite conform to those guidelines for TAG formation imposed by linguistic theory and custom as discussed in section 2.1. In this section, we describe extensions to our algorithm that correct the more salient of these defects. In particular, we describe extensions of the algorithm that handle empty elements, predicative auxiliary trees, conjunction, co-anchors, and punctuation.

2.3.1 Empty Elements

Empty elements are terminal nodes in bracketings that do not represent lexical items. They have played an important part in transformational generative linguistic frameworks (Haegeman 1994). They are also found in the Penn Treebank and are usually represented in handwritten TAG grammars. Consequently, it is desirable to amend the tree extraction procedure to adequately represent them in the resulting extracted grammar. We discuss goals of the extraction procedure with respect to empty elements, introduce modifications in the tree extraction procedure to handle empty elements, and then describe how well the goals were achieved.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

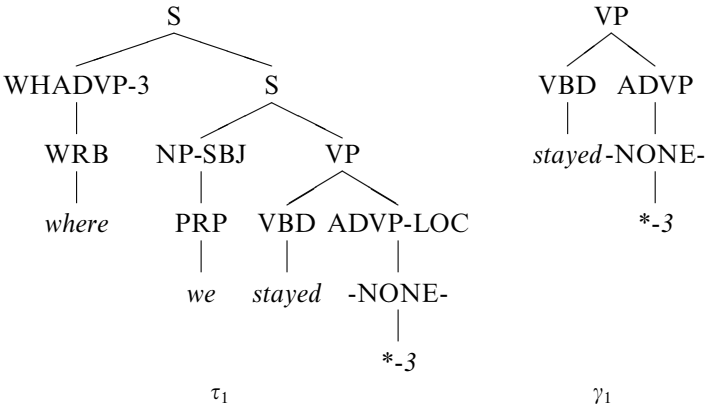


Fig. 4. Partial tree γ_1 is extracted from parse tree τ_1. Subtree dominating ADVP is grafted onto γ_1, even though ADVP is an adjunct, to help ensure that no extracted elementary tree is anchored by an empty element.

There are several goals behind our tree extraction procedure when dealing with empty elements. First, because the goal of the tree extraction procedure is to extract *lexicalized* trees, it follows that no extracted tree should be anchored by an empty element. Second, it is important to localize a trace with its landing site in the same elementary tree, if possible. This follows from the goal of localizing dependencies.

Modifications to the tree extraction procedure ensure that no extracted tree is anchored by an empty element. One such modification involves the head-finding procedure. Recall that this procedure chooses a head child η_h out of a set of daughter nodes $\eta_1, \ldots, \eta_m$ and their parent node η^p. It does this by consulting a head percolation table. We modify this procedure to preferentially choose a daughter that dominates a lexical item as follows:

Find_head(Parent η^p, Daughters $\{\eta_1, \ldots, \eta_m\}$)

1. Select child η_h out of $\{\eta_1, \ldots, \eta_m\}$ using head percolation table;
2. If $\forall \eta \in \{\eta_1, \ldots, \eta_m\}$, η does not dominate a lexical item, then return η_h;
3. If η_h does not dominate a lexical item, then return **Find_head**(η^p, $\{\eta_1, \ldots, \eta_m\} - \eta_h$);
4. Return η_h.

Another modification preferentially grafts partial trees anchored by empty elements onto partial trees anchored by non-empty elements, instead of factoring them into auxiliary trees, for example (see Figure 4).

These modifications are sufficient to ensure that if the input bracketing contains at least one leaf node that is a lexical item, then every elementary tree extracted by the tree extraction procedure will have a lexical anchor, as the following proof demonstrates.

Theorem

If the input bracketing contains at least one leaf node that is a lexical item, then any elementary tree extracted by the tree extraction procedure will have a lexical anchor.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

Proof
Suppose on the contrary that there is an extracted elementary tree γ that is anchored with an empty element. There are three cases. The first case is that γ substitutes into another elementary tree. But this is not possible because the extraction procedure would graft γ onto another elementary tree such that the empty element is not the anchor. The second case is that γ adjoins into another elementary tree. This is not possible either because it would be grafted onto another elementary tree. The only other possibility is that it neither adjoins nor substitutes into any other elementary tree. But this, too, is impossible. If it neither adjoins nor substitutes into another elementary tree, then the root node of γ must correspond to the root of the Penn Treebank bracketing. But then because the modified head detection algorithm preferentially chooses lexical heads over empty heads and because it is assumed each Penn Treebank bracketing always contains at least one lexical item, it can be shown that γ is lexicalized by induction on the depth of the tree corresponding to the Penn Treebank bracketing. □

Other modifications to the tree extraction procedure ensure that traces are localized with their landing sites. In this regard, it is useful to introduce the notion of a trace set. A *trace set* $\Delta(\gamma)$ corresponding to a partial tree γ is a set of traces that occur in γ for which landing sites have not been yet encountered. In this case, "not yet encountered" means that the landing site has not been seen in that part of the supertree ρ dominating γ such that ρ represents the part of the Penn Treebank bracketing that has been processed so far by the tree extraction procedure. Trace sets are computed for partial trees as the partial trees are being constructed by the extraction procedure.

Figure 5 gives an example of trace sets. Consider the extraction of the partial tree γ_0 that is anchored by the verb "eaten" from the parse tree τ_0 in Figure 5. The various stages in the construction of this partial tree are shown as γ_0^0, γ_0^1, and γ_0^2. When the trace "*-1" is incorporated into γ_0, "1" is added to $\Delta(\gamma_0)$. When its corresponding landing site NP-1 is incorporated into γ_0, "1" is deleted from $\Delta(\gamma_0)$.

The basic tree extraction procedure is modified in order to take into account trace sets. In particular, when this procedure is determining whether daughter nodes are to be either complements or adjuncts, a daughter node η_i that is a landing site for some trace in the trace set $\Delta(\gamma_h)$ of the current head partial tree γ_h is automatically labeled as a complement; it therefore becomes localized as a substitution node in γ_h.

2.3.2 *Predicative auxiliary trees*

Aside from recursion that results from modification, there is also recursion that occurs because of extraction out of clausal complements. See Figure 6 for an example. As it currently stands, the tree extraction procedure will not factor such recursion; i.e. it fails to produce any predicative auxiliary trees. These modifications need to be made in order to add this facility. Any time a partial tree γ_h grows in size, a check is made to determine whether γ_h meets criteria for it to be considered a predicative auxiliary tree. If so, it is factored out.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

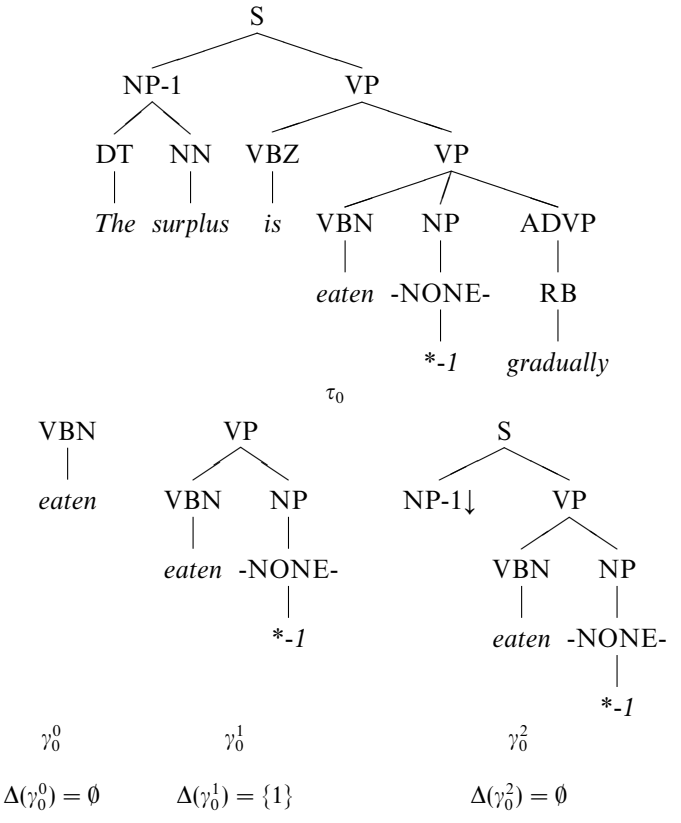


Fig. 5. Partial trees $\gamma_0^0, \gamma_0^1, \gamma_0^2$ and their corresponding trace sets $\Delta(\gamma_0^0), \Delta(\gamma_0^1), \Delta(\gamma_0^2)$ in different stages of the tree extraction procedure applied to the bracketing τ_0.

A partial tree γ_h is factored out into a predicative auxiliary tree if the following criteria are met. First, there must exist a substitution node η^s in γ_h that has the same label as the root node η^r of γ_h. The intention is that η^r will be the root of the predicative auxiliary tree while η^s will turn into its foot node. Second, the initial tree α which substitutes at η^s must contain some trace for which a landing site has not been found. This condition is required because of the desire to factor out only that recursion that occurs with extraction from a complement. It can be checked by examining the trace set $\Delta(\alpha)$.[5]

2.3.3 Conjunction

Without special treatment, the parse tree τ_2 in Figure 7 is decomposed into the elementary trees shown in that figure. If we want to follow a treatment of conjunction

[5] An alternative method for determining when to factor out a predicative auxiliary tree would be to check when sister nodes of the anchor that are complements have the same label as the root node. Either method would disallow a wrong prediction of factoring out a predictive auxiliary tree when extracting from a subject, for instance. The alternative would have difficulty, however, with the bracketing τ_1 in Figure 16.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

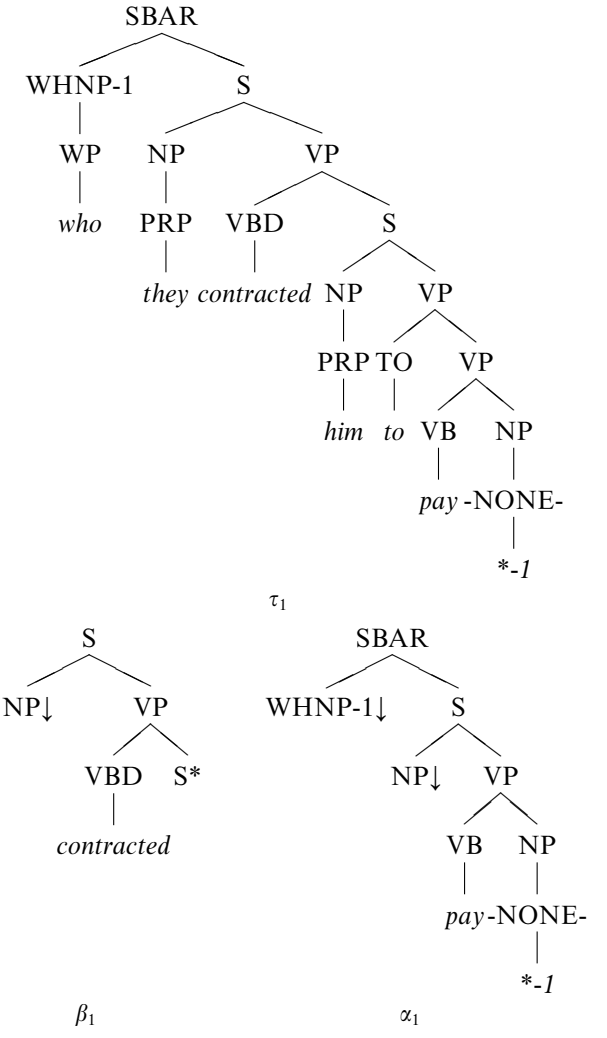


Fig. 6. The modified version of the extraction procedure can extract predicative auxiliary tree β_1 and localize the trace and landing site in α_1 given parse tree τ_1.

often found in TAG literature, we should prefer a decomposition shown in Figure 8. In order to obtain the latter, the following additional steps are prepended to the inductive step of the algorithm (i.e. before marking of complements and adjuncts). Recall that $\eta_1, \ldots, \eta_m$ is the sequence of daughters currently under consideration.

1. Identify a sequence of daughters $\eta_i, \ldots, \eta_j$, such that $1 \leq i < j \leq m$, representing conjuncts and their associated punctuation and conjunctions. The description of how the conjuncts are detected is described below. η_i is the leftmost conjunct. η_j is the rightmost conjunct. For example, given the node labels for $\eta_1, \ldots, \eta_m$ are "DT NN_1 CC NN_2 NN_3," the labels corresponding to $\eta_i, \ldots, \eta_j$ would be "NN_1 CC NN_2." See τ'_2 in Figure 8.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

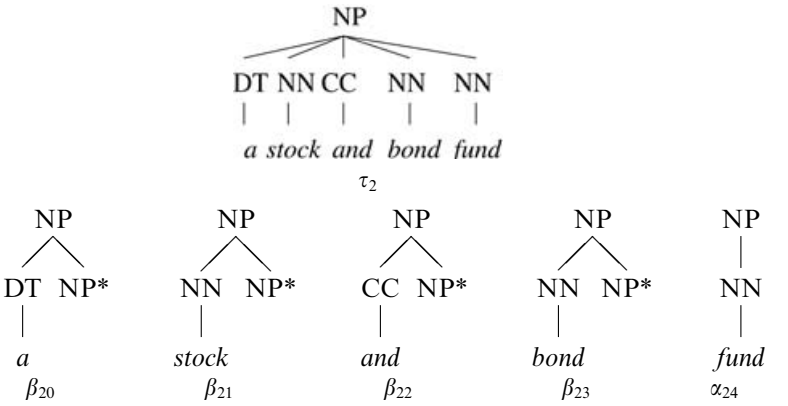


Fig. 7. Elementary trees $\beta_{20}, \beta_{21}, \beta_{22}, \beta_{23}$ and α_{24} are extracted from τ_2 using a version of the tree extraction procedure that does not specially treat conjunction.

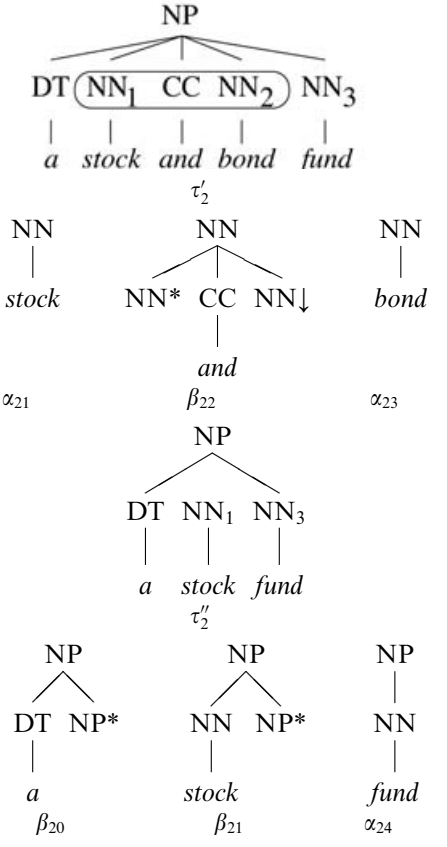


Fig. 8. τ_2' represents step 1 of conjunction processing. α_{21}, β_{22}, and α_{23} are the results of step 2. In step 3, default processing resumes on τ_2'', eventually resulting in β_{20}, β_{21} and α_{24}.

2. From the daughters $\eta_{i+1}, \ldots, \eta_j$, factor each conjunct and its associated punctuation or conjunction into its own auxiliary tree. The auxiliary tree is anchored with the punctuation or conjunction. It adjoins onto the leftmost conjunct

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

η_i. Conjuncts become initial trees that substitute into the auxiliary tree. Continuing the example, CC and NN_2 would be localized in an auxiliary tree that would adjoin onto NN_1. See α_{21}, β_{22}, and α_{24} in Figure 8.

3. Having now completed conjunction processing, default processing resumes on nodes $\eta_1, \ldots, \eta_i, \eta_{j+1}, \ldots, \eta_m$. In our running example, processing would continue (e.g. determining the head daughter, etc.) on those daughters that are labeled "DT NN_1 NN_3." See τ_2'' in Figure 8.

A sequence of daughters $\eta_i, \ldots, \eta_j$ is recognized as conjunction if it has the following characteristics. First, daughters representing conjuncts must share the same node label. Second, a punctuation mark–in particular a comma, a colon, or a semicolon–or a conjunction must lie between two such daughters that appear consecutively. As examples of these characteristics, note that labels corresponding to $\eta_i, \ldots, \eta_j$ may be "NP , NP CC NP" or "NP : NP : CONJP NP" but may neither be "NP CC VP" (a violation the first condition) nor "NP NP" (a violation of the second condition).

β_{20}, β_{21}, β_{22}, α_{23}, and α_{24} of Figure 8 are the trees that are extracted after special processing for conjuncts has been added to the tree extraction procedure. Note that the special processing has eliminated the difficulties mentioned earlier.

Another problem, one that is endemic to any TAG, is that it is sometimes difficult to represent TAG's usual domain of locality using TAG elementary trees when certain cases of conjunction are considered. This is the case for example with VP conjunction involving gapping in the sentence "Ulrich raised and bred marmosets" which has only two NPs which must be shared between two transitive verbs. Such examples motivate Sarkar and Joshi (1996) to propose a more elaborate solution involving the introduction of an extension of TAG that introduces a new operation *conjoin*. It has not been implemented in any broad-coverage TAG, however, because of the massive ambiguities that it would introduce for a parser (XTAG-Group 2001).

2.3.4 Co-anchors

Although TAG elementary trees typically have only one anchor (XTAG-Group 2001; Frank 2002), and the tree extraction procedure as it has been described up to now only extracts those kinds of trees, there are situations where more than one anchor per elementary tree is desirable. Such a case occurs when two trees are composed by a TAG operation but their meaning is not strictly compositional. For an example, see Figure 9.

We modify our tree extraction procedure to recognize certain violations of compositionality of meaning and in those cases to extract a multiply anchored tree. The violations that we consider are either those where the anchor is a particle and a sister to the main anchor, or those where the anchor is a preposition projecting to a complement-marked PP that is a sister to the main anchor. Hence, the multiply anchored tree α_3 in Figure 9 is produced, instead of α_4 and α_5. Note that because only prepositions or particles are allowed as co-anchors, representing idioms as multiply anchored elementary trees is not possible although this is the usual way TAG represents idioms (XTAG-Group 2001).

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

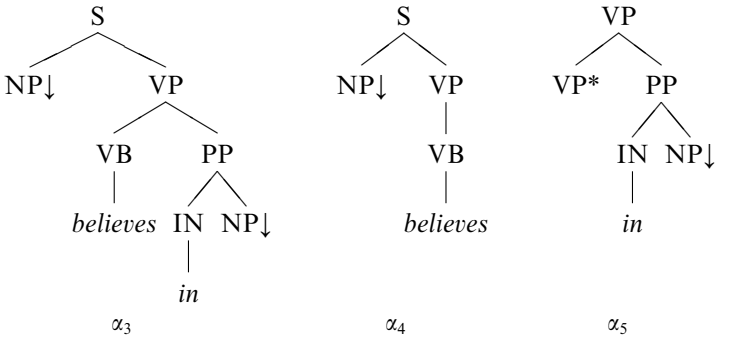


Fig. 9. α_3 is a multiply anchored elementary tree; α_4 and α_5 are singly anchored elementary trees which, when composed, have a different meaning than α_3.

2.3.5 Punctuation

It has been seen that commas are used as anchors for elementary trees representing conjuncts when they are found in conjunctive structures. There are numerous other cases of punctuation that are found in the Penn Treebank.

Paired punctuation occurs when two separate punctuation symbols are used to delimit a phrase. An open quote followed by its corresponding closed quote is an example of this. It is difficult to form a coherent strategy for dealing with paired punctuation because we have noticed that they are frequently beyond the TAG domain of locality. Consequently, whether the punctuation is paired or not, we take a uniform approach of factoring each punctuation symbol into its own auxiliary tree. This can be done easily by marking each punctuation symbol as an adjunct in the complement adjunct table.

3 Parameterization

Localizing dependencies and factoring recursion are the principal themes of TAG design. The tree extraction procedure of section 2 can be parameterized in ways that affect the kind of grammar that is extracted while still being in concordance with these themes. In this section, we describe a selection of these parameterizations.

The method that is used to distinguish complements from adjuncts plays an important role in the localization of dependencies. There are different approaches for this task, one being described in section 2. Call this CA1. We distinguish this against another approach (call this CA2) that is used by Xia (1999). Like CA1, CA2 employs heuristics that are based on local structural information in order to distinguish a given input node as either a complement or an adjunct. Also, both use label and semantic tag information to make this decision. One difference is that whereas CA1 uses the information from the input node and its parent to make its decision, CA2 uses information from the input node and its sister head node. Another difference is that CA2 also looks at the distance between the input node and the head node when deciding. In practice, CA2 will mark nodes more liberally as

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

adjuncts.[6] For example, whereas CA1 would mark an input node as a complement, CA2 would mark the same node as an adjunct because it is too far from the head node. Also, CA1 may mark an input node labeled with CLR as a complement whereas CA2 always marks such nodes as adjuncts.

Another parameter in the extraction procedure is one that determines how empty elements that are found in the Penn Treebank should be represented in the extracted grammar. Normally, a TAG grammar includes only certain kinds of empty elements, such as a trace representing syntactic movement of a complement, but not others, such as a placeholder representing currency values (e.g. euro, dollar, yen). Therefore, we can envisage two potential policies regarding empty elements. The first policy is to include all of these kinds of elements in the extracted grammar. Call this ALL. The alternative is to only include those kinds of elements whose syntactic category is recognized as a complement and hence must be localized in an elementary tree in accordance with our principles of TAG formation. Call this SOME.

A third parameter defines a label set strategy, which determines how nodes in elementary trees of an extracted grammar are to be labeled. The set of nonterminal and terminal labels that is used to label nodes in the Penn Treebank is quite extensive (Bies, Ferguson, Katz and MacIntyre 1995). Call this FULL. In contrast, the label set used in the XTAG grammar is less detailed (XTAG-Group 2001), in part because some Penn Treebank labels are captured by the structure of TAG elementary trees. Call this MERGED.

4 Evaluation of grammar variations

One criterion for evaluating extracted grammars is their size. The grammars' sizes in terms of the number of tree frames (lexicalized trees minus their anchors) and lexicalized trees are shown in Table 4. All of the extracted grammars are relatively large when compared to the hand-crafted XTAG grammar; the smallest extracted grammar has about 3000 tree frames whereas the XTAG grammar has 500. There is a large difference between extracted grammars in terms of number of tree frames; the largest grammar has more than twice the number of tree frames than the smallest. On the other hand, there is a smaller difference between grammars in terms of lexicalized trees.

A finer-grained analysis of the factors that influence the size of extracted grammar can be performed by computing how much grammar size varies when the parameterization of grammar is systematically changed. This can be quantified by computing the variance of grammar size.[7] Grammar size in terms of tree frames is most affected by choice of label set strategy; variance due to label set strategy is more than twice as large as variance due to complement detection strategy or empty

[6] The procedure to detect complements and adjuncts in Xia (1999) is parameterized. The specific parameterization investigated by Xia (1999) is what we mean by CA2. Other parameterizations of the same procedure will yield different behaviors.

[7] The variance $Var(S)$ of a set S of samples $\{s_1, \ldots, s_N\}$ is defined to be $Var(S) = \frac{1}{N-1} \sum_{i=1}^{N} (s_i - E(S))^2$, where $E(S)$ is the expected value, or mean, of set S. In particular, $E(S) = \frac{1}{N} \sum_{i=1}^{N} s_i$.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

Table 4. *Size of various extracted grammars in number of tree frames and number of lexicalized trees*

Comp Adjunct	Empty Elements	Label Set	Grammar Size	
			Frames	Lexicalized Trees
CA1	ALL	FULL	8675	113456
CA1	ALL	MERGED	5953	109744
CA1	SOME	FULL	7446	110134
CA1	SOME	MERGED	5053	106457
CA2	ALL	FULL	6488	111034
CA2	ALL	MERGED	4358	107285
CA2	SOME	FULL	4723	106422
CA2	SOME	MERGED	3075	102679

Table 5. *Coverage of various extracted grammars. Coverage is in terms of percentage of tree frames and lexicalized trees in the test corpus. Missed coverage is divided into W, T seen separately – a word and tag were both seen in the training corpus, but not together and either W, T not seen – either the word or the tag or both were not seen in the training corpus.*

Comp Adj	Empty Elemt	Label Set	Coverage		Missed Coverage	
			Frames	Lex Trees	W,T seen separately	W,T not seen
CA1	ALL	FULL	99.56	92.04	5.09	2.87
CA1	ALL	MERGED	99.69	92.35	4.89	2.76
CA1	SOME	FULL	99.65	92.41	4.80	2.79
CA1	SOME	MERGED	99.77	92.74	4.58	2.68
CA2	ALL	FULL	99.65	92.26	4.96	2.79
CA2	ALL	MERGED	99.76	92.59	4.73	2.68
CA2	SOME	FULL	99.81	92.76	4.60	2.64
CA2	SOME	MERGED	99.88	93.08	4.35	2.57

elements strategy. The latter two are about the same. In contrast, grammar size in terms of lexicalized trees is most affected by the empty elements strategy, followed by label set strategy and complement detection strategy. The latter two have about 20% and 80% less variance, respectively, than for the empty elements strategy.

Another criterion for evaluating extracted grammars is coverage on unseen data. Table 5 shows the results of evaluating the various extracted grammars according to their breadth of coverage on held out data (section 23 of the Penn Treebank). The coverage is very good; more than 99% of the time, a tree frame in the unseen data is also found in the extracted grammar. It indicates that regardless of its parameterization, an extracted grammar captures the most commonly used syntactic structures from the training corpus.

In contrast, the grammars poorly cover the unseen data in terms of lexicalized trees. There, the grammars exhibit about 92–93% coverage. This is not unexpected;

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

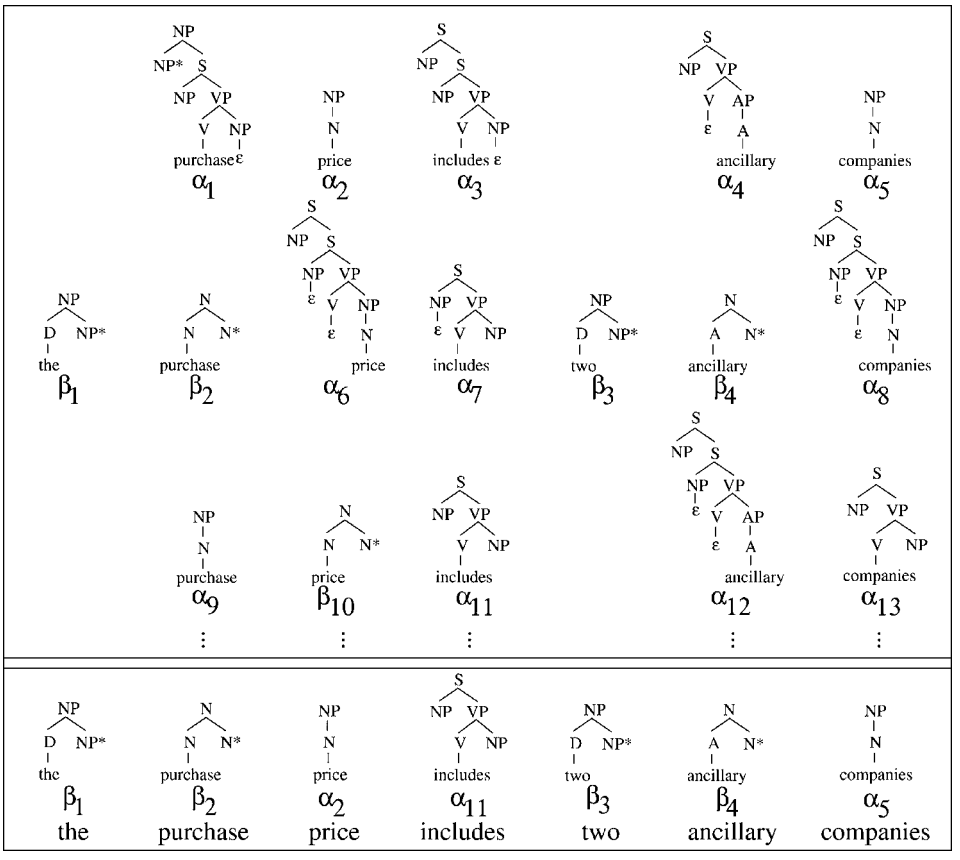


Fig. 10. A selection of the supertags associated with each word of the sentence "the purchase price includes two ancillary companies."

because lexicalized trees are combinations of lexical items and tree frames, their number should far exceed the number of tree frames alone. Looking at the variances of the different parameters, we see that the complement detection parameter affects coverage the least while the empty elements parameter affects coverage the most. This mirrors variances in grammar size in Table 4.

Further inspection of Table 5 reveals that 4–5% of the missed coverage is due to cases where the word and the supertag have been seen in the training corpus, just not together. Therefore, a hypothetical procedure that could infer the existence of these missing word-supertag combinations from the training corpus would boost coverage by 4–5% in absolute terms. We will investigate this issue later.

Supertagging is the task of assigning a supertag to each word of an input sentence such that the supertagged sentence can be composed via TAG operations to form the correct parse tree for the sentence. It is a disambiguation process because any given word of a language may typically be associated with tens or hundreds of different supertags, out of which the supertagger must choose the most appropriate one given that the word appears in a particular position in a particular sentence.

In fact, Bangalore and Joshi (1999) calls supertagging *almost parsing* because it greatly reduces parsing ambiguity much like any statistical parser (Sarkar 2000;

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

Table 6. *Supertagging accuracy of various grammars. Supertagging accuracies for 1-best supertagger and 30-best supertagger are shown*

Comp Adj	Empty Elemt	Label Set	% Supertag Acc	
			1-best	30-best
CA1	ALL	FULL	78.55	91.34
CA1	ALL	MERGED	79.23	92.25
CA1	SOME	FULL	79.34	91.80
CA1	SOME	MERGED	80.09	92.80
CA2	ALL	FULL	79.07	91.62
CA2	ALL	MERGED	79.65	92.61
CA2	SOME	FULL	80.03	92.27
CA2	SOME	MERGED	80.62	93.31

Chen *et al.* 2002a). Consider for example the sentence "the purchase price includes two ancillary companies." As shown in Figure 10, each word in the sentence may be associated with many different supertags. In parsing this sentence, a TAG parser would need to consider combining each word with every supertag (the top part of Figure 10) as well as composing the resulting supertag combinations together to form a parse tree. Alternatively, if it is given a supertagged sentence (the bottom part of Figure 10), the TAG parser has less to take into consideration.

A third method for assessing the extracted grammars is to evaluate the supertagging accuracies of models based on each of the grammars. In particular, we choose to evaluate grammars using the trigram model, which is much like the trigram model for part of speech tagging. We choose this model because of its simplicity with respect to other supertagging models and also other stochastic parsing models. In particular, because of its simplicity, we can directly focus on smoothing the probability distributions in this model, rather than for example focusing on tweaking the model by proposing four or five different levels of backoff. The results of this assessment are shown in Table 6. Note that the most pronounced effect is that all of these grammars' corresponding models have supertagging accuracies around 80% whereas the XTAG grammar has a supertagging accuracy of around 91% (Chen, Bangalore and Vijay-Shanker 1999).[8]

Again we analyze the effect of parameterization. The empty elements parameter affects supertagging accuracies the most. The complement detection parameter affects them the least. One might expect the latter to have a greater effect because more labels marked as complements means that the extracted elementary trees may have a larger domain of locality, and it is precisely these kinds of trees that supertagging models find hard to distinguish. Even though they may be extracted from the training corpus, however, their occurrence is typically so rare that they are unlikely to be seen again in any test corpus. There is a very strong direct relationship between the coverage of a grammar in terms of lexicalized trees (more so than tree frames) and the supertagging accuracy of its corresponding model. Coverage appears to be

[8] There is more of a sparse data problem with extracted grammars than with hand-designed grammars.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

Table 7. *Ability of various extracted grammars to localize traces with their landing sites*

Comp Adjunct	Empty Elements	Label Set	Frame Types		Frame Tokens	
			Traces	Null Elements	Traces	Null Elements
CA1	ALL	FULL	2381	2560	21508	42593
CA1	ALL	MERGED	1685	1899	21508	42593
CA1	SOME	FULL	1258	1392	18394	25305
CA1	SOME	MERGED	871	1002	18394	25305
CA2	ALL	FULL	1847	2130	21458	42643
CA2	ALL	MERGED	1258	1538	21458	42643
CA2	SOME	FULL	589	655	16153	24035
CA2	SOME	MERGED	351	409	16153	24035

more consequential to supertagging accuracy than the ability to condition on more detailed information that a larger grammar (with poorer coverage) would provide. Accordingly, because the empty elements parameter affects coverage the most, it also has the greatest effect on supertagging accuracy. Now consider the label set parameter. Earlier we commented that the Penn Treebank has labels that might be considered redundant because they make structural distinctions that are encoded in the elementary tree structure of TAG. The supertagging results provide evidence that accuracy is not lost if these labels are simplified; comparing MERGED versus FULL versions of models that are otherwise the same, there is only one case where the 1-best supertagging accuracy goes down.

The fourth method that we use to evaluate the extracted grammars is in terms of the number of traces that they localize with their landing sites. Table 7 counts the frequency of empty elements in the set of tree frames and in the treebank. "Frame Types" is a count of the former while "Frame Tokens" is a count of the latter. "Traces" refers to empty elements which the grammar localizes with their landing sites in the same tree frame; "Null elements" refers to empty elements that do not have corresponding landing sites. The parameter of complement detection makes the most sizeable difference in terms of the number of tree frame types that localize traces with their landing sites. But these extra types that are created are quite rare. The parameter of empty elements makes the next largest difference. These extra types occur more frequently. Sixty percent of trace tokens in the Penn Treebank are localized by the grammars CA1-ALL-FULL and CA1-ALL-MERGED, which perform the most of this kind of localization out of all of the various extracted grammars.

The final method that we use to evaluate extracted grammars is aimed at determining how many of those complements that are automatically extracted are indeed complements. We adopt as a gold standard those complements that are labeled in the PropBank, a semantically annotated version of the Penn Treebank. In contrast, most examples of previous work (Buchholz 1998; Xia *et al.* 1998;

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

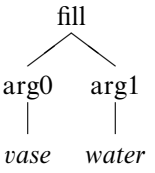


Fig. 11. PropBank-style semantic representation for both *The vase fills with water* and *Water fills the vase.*

Collins 1999; Chiang 2000; Merlo and Leybold 2001; Hockenmaier and Steedman 2002b) take as the gold standard the application of heuristics applied over the Penn Treebank. In our preliminary evaluation, we evaluate the performance of two grammars CA1-SOME-FULL and CA2-SOME-FULL. This is because we consider the complement detection strategy the most interesting to evaluate since it is the evaluation of two different kinds of heuristics. In contrast, the empty elements strategies are not heuristic in nature and the label set strategies have no effect on complement detection.

The PropBank (Kingsbury, Palmer and Marcus 2002) overlays the syntactically annotated Penn Treebank with semantic predicate argument information. Each word corresponding to a predicate is assigned a sense tag, and each complement and adjunct is assigned a local semantic label. Complement labels consist of numbers from 0 to 5 which are used consistently across syntactic alternations for the same verb meaning (see Figure 11). Adjuncts are given special tags such as TMP or LOC derived from the original annotation of the Penn Treebank.

There are several caveats about our evaluation. First, we are using a pre-release version of the PropBank where not all of the verbs have been annotated with semantic role information. In particular, we are using the same version of the corpus as is used in Chen and Rambow (2003) and Gildea and Hockenmaier (2003). Although the annotation is incomplete, all of the instances of the most common 1600 verbs have been semantically annotated. Second, we are ignoring any evaluation of head detection. This is the source of some irregularities in our evaluation. In particular, our tree extraction procedure assumes that given a set of siblings in the treebank, one sibling is selected as the head. In contrast, the PropBank may consider all of the siblings as complements of another head. This is the case in, for example, small clause constructions. Contrast the extracted trees in Figure 12 with the PropBank semantic representation in Figure 13. Third, we are evaluating complement detection only, not detection of adjuncts. Addressing this issue, we would like to stress that this is a preliminary investigation. Furthermore, we argue that detection of PropBank-labeled adjuncts is easier than detection of complements. In particular, the labeling of adjuncts in the PropBank appears to closely follow Penn Treebank semantic tag labeling. For example, whenever there is a node marked as TMP or LOC in the Penn Treebank, the node is marked as TMP or LOC, respectively, in the PropBank. Since nodes with Penn Treebank semantic tags such as TMP or LOC are always marked as adjuncts by CA1 or CA2, it would seem that CA1 and CA2 would achieve high accuracy in the labeling of adjuncts, and also that there would be not much difference in their accuracies.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

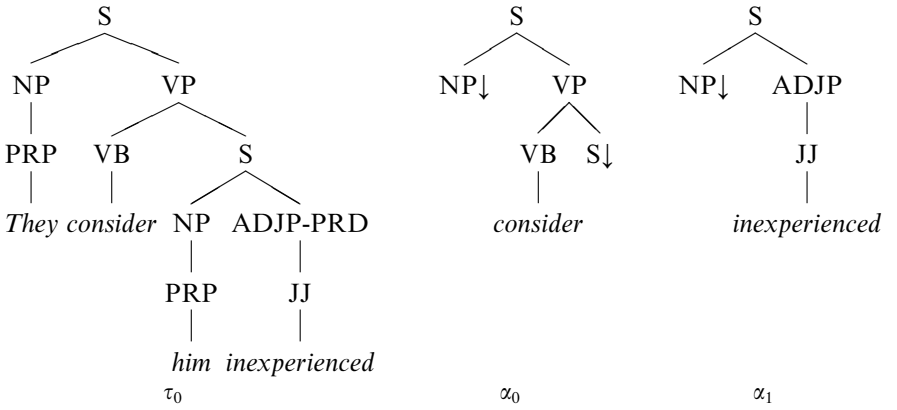


Fig. 12. Elementary trees α_0 and α_1 are extracted from τ_0. Note that "inexperienced" but not "him" is among the complements of "consider."

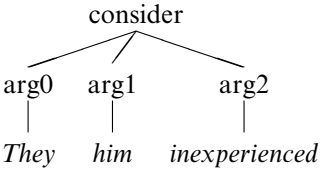


Fig. 13. PropBank semantic representation for *They consider him inexperienced*. Note that both "inexperienced" and "him" are complements of "consider."

Our evaluation is performed over sections 02–21 of the PropBank. One may consider several alternative forms of evaluation. The first is to determine true complements out of all complements detected by the tree extraction procedure. The second is to make this determination out of all of the complements that are found in the PropBank. We choose a variant of the latter because the former ideally requires a fully labeled version of the PropBank. In particular, our evaluation assumes that each trial is an instance of a PropBank complement corresponding to a complement of some elementary tree as determined by our tree extraction procedure. Every instance is checked to see whether the anchor of the elementary tree corresponds to the complement's predicate as marked in the PropBank.

This check is useful because it allows us to distinguish PP complement versus adjunct ambiguity. The tree extraction procedure marks the NP object of the preposition as a complement in both of these cases. Only if heuristics decide that the PP is an argument does the tree extraction procedure localize the NP object and its predicate in the same elementary tree. This check is also is useful in general because it is an evaluation of the extracted grammar's consistency in terms of mapping to a semantic representation.

The results are shown in Table 8. Columns of the table correspond to extracted grammars CA1-SOME-FULL and CA2-SOME-FULL. Rows correspond to various evaluations. The first row, *Arg-in-domain*, counts the cases where the PropBank and the extracted grammar agree that a predicate has a certain complement. The second and third rows count cases where they agree that there is a complement, but where

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

Table 8. *Evaluation of complement detection strategies CA1 and CA2 using the PropBank as a gold standard*

	CA1-SOME-FULL		CA2-SOME-FULL	
	Instances	Percentage	Instances	Percentage
Arg-in-domain	88458	86.7%	81140	81.7%
PP-out-domain	8028	7.88%	13452	13.5%
Other-out-domain	5438	5.34%	4759	4.79%
Total	101924	100%	99351	100%

the extracted grammar does not localize it in an elementary tree whose anchor is the complement's PropBank predicate. The second row, *PP-out-domain*, counts only those cases corresponding to PP-attachment ambiguity. The third row, *Other-out-domain*, counts the remaining cases.

Table 8 shows several items of interest. First, CA1 performs much better than CA2 in detecting complements. Recall that CA2 differs from CA1 in that CA2 automatically marks nodes far from the head as well as those nodes marked as CLR as adjuncts. Our evaluation shows that these distinctions are not helpful in identifying complements. In this regard, we agree with Merlo and Leybold (2001) that those nodes marked as CLR should be marked as complements. Second, in about 5% of the cases (*Other-out-domain*), it is not possible for either CA1 or CA2 to detect a complement such that it is localized in the same elementary tree as its predicate. In a manual evaluation of 80 of these cases, we find that about half of them (2.5% of the total) are cases of control or small clause constructions, an example of which was given previously in Figures 12 and 13. The other cases include co-indexation of complements across clausal and VP conjunction, or across sentential adjuncts. The last row, *Total*, gives the total number of instances that were evaluated for each grammar. The numbers for CA1-SOME-FULL and CA2-SOME-FULL do not match. One reason for the discrepancy is that given the parameterization SOME for both of the grammars, use of CA1 will include more empty elements in the resulting extracted grammar than use of CA2.

5 Inducing higher level organization of extracted TAGs

Section 2 describes a method to extract a TAG grammar from a treebank. There is little structure to the resulting TAG that is generated; it merely consists of a set of tree frames. In contrast, complex handwritten TAGs often have their tree frames grouped into hierarchical subsets according to grammatical properties shared between tree frames (XTAG-Group 2001). In this section, we will explain why this high level organization of a TAG is important. We will then introduce two distinct approaches to inducing this higher level organization from our extracted grammars.

Without a high level organization of TAG, a TAG consists of nothing more than a set of tree frames. If the TAG represents a broad coverage grammar, this set of tree frames can be very large. Use of such a large grammar incurs various difficulties. For

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

example, as noted by Xia, Palmer, Vijay-Shanker, and Rosenzweig (1998), grammar maintenance and development is hindered because it is difficult to check the large grammar for inconsistencies. Also, in a complete natural language system, a grammar does not stand in isolation, but needs to be related to other components, such as a semantic component. A high level organization of TAG eases this task. Moreover, it can facilitate the manual development of a TAG grammar (by more abstractly pinpointing, for example, omissions in the manually developed grammar that exist in the extracted grammar), and also facilitate comparison between different kinds of TAGs, TAGs of different languages (Xia *et al.* 2000), or grammars of different formalisms, for example.

Smoothing sparse data is one potential application of a high level organization of TAG that we will consider in more detail in section 6. As revealed in section 4, sparse data is a serious problem for extracted grammars. Lexicalized tree coverage is particularly problematic. It is possible to use the high level organization of TAG to infer the presence of unseen lexicalized trees from a seen lexicalized tree. For example, given that the word "hit" appeared transitively, we infer that it can also appear in a passive construction, given that the transitive and the passive tree frames are related in the organization.

Upon examination of these tree frames for a typical broad coverage grammar, it is apparent that many tree frames share structural similarities. These similarities reflect the recurrence of the same grammatical properties in different tree frames. We will describe two distinct approaches to organize tree frames according to their grammatical properties. One approach relies on hand-coding of regularities in how syntactic phenomena are annotated. The other approach is more automated, using distributional similarity in order to induce groupings of related tree frames.

5.1 Feature vector based tree families

In this section, we describe a procedure that induces tree families from a feature vector representation of an extracted TAG. First, we describe the feature vector representation. Second, we describe tree families. Third, we describe the mapping from one onto the other.

A feature vector is a set of attribute-value pairs. We will also call attributes or attribute-value pairs *features*. Each attribute-value pair represents a linguistically salient aspect of the tree frame. The goal of our procedure is to take as input a tree frame and then output a feature vector. The features in this feature vector representation include part of speech, subcategorization, maximal projection, modifyee, direction of modification, co-anchors, and also features that capture transformational relationships. A list of all features and possible feature values are listed in Table 9.[9] This feature vector representation has many uses. For example, it can be used as an intermediate representation in mapping an extracted grammar onto a semantic representation.

[9] In the table, *Topicalized-X* stands for multiple features, each one instantiated with a different node label for X. Similarly, *wh-movement-X-Y* also stands for multiple features.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

Table 9. *List of each feature in a feature vector along with possible values*

Feature	Values
Part of speech	DT, NN, VB, RB, ...
Subcategorization	{NP}, {NP, S}, ∅, ...
MaxProj	S, NP, VP, ...
Modifyee	NP, VP, S, ...
Direction	LEFT, RIGHT
Co-anchors	{of}, {by}, ∅, ...
Declarative	TRUE, FALSE
Empty Subject	TRUE, FALSE
Complementizer	TRUE, FALSE
Passive	TRUE, FALSE
By-Passive	TRUE, FALSE
Topicalized-X	TRUE, FALSE
Wh-movement-X-Y	TRUE, FALSE
Topicalized-VP	TRUE, FALSE
Subject-Aux Inversion	TRUE, FALSE
Relative Clause	TRUE, FALSE

In general, feature values are obtained for each tree frame in the grammar using linguistically motivated principles in a straightforward manner. For example, the feature *subcategorization*, referring to the subcategorization frame, basically consists of finding the substitution nodes or foot node that are sisters to the preterminal node of a tree frame. Values for features that refer to transformations, such as *Topicalized-X* are obtained by pattern matching a given tree frame t against an underspecified tree-representation that describes the constraints that must hold for the transformation to be said to apply to t. Details of decomposition of tree frames into feature vectors can be found in Chen (2001).

XTAG contains a number of hand-crafted tree families (XTAG-Group 2001). A tree family is a set of transformationally related tree frames that contains roughly the same predicate argument structure. For example, a tree frame representing a transitive declarative syntactic construction will be grouped into the same tree family as another representing a transitive relative clause construction.

We can define tree families for our extracted grammar in terms of the feature vector representation. We define a tree family to be a set of tree frames such that the feature vector corresponding to each tree frame in the set meets these conditions, with one exception. There are two conditions. First, it must share the same values for the *part of speech* feature, the *subcategorization* feature, and the *co-anchor* feature as every other feature vector of a tree frame that is found in the same set. Second, it must have at least one transformational feature that has a non-null value. The exception is that co-anchor "by" is deleted from the *co-anchor* feature of a tree frame that has its *By-Passive* feature activated. This exception is intended to ensure, for example, that a passive transitive tree frame containing a by-passive phrase is included in the same tree family as a declarative transitive tree frame, even though the latter tree frame does not contain "by" as a co-anchor.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

5.2 Organizing trees via distributional similarity

In the previous section, a method for inducing feature vectors from tree frames was introduced as a means of constructing a higher level organization of TAG from a base set of tree frames. This method has several potential disadvantages. First, the kinds of features to be included are decided manually. Although this decision process includes linguistically significant features, it might be possible that an important feature is excluded, or that a different feature decomposition may be more effective. Second, there is manual labor involved in creating rules to map tree frames to features. These rules may be needed to be re-created or modified, manually, if a different treebank based on different linguistic assumptions were used, albeit many of the rules are based on widely-held linguistic assumptions.

In this section, we introduce a method to automatically induce higher level structure from a base set of tree frames using the technique of distributional similarity. Similarity measures have been used in various fields of statistical natural language processing including information retrieval (Manning and Schütze 1999) and smoothing for language modeling (Dagan, Lee and Pereira 1999; Lee 1999).

Recall that a tree family is a set of tree frames containing roughly the same predicate argument structure. Given a lexical item is associated with one tree frame in a tree family, it can often be associated with the other members of that tree family. For example, given the word "understands" appears with a transitive declarative tree frame, it can appear with other tree frames in the transitive tree family which represent transformationally related syntactic constructions like relative clause or topicalization. Speaking in terms of distributional similarity, we can posit that two tree frames are similar if and only if they tend to appear with the same lexical items.

Here, we experiment using the similarity measure of Jaccard's coefficient.[10] In Lee (1999), Jaccard's coefficient is the second highest performing measure, and has the added advantage of being computationally inexpensive to compute.

In our application, Jaccard's coefficient $jac(t, t')$ defines the similarity between two supertags t and t'. Let WT be the set of words that are associated with supertag t in the training corpus. Likewise, let WT' be the set of words that are associated with supertag t'. Jaccard's coefficient is defined as follows:

$$jac(t, t') = \frac{|WT \cap WT'|}{|WT \cup WT'|}$$

The similarity measure is defined from Jaccard's coefficient:

$$sim(t, t') = \begin{cases} jac(t, t') & \text{if } jac(t, t') > \tau \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

τ is simply a cutoff value that ignores all of the tags t' whose similarity to t is below the (constant) threshold.

$S(t_i)$ is the set of supertags that are most similar to t_i. It is possible to set $S(t_i)$ to be the set of all supertags, but it is usually better to set it to some relatively

[10] The similarity measure of α-skew divergence was also tried. It was found to produce a minimal difference in results. See Chen (2001) for more information.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

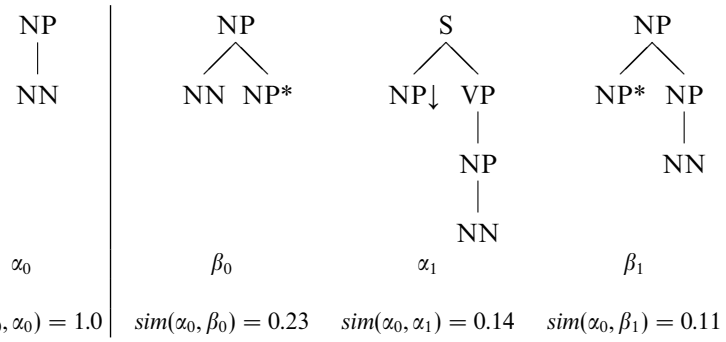


Fig. 14. Most similar supertags to a noun phrase argument supertag (α_0) according to Jaccard's coefficient.

low-valued constant number of supertags for the sake of efficiency. Let $\rho = |S(t_i)|$ – the size of the "radius" of nearby supertags.

5.2.1 Examples of similar supertags

We give several examples of most similar supertags as chosen by Jaccard's coefficient. We set the parameters to be $\tau = 0.1$, $\rho = 40$.

Consider first the most similar supertags to a supertag that represents a noun phrase argument (α_0 of Figure 14). The most similar supertags to α_0 according to Jaccard's coefficient are listed in Figure 14. Note that these supertags are intuitively good candidates for smoothing purposes. For example, given the word "train" appears as a noun modifier (β_0) as in "the train station," it would make sense that it could also appear as an NP argument as in "the large train." These supertags are not transformationally related. Also note that different pairs of supertags have different similarities, a fact that will prove useful in smoothing.

Now consider the most similar supertags to a transitive declarative verb supertag (α_3 in Figure 15). Some of the most similar supertags to α_3 according to Jaccard's coefficient are shown in Figure 15. Notice that these supertags are transformationally related. For example, α_4 represents an "empty subject" transformation, and β_4 represents a "relative clause" transformation. Thus, distributional similarity as applied to supertags seems to be a method to automatically induce tree families, which have hitherto been manually defined.

6 Smoothing using high-level organization of TAG

In section 5, several methods are introduced in order to induce a higher level organization of TAG. As discussed in that section, there are many potential applications of such a higher level organization, including the potential to assuage the sparse data problem. In section 4, it is seen that sparse data seems to be a critical issue for automatically extracted TAGs. For this reason, we conduct experiments to determine the effectiveness of using higher level organization of TAG in order to ameliorate sparse data in supertagging models.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

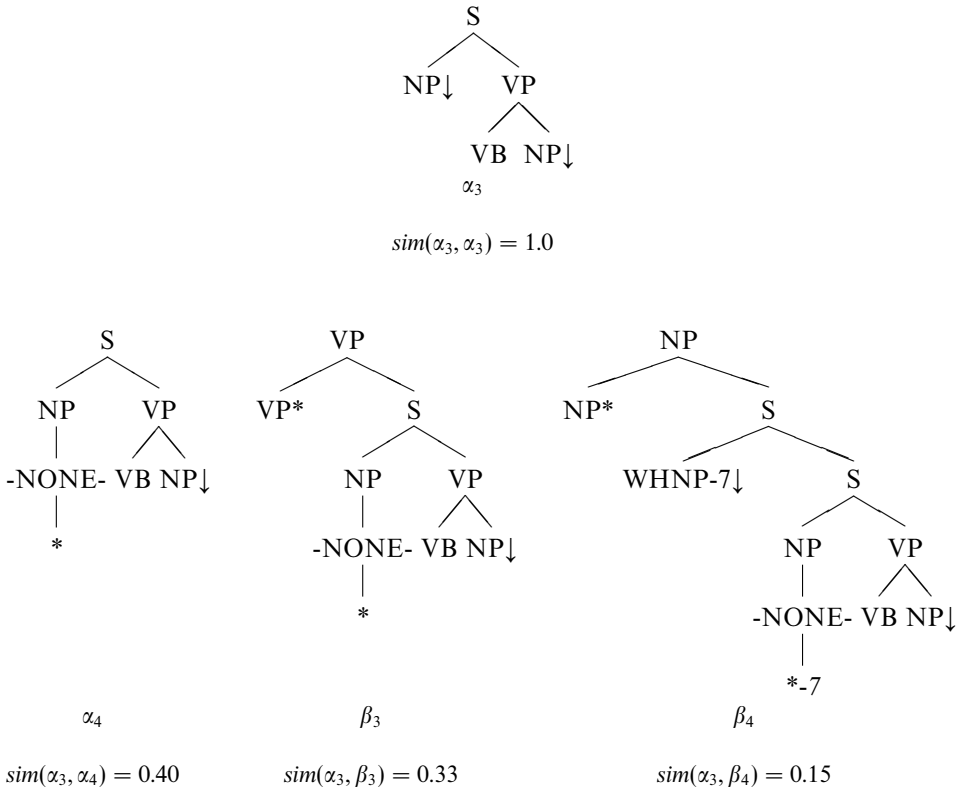


Fig. 15. Some of the most similar supertags to a transitive declarative tree supertag (α_3) according to Jaccard's coefficient.

Our objective is to improve the performance of the trigram model for supertagging by smoothing one of its probability distributions. The trigram model requires the estimation of several such probability distributions: $P(w_i|t_i)$ representing the probability that the ith word of the sentence is w_i given that it is the anchor of supertag t_i, and $P(t_i|t_{i-1}t_{i-2})$ representing the probability that the ith supertag in the sentence is t_i given that the two previous supertags are t_{i-1} and t_{i-2}. In this section, we will be concerned exclusively with smoothing $P(w_i|t_i)$, assuming that $P(t_i|t_{i-1}t_{i-2})$ is smoothed using Katz' backoff (Katz 1987), even though we acknowledge the possibility that smoothing techniques akin to the ones we are using may also be used to smooth $P(t_i|t_{i-1}t_{i-2})$.

In this section, the grammar CA1-SOME-FULL is used, where tree frames with co-anchors are allowed. Unless otherwise noted, training data is sections 02–21 of the Penn Treebank, tuning data is section 22, and test data is section 23.

The outline of this section is as follows. We test several variations of the trigram supertagging model which are intended to serve as baselines to which we can compare the smoothing techniques currently under investigation. We subsequently introduce and evaluate two smoothing procedures, one based on the semi-automatic induction of high level organization of TAG, and the other based on an organization induced using distributional similarity methods.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

6.1 Baselines

This section establishes the sparse data problem that we are going to investigate. We explain exactly what probability distribution we are going to smooth. We then provide three baselines for comparison: an unsmoothed trigram model of supertagging, a model smoothed using part of speech information, and a model trained using both the training set and the test set.

We employ the n-best supertagging model described in Chen, Bangalore and Vijay-Shanker (1999). It is a promising approach to make parsing more efficient and accurate. It uses the same probability model as the trigram supertagging model, with certain alterations to the Viterbi algorithm to return the n-best tags per word. In these experiments, we vary n between 1 and 10. Furthermore, results are most often reported for specific kinds of instances of test data. Category "w_i, t_i together" are those instances in test data where $P(w_i, t_i) > 0$.[11] Category "w_i, t_i separate" are those instances where $P(w_i, t_i) = 0$ but $P(w_i) > 0$ and $P(t_i) > 0$. Category "w_i unseen, t_i seen" are those instances where $P(w_i, t_i) = 0$ and $P(w_i) = 0$ but $P(t_i) > 0$.

We are concerned with smoothing $P(w_i, t_i)$ only. There are several reasons why sparse data exists for this probability distribution. One reason is that the word w_i may not be seen in the training data. Because a high level organization of TAG cannot straightforwardly help smoothing here, in cases such as this we apply methods of Srinivas (1997) and Weischedel, Meteer, Schwartz, Ramshaw and Palmucci (1994): features of known words, including prefixes, suffixes, etc., are used for smoothing. Another reason for sparse data is that the word w_i and the supertag t_i both have been seen in the training data, but not together. This is the more frequent cause of sparse data (see Table 5). In situations like these, we will compare various smoothing techniques including using a high level organization of TAG.

The results of the n-best supertagger with no smoothing is shown in Table 10. It was trained on sections 02–21 of the Penn Treebank and tested on section 23. Accuracies and ambiguities are shown for different values of n-best up to $n = 10$. Note the precipitous drop in accuracy from the column representing the category "w_i, t_i seen together" to the column representing total accuracy. Most of the drop in accuracy is due to the category "w_i, t_i seen separately" (not included in these figures). This can be seen because this category accounts for the majority of the remaining instances, according to section 22, none of which the unsmoothed supertagger guesses correctly. The last row of Table 10, labeled by ∞, shows the accuracy and ambiguity if each word w is tagged with all supertags t such that $P(w|t) > 0$. (Unknown words are tagged with the set of the possible supertags t suggested by the unknown word smoothing procedure.) It can be considered as an upper bound of the supertagging accuracy in the sense that if the supertagger were always able to choose the correct supertag t out of the set of supertags $T(w) = \{t|P(w|t) > 0\}$ then the accuracy would be 94.37% as seen in Table 10.

[11] P stands for "probability." and $P(w_i, t_i)$ is the joint probability of word w_i and supertag t_i occurring in a training corpus.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

Table 10. *Results of n-best supertagger with no smoothing*

	Total		w_i, t_i seen together	
n-best	Pct Acc	Ambig (tags/wd)	Pct Acc	Ambig (tags/wd)
1	79.24	1.00	84.96	1.00
5	89.98	4.23	96.30	4.30
10	91.05	7.02	97.44	7.18
∞	94.37	45.59		

Table 11. *Results of n-best supertagger smoothing using part of speech information only ($\gamma = 0.0100$)*

	Total		w_i, t_i seen together		w_i, t_i separate	
n-best	Pct Acc	Ambig (tags/wd)	Pct Acc	Ambig (tags/wd)	Pct Acc	Ambig (tags/wd)
1	79.32	1.00	85.06	1.00	0.15	1.00
5	89.80	4.34	96.09	4.38	0.51	4.08
10	90.99	7.42	97.33	7.53	0.85	6.40
∞	94.88	202.87				

In stochastic models of TAG, it is common to smooth $P(w_i|t_i)$ using part of speech information. For example, when predicting the word w_i, Chiang (2000) immediately backs off from conditioning on the entire tree frame t_i to conditioning on the tree frame's preterminal (part of speech) p_i. For this reason, we investigate the effect on supertagging accuracy when smoothing $P(w_i|t_i)$ using part of speech. The following formula is used to smooth $P(w_i|t_i)$ using part of speech:

$$P(w_i|t_i) = \lambda(t_i)P_{MLE}(w_i|t_i) + (1 - \lambda(t_i))P_{MLE}(w_i|p_i) \tag{2}$$

$\lambda(t_i)$ has the following definition:

$$\lambda(t_i) = \frac{\gamma|t_i|}{1 + \gamma|t_i|} \tag{3}$$

where γ is a free parameter that determines the relative influence of $P_{MLE}(w_i|t_i)$ and the smoothing distribution $P_{MLE}(w_i|p_i)$. γ is set experimentally from the set $\{0.0010, 0.0100, 0.1000, 1.0000\}$ by evaluating corresponding supertagging models on the development corpus. The optimal value of γ is 0.0100. Subsequently, the supertagging model was run on the test corpus. An evaluation of the performance of this model is shown in Table 11.

Another baseline giving a quasi-upper bound is obtained by training the supertagging model according to the standard procedure with the following exception. Probabilities for $P(w_i|t_i)$ are estimated using not only the training corpus (Penn Treebank sections 02–21) but also the test corpus (section 23). (Probabilities for $P(t_i|t_{i-1}t_{i-2})$ are estimated as usual using only the training corpus.) Although this

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

Table 12. *Quasi-upper bound of smoothing using n-best supertagger*

	Total		w_i, t_i seen together		w_i, t_i separate	
n-best	Pct Acc	Ambig (tags/wd)	Pct Acc	Ambig (tags/wd)	Pct Acc	Ambig (tags/wd)
1	85.60	1.00	87.36	1.00	53.19	1.00
5	96.13	4.22	97.21	4.31	82.79	3.88
10	97.18	7.00	98.09	7.23	87.87	5.66
∞	99.53	42.40				

model's performance does not give a strict upper bound, it is a quasi-upper bound because it biases its probabilities towards events in the test corpus.

This model's results on the test corpus are shown in Table 12. Of particular note is the fact that the overall 1-best supertagging accuracy is only 85.60%. Also of interest is the 1-best supertagging accuracy for the category "w_i, t_i separate." It might be believed that the quasi-upper bound for the accuracy in this category would be an accuracy that is around the level of overall supertagging accuracy with no smoothing, that is, about 80%. The result here instead suggests the quasi-upper bound is around 53%.

6.2 Smoothing using feature vector based tree families

In this section, we shall explore smoothing using tree families induced from feature vectors. This addresses smoothing those kinds of supertags that are most in need for smoothing: "w_i, t_i separate." Tree families are induced from feature vectors as described in section 5.1. Here we define how we use tree families to smooth $P(w_i|t_i)$. This is followed by the description of smoothing experiments.

The following formula is used to smooth $P(w_i|t_i)$ using tree families:

$$P(w_i|t_i) = \begin{cases} \lambda(t_i)P_{MLE}(w_i|t_i) + (1-\lambda(t_i))P_{MLE}(w_i|f_i) & \textit{if } FAMILY(t_i) \\ \lambda(t_i)P_{MLE}(w_i|t_i) + (1-\lambda(t_i))P_{MLE}(w_i|p_i) & \textit{otherwise} \end{cases} \tag{4}$$

where $FAMILY(t_i)$ is a predicate which is true if and only if t_i belongs to some tree family, f_i is the tree family of supertag t_i, and p_i is t_i's part of speech. $\lambda(t_i)$ is defined as:

$$\lambda(t_i) = \frac{\gamma|t_i|}{1+\gamma|t_i|} \tag{5}$$

where γ is a free parameter whose value of 0.0100 has been determined experimentally in the same manner as has been used for the model that smooths using part of speech only.

The n-best supertagging results for this model are shown in Table 13. Overall, the 1-best supertagging results are not statistically significantly better than the baseline (no smoothing) results. Statistical significance is determined using the t-test. Turning to the 1-best results of the category "w_i, t_i separate," the results of smoothing using tree families are statistically significantly better than the baseline and also smoothing with part of speech. The relative results on this category continue to improve as the

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

Table 13. *Results of n-best supertagger smoothing using tree families* ($\gamma = 0.0100$)

	Total		w_i, t_i seen together		w_i, t_i separate	
n-best	Pct Acc	Ambig (tags/wd)	Pct Acc	Ambig (tags/wd)	Pct Acc	Ambig (tags/wd)
1	79.42	1.00	85.10	1.00	1.06	1.00
5	89.85	4.31	96.02	4.34	2.83	4.07
10	91.05	7.36	97.22	7.46	4.24	6.51
∞	94.93	195.42				

n in *n*-best is increased. This is probably due to the fact that smoothing using tree families induces a less flat probability distribution in a useful way than smoothing using part of speech.

On the other hand, this smoothing approach does not boost the overall accuracy very much. One might wonder why smoothing using tree families does not perform better. From a manual comparison of the errors that are produced by this smoothing method, we have uncovered various reasons why this is the case. One reason is that the tree families only include those supertags that represent predicate argument structure, which excludes the possibility of smoothing for example the NP argument supertag α_0 (Figure 14) with the predicate argument supertag α_1. This also excludes the possibility of smoothing two supertags that both do not represent predicate argument structure, such as α_0 and β_0 in the same figure – and these two supertags are more common than predicate argument supertags. Redefinition of tree families to include these supertags may yield better results. Another reason is that smoothing using tree families yields probability distributions that do not make enough useful distinctions. It gives about the same probability mass to each tree frame in the same tree family, although tree frames representing some transformations occur much more rarely than tree frames representing other transformations. Modeling the prior probability of a tree frame given a tree family may give better results.

6.3 Smoothing using distributional similarity

In section 5.2, it was seen that distributional similarity techniques seem to automatically induce tree families from extracted grammars. In this section, we investigate smoothing $P(w_i|t_i)$ using these techniques. Note that this may overcome certain problems of smoothing using feature vector based tree families. For example, it is possible to assign varying probabilities to every distinct pair of supertags in a set of distributionally similar supertags according to their degree of similarity.

The similarity measure of Jaccard's coefficient is used for computing smoothed values for $P(w_i|t_i)$.[12] Subsequently, the smoothed values $P_{SIM}(w_i|t_i)$ are used in n-best

[12] α-skew divergence has also been tried giving similar results. See Chen (2001) for details.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

supertagging. These values are determined as follows:

$$P_{SIM}(w_i|t_i) = \sum_{t'_i \in S(t_i)} \frac{SIM(t_i, t'_i)}{norm(t_i)} P_{MLE}(w_i|t'_i)$$

$$norm(t_i) = \sum_{t'_i \in S(t_i)} SIM(t_i, t'_i)$$

norm(t_i) is the normalization value. The parameters are τ (threshold value), ρ (number of most similar supertags to consider), and γ (setting the relative influence of P_{SIM} and P_{MLE}). In this case, we choose to vary τ and ρ, while setting γ to be constant. Of course, it might be the case that the best supertagging results for a particular set of parameter settings would be gained by setting γ individually for each set.[13]

We first discuss how γ is set. Then we discuss the supertagging results with Jaccard's coefficient.

To determine the value of γ, it is varied from the set $\{1.0000, 0.1000, 0.0100, 0.0010, 0.0001\}$. For each value of γ, supertagging is performed using smoothing with Jaccard's coefficient with $\tau = 0.08$ (threshold value) and $\rho = 40$ (number of most similar supertags). A local maximum 1-best supertagging accuracy is achieved when $\gamma = 0.0010$. It is perhaps interesting to note that in the training corpus, the median value of supertag occurrences was about 1000. For this median supertag t_{med}, the value $\lambda(t_{med})$ is

$$\begin{aligned} \lambda(t_{med}) &= \frac{\gamma\ |t_{med}|}{1.0 + \gamma\ |t_{med}|} \\ &= \frac{0.001(1000)}{1.0 + 0.001(1000)} \\ &= 0.5 \end{aligned}$$

That is to say, t_{med} is smoothed so that half of $P(w_i|t_{med})$ is contributed by P_{SIM} and the other half by P_{MLE}.

We now consider supertagging experiments where $P(w_i|t_i)$ is smoothed using distributional similarity with Jaccard's coefficient as the similarity measure. τ is varied between 0.04 and 0.16. ρ is varied between 25 and 55. The best results in terms of supertagging accuracy and coverage comes from setting τ to the minimum value 0.04.[14] The best results in terms of supertagging accuracy comes from setting ρ to the minimum value of 25.[15] The best results in terms of coverage comes from

[13] In Dagan, Lee and Pereira (1999), γ is always set to zero. Their situation is incomparable to the situation here, however, because in their experimental setup the baseline is backing off from $P(w_i|w_j)$ to the unigram $P(w_i)$ using Katz's back-off. Here, we do not back off to the unigram.

[14] It seems to show that setting τ to higher values means dismissing too many moderately similar supertags from being used in smoothing.

[15] Setting it higher means that there are more pairs of w and t such that the smoothed probability distribution $P(w|t) > 0$. In this situation, the effect seems to be that the supertagger gets more confused when supertagging words in the category "w_i, t_i seen together" but has an easier time supertagging words in the category "w_i, t_i separate."

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

Table 14. *Parameters* $\tau = 0.04$ *and* $\rho = 25$ *achieve the highest supertagging accuracies on n-best supertagging, smoothing with Jaccard's coefficient*

	Total		w_i, t_i seen together		w_i, t_i separate	
n-best	Pct Acc	Ambig (tags/wd)	Pct Acc	Ambig (tags/wd)	Pct Acc	Ambig (tags/wd)
1	80.65	1.00	85.39	1.00	21.34	1.00
5	91.80	4.37	95.91	4.39	45.48	4.45
10	93.04	7.48	96.93	7.55	51.27	7.49
∞	97.71	158.24				

setting ρ to the maximum value 55, obviously.[16] The parameters with the best overall supertagging results ($\tau = 0.04$ and $\rho = 25$) are shown in Table 14.

Smoothing $P(w_i|t_i)$ using distributional similarity with Jaccard's coefficient as the similarity measure has a promising effect on supertagging accuracy. Overall 1-best supertagging accuracy is boosted from 79.24% with no smoothing to 80.65%, a decrease in error rate of slightly under 7%. According to the t-test, this is a statistically significant result, $p < 0.05$. The corresponding decrease in error rate for 10-best supertagging is over 22%. This change is largely explained by the accuracy in the category "w_i, t_i seen separately," which is 21.34% for 1-best and 51.27% for 10-best supertagging. Both of the percentages are impressive, however, considering the no smoothing starting point of zero percent and also that they constitute 40-60% of the upper bound accuracies – recall Table 12. Nevertheless, the discrepancy in the percentages indicates that some fine tuning of the way distributional similarity is used for smoothing could yield probability distributions that shift more of the 51.27% improvement to the 1-best result.

An error analysis provides some direction as to how this fine tuning might take place. We performed a manual inspection of the errors in the test corpus tagged by a model smoothed by distributional similarity.[17] From this, we ascertain that this type of smoothing does not smooth low frequency supertags very well. This is not surprising because the paucity of samples of a low frequency supertag in the training corpus means it is difficult to assess how similar it is with other supertags. This source of error is not insignificant because frequencies of supertags tends to follow a Zipfian distribution; there is quite a long "tail" of low frequency supertags. High frequency supertags are another source of error. Although we find that these errors are greatly reduced by this kind of smoothing, there are so many high frequency supertags that still many of these errors remain. Our inspection provides clues as to why these errors are not corrected. Of particular interest is that one rather large source of error involves annotation error in the Penn Treebank. 20% of 30 randomly chosen errors involving high frequency supertags are of this type. Such a high percentage suggests that the remaining errors due to high frequency supertags may be difficult to correct using any approach.

[16] There is a trade-off in that the extra coverage comes at the expense of more ambiguity.
[17] See Chen (2001) for a more detailed perspective.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

We can imagine extensions to our smoothing approach that would ameliorate these sources or error. For example, an approach could be developed in which high frequency supertags are smoothed using distributional similarity over supertags as usual while low frequency supertags are smoothed using distributional similarity over features of supertags. The motivation here is that better similarity values might be obtained over features of supertags instead of whole supertags because more samples of the former than the latter can be obtained. Another avenue of improvement is to add features to the supertagging model which can distinguish between the causes of different kinds of capitalization of words.

6.4 Comparison of approaches

In this section, we have examined various methods for inducing a higher level organization of TAG and tested their applicability for smoothing. One approach introduces a semi-automatic method for decomposing tree frames into feature vectors. Among other attractions, the feature vector representation is a level of abstraction that can simplify the interface between extracted TAG grammars and other components, such as a semantic component or a component representing another grammatical formalism. Using the feature vector representation, tree frames are partitioned into tree families which are subsequently used for smoothing a supertagging model. The resulting model performs better than smoothing using part of speech, apparently because the probability distribution of the former is less flat in a useful way than that of the latter. It does not perform as well as distributional similarity smoothing, however. This is the case because the latter can induce a more flexible notion of tree family (for example, tree frames with different parts of speech can be found in the same tree family), it can associate a word with supertags inside a tree family from the existence of that word with supertags outside that tree family, it can associate different probabilities to different tree frames in the same tree family, and it can smooth supertags that occur outside any tree family. Weaknesses with smoothing using distributional similarity also exist; for example, it does not handle smoothing of low frequency supertags especially well.

7 Related work

In this section, we situate our work among related work. Most are related to extraction of TAGs from treebanks. These include Neumann (1998), Xia (1999) and Chiang (2000). We also examine the related topic of extracting a CCG from a treebank, work performed in (Hockenmaier and Steedman 2002a).

7.1 Stochastic lexicalized tree grammar

Neumann (1998) presents a method for extracting a lexicalized tree grammar from a treebank. The goals of his method are to explore the relationship between the extracted grammar and the treebank, and to use the extracted grammar for parsing. There are both similarities and differences with our work.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

It is closely related to our work in both the grammar that is produced and in the method that is used to produce the grammar. It is related by the former because Neumann (1998) produces a tree adjoining grammar, complete with initial and auxiliary trees. It is related by the latter because Neumann (1998) localizes head-dependency relationships into the same elementary tree roughly as we do, by using a head percolation table to determine the heads of interior nodes of a treebank parse, and then stipulating that an elementary tree have a trunk whose nodes have the same head.

There are also differences between our work and Neumann (1998). Unlike Neumann (1998), we extract complement and adjunct labels from the Penn Treebank's semantic tags to group into the same elementary tree only those nodes that are in a complement relationship with the head. Another difference is that we fragment conjunction structures so that each conjunct resides in a separate elementary tree. We also fragment selected punctuation into separate auxiliary trees. Our extra fragmentation results in more perspicuous grammars: the method of Neumann (1998) requires 12000 tree frames to cover sections 02, 03, and 04 only of the Penn Treebank. Furthermore, our method results in grammars whose trees better obey TAG's domain of locality since, for example, our method factors into separate auxiliary trees those PP's representing adjuncts. The method that we use in order to derive predicative auxiliary trees also differs from Neumann (1998). For us, a candidate tree is extracted as a predicative auxiliary tree if the root label matches the label of a node on the frontier of the candidate tree. Furthermore, the tree dominated by that node on the frontier must contain a trace for which the corresponding landing site has not yet been found. Neumann (1998) uses the former condition, but not the latter, which occasionally results in ill-formed predicative auxiliary trees.

7.2 Extraction of TAG from bracketed corpora

Xia (1999) presents a method for extracting a lexicalized TAG from a treebank. In Xia, Palmer and Joshi (2000), the extracted TAG is compared to a hand-crafted TAG (XTAG), and supertagging experiments are performed on the extracted TAG. In Xia *et al.* (2000), TAGs are extracted from treebanks of different languages. These TAGs are then compared. Note that this work has been done in parallel with our own work.

It is closely related to our work in that the end result is a lexicalized TAG and in the method that is used to produce the grammar. We both use a head percolation table in order to determine the heads of interior nodes of a treebank parse, and then define the trunks of elementary trees to be those nodes which have the same head. We both mark non-head nodes as either complement or adjunct, and factor adjuncts into separate modifier auxiliary trees. Following our suggestion, she also factors conjunction into separate auxiliary trees.

There are some differences between our work and Xia (1999). The method that we use to detect complements and adjuncts differs from the method that is used in Xia (1999). These methods were compared in section 4, with the method of Xia (1999) excelling in some areas such as perspicuity and better supertagging accuracy, and

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

Table 15. *Evaluation of Extracted Grammars from Xia (G_1 and G_2) and our paper (CA2-SOME-FULL and CA2-SOME-MERGED)*

Grammar	Tree Frames	Missed Coverage		% Supertag Accuracy
		W,T seen separately	W,T not seen	
G_1	6926	4.68	2.72	82.21
CA2-SOME-FULL	4723	4.60	2.64	80.03
G_2	2920	3.42	2.55	84.41
CA2-SOME-MERGED	3075	4.35	2.57	80.62

our method excelling in others such as ability to localize traces with their landing sites. Xia (1999) has done work in extracting multi-component TAGs using trace information, whereas that is beyond the scope of our work. Xia (1999) extracts different kinds of grammars that are based on different kinds of label sets, either the full Penn Treebank label set or a reduced label set. In the work that we present here, we compare grammars that are based on a greater variety of parameterizations. Also, we show how higher level organization can be induced from the extracted grammar and how it can play an important role in smoothing stochastic TAG-based models, issues that are not considered in the work reported in Xia (1999).

Xia (1999, 2000) evaluates extracted TAGs using many of the same tests that we use. Therefore, we may attempt to perform a comparison, which is shown in Table 15. Grammars G_1 and G_2 are extracted using LexTract, the algorithm of Xia. The former is composed of tree frames that are labeled with the FULL label set, whereas the latter's tree frames are labeled from a label set where some part of speech tag labels have been merged together. Therefore, G_1 and G_2 most closely correspond to our grammars CA2-SOME-FULL and CA2-SOME-MERGED, respectively.

Note other than the coverage of G_1 and CA2-SOME-FULL, the numbers do not seem to match up particularly well. This can be explained by differences between LexTract and our algorithm. For example, unlike XTAG or our extracted grammars, LexTract assumes that supertags for punctuation are the same as the parts of speech given to punctuation as specified in the Penn Treebank, i.e. the attachment point for a punctuation symbol in the parse tree is not specified by the supertag. Therefore, one would expect the supertagging accuracy to be higher for LexTract. Another possible difference is that the merged label set that we use may be significantly different than the one used by Xia.[18]

7.3 Stochastic lexicalized tree insertion grammar

Chiang (2000) presents a method for extracting a lexicalized TAG from a treebank. The goal of his approach is to perform stochastic modeling of TAG for parsing

[18] We have not given the specifics of how the label set is merged. Neither are details of how Xia merges the label set to be found in Xia (Xia 1999); Xia *et al.* 2000; Xia, Palmer and Joshi 2000).

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

disambiguation. There are many similarities, and a few differences with our work. Our work and his work were done independently of each other.

It is closely related to our work in what is produced and how it is produced. The end result of Chiang (2000) is a lexicalized tree insertion grammar (LTIG)[19] which is almost the same as TAG. Like we do, Chiang (2000) uses a head percolation table in order to determine the headedness of phrasal nodes, with paths in the treebank whose nodes have the same head being localized in the same lexicalized tree. Chiang (2000) also marks non-head nodes as either complement or adjunct, with adjuncts being factored into modifier auxiliary trees. He also factors conjuncts into separate auxiliary trees. Chiang (2000) uses a cutoff grammar with parameter $k = 2$ in his parsing algorithm, i.e. tree frames with frequency one in the training corpus are deleted from the grammar.

There are also differences between our work and Chiang (2000). Notably, Chiang (2000) follows exactly the head percolation rules and the strategy of detection of complements that is used by Collins (1997). As a result, there are some significant differences between the domain of locality of the grammar that is extracted by Chiang (2000) and the traditional kind of TAG domain of locality. For example, the grammar that is extracted by Chiang (2000) does not localize any trace with its landing site. Other examples include predicative nouns not being localized with their subjects and subjects in *by*-phrases not being localized with their passive verbs. Also, because Chiang (2000) extracts a TIG instead of a TAG, certain configurations of dependencies that are legal in TAG are not easily represented in the grammar of Chiang (2000). For example, given a particular node η, it is not possible to adjoin onto η both a tree from the left and a tree from the right. Consequently, in order to allow this, the grammar of Chiang (2000) must represent a node η by two nodes η_l and η_r, where η_l is designated to adjoin trees from the left only, and η_r is designated to adjoin trees from the right. Chiang (2000) uses only the condition that a root node label must match the label of some leaf node for a predicative auxiliary tree. Therefore, some predicative auxiliary trees can be extracted that should not be (like Neumann (1998), and unlike our procedure). For example, it is possible to factor out a predicative auxiliary tree whose foot node represents a sentential subject. Another case is illustrated in Figure 16; this approach would improperly extract the predicative auxiliary tree β_1 from parse tree τ_1. Chiang (2000) allows adjunction between two complement nodes, akin to D-Tree grammars (Rambow, Vijay-Shanker and Weir 1995). Chiang (2000) deletes all of the empty nodes from the Penn Treebank.

The grammar of Chiang (2000) is quite compact; only about 3600 tree frames are extracted from the training corpus of the Penn Treebank. The small size of this grammar largely derives from the facts that no empty elements are represented, there are differences in the head percolation table, and adjunction is allowed between two complement nodes. From another perspective, the possible drawback that the grammar of (Chiang 2000) has a smaller domain of locality than is traditionally

[19] LTIG is also known as lexicalized context-free grammar (LCFG), originally introduced by Schabes (1993).

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

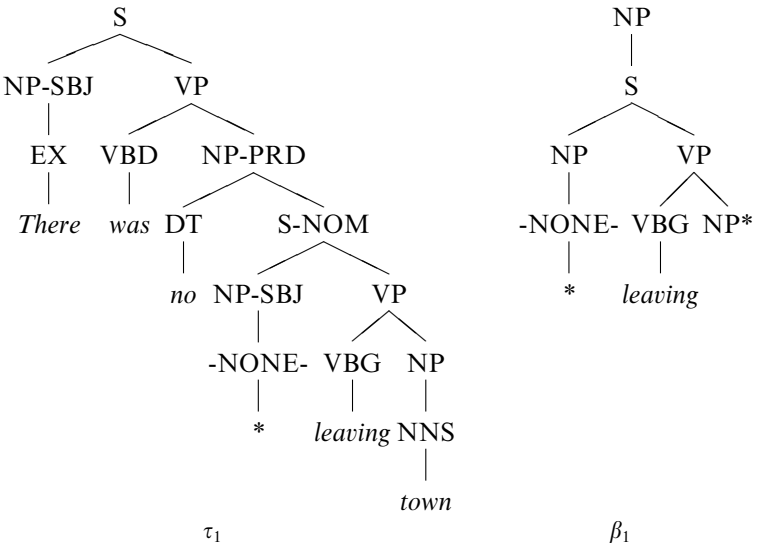


Fig. 16. Predicative auxiliary tree β_1 is improperly extracted from parse tree τ_1 given an extraction procedure that always extracts a predicative auxiliary tree whenever the root node matches a leaf node.

represented by TAG is counterbalanced by the advantage that the grammar is smaller. Because of its small size, it is unsurprising that the grammar of Chiang (2000) has only 0.20% missed coverage in terms of tree frames and 4.00% misses in terms of lexicalized trees.[20] In contrast, our extracted grammars get 0.19% to 0.44% missed coverage for the former and 6.92% to 7.96% misses for the latter. A caveat is that Chiang (2000) measures missed coverage on section 00 of the Penn Treebank, whereas our figures are from section 23.

Chiang (2000) presents the results of a stochastic TIG parser that is trained on an extracted TIG. The results as evaluated on section 23 of the Penn Treebank (86.7% labeled bracketed F-measure) are comparable but slightly lower than to state of the art stochastic parsers (88.1% to 90.1% F-measure) (Collins 1999; Charniak 2000). Collins (1999) notes that a problem with parsing using grammars such as TIG or TAG is that the sparse data problem is particularly severe. As we mentioned in section 6.1, the stochastic backoff model of Chiang (2000) is relatively simple: when predicting the lexical anchor w_i of a tree frame t_i, Chiang (2000) backs off from conditioning on t_i to conditioning on t_i's part of speech tag. Perhaps use of smoothing using distributional similarity in the context of stochastic TAG parsing would boost the accuracy of TIG or TAG parsers so that they achieve state of the art accuracy.

7.4 Combinatory categorial grammar

Combinatory categorial grammar (CCG) is a lexicalized grammatical formalism with many properties similar to that of TAG. Hockenmaier and Steedman (2002a) present

[20] Recall that extracted TAG grammars have a Zipfian distribution.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

an approach for extracting a CCG from the Penn Treebank. Some applications of the resulting CCG include CCG supertagging (Clark 2002) and full CCG parsing (Hockenmaier and Steedman 2002b). The goal of their work is to obtain a system that is able to perform semantic interpretation. Consequently, they feel free to modify the Penn Treebank annotation in the service of their goal, whereas our work is restricted to extracting TAGs such that unlabeled Penn Treebank bracketings may be assembled.

Combinatory categorial grammar (CCG) is a grammar formalism with the same generative capacity as TAG. In CCG, word order and valency information is represented as syntactic categories which are assigned to words. Hence, these categories contain the same kinds of information encoded in supertags. However, whereas phrase structure in supertags are represented rigidly and explicitly, it is absent in CCG categories. Instead, two categories may be composed by various operations including function application, or one category may be transformed by type raising. A sequence of function applications and type raising forms a derivation tree. A consequence of the CCG approach is that certain grammatical phenomena such as coordination may be handled more cleanly than in TAG.

Hockenmaier and Steedman (2002a) automatically extract a treebank with CCG annotations from the Penn Treebank. Because CCG is a lexicalized formalism, just as in the TAG case it requires a method to determine the head of a phrase, and also whether daughter nodes to the head are complements or adjuncts. Hockenmaier and Steedman do this using a head percolation table and heuristics based on Collins (1999). Since this is the same method that is used to extract a TIG in Chiang (2000), this extracted CCG grammar appears to have nearly the same domain of locality as this extracted TIG, with differences from our extracted TAG roughly the same as those domain of locality differences outlined in section 7.3. Hockenmaier and Steedman treat two cases of coordination: simple coordination (e.g. NP coordination) and argument cluster coordination. The former is handled by our extracted grammar just as well. The latter is handled less well because it is a kind of gapping, where the second conjunct lacks both the subject and the verb, as in the sentence "The company gave workers extra benefits and affiliates more compensation." Although there is no concept in CCG of localizing traces with landing sites, this information is represented in the categories of head words of landing sites. In order to ensure these categories are assigned properly, one preprocessing step in CCG extraction decorates internal nodes of Penn Treebank bracketings with slash categories similar to those used in GPSG (Gadzar *et al.* 1985), whose analogue in our TAG extraction procedure would be the use of trace sets.

An interesting result is that only 1301 CCG categories are needed to cover the training corpus. One caveat to this result is that the CCG grammar is extracted from a version of the Penn Treebank where certain annotation errors are cleaned up. Furthermore, note that the portions of the CCG grammar corresponding to parts of speech seem to lack the detail of the parts of speech found in the Penn Treebank. For example, there appears to be no distinction made between proper and common nouns. If so, it would be more appropriate to compare the count of 1301 CCG categories with the count of tree frames in our MERGED grammars

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

(3075-5953 tree frames). Note also that unlike a goal of our work, their goal does not include developing a CCG grammar that can reconstruct the original unlabeled Penn Treebank bracketings. They take advantage of this by first normalizing the Penn Treebank bracketings in various respects, such as coordination, before CCG extraction occurs. Aside from these reasons, the compactness of the grammar may in large part be due to the CCG formalism itself. In particular, a large set of transformationally related TAG tree frames for a given predicate typically boil down into only several CCG categories (Doran and Srinivas 1994).

Despite its compactness, Hockenmaier and Steedman state that the extracted CCG suffers from sparse data, in particular with respect to pairs of word and CCG category that are seen in a held out corpus, but not in the training corpus. In order to reduce this problem, Hockenmaier and Steedman introduce type-changing rules, which map a large set of categories corresponding to modifiers and extracted elements into a smaller set of categories. They show that this reduces the misses in coverage on a held out CCG corpus of categories from 3.0% to 2.2%. This is a relative reduction in misses of 26.6%. In contrast, by comparing the ∞ rows in Table 10 and Table 14, we show that by using our technique of smoothing using distributional similarity, we reduce absolute misses from 5.63% to 2.29%, a relative reduction in misses of 59.3%.[21] Aside from increasing coverage, their idea may also be conceivably integrated into a smoothing model for a statistical CCG full parser or supertagger, in analogue to the work presented in this article for smoothing supertagging models for TAG. Conversely, our ideas for smoothing TAG should also work for smoothing CCG.

Clark (2002) performs experiments with CCG supertagging, i.e. training a statistical model to tag raw input sentences with CCG categories as a preprocessing step for a CCG full parser. The set of categories used in Clark (2002) does not include any type-raised categories, i.e. it is the set of categories obtained after application of type-changing rules. Clark (2002) performs a variety of supertagging experiments using different feature sets and different modeling techniques. With an optimized feature set including pairs of words, categories, and part of speech tags, over a 5-word window size a maximum entropy model for CCG supertagging performs at 90.5% accuracy. This compares favorably with the supertagging results of grammars used in this paper (78.55% to 80.62% accuracy). Note however that unlike Clark (2002) the main point of this paper is not to obtain the highest possible supertagging accuracies, but to see what happens to supertagging accuracy when different extracted TAGs are used, and to evaluate various approaches for smoothing probability distributions of stochastic models of extracted TAGs.

8 Conclusions

Recent work in statistical natural language processing has shown the importance of lexicalized models in creating systems with superior performance. Although TAG

[21] Chen (2001) contains more exhaustive experiments for extracted TAGs that focus on reducing missed coverage using techniques employed in this paper.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

is an example of a lexicalized formalism upon which much theoretical linguistic research has been performed, its use in stochastic modeling has been relatively modest. There are various reasons for this, for example the difficulty of manually creating a TAG-annotated corpus.

We present a method for automatically inducing a TAG from the Penn Treebank. Our method is appealing for various reasons. First, treebanks such as the Penn Treebank are widely available. Second, the amount of labor that is used to induce a TAG using our procedure is quite modest. Third, our procedure extracts linguistically motivated TAGs; for example, the resulting tree frames of the TAG obey what is customarily considered to be TAG's domain of locality, and they factor recursion, whether it be of a kind that is predicative, modifier, or conjunction, as appropriate. Fourth, as the specific definition of what a TAG should be is not rigid, our procedure is parameterizable to extract different kinds of TAGs.

We also present an evaluation of different extracted TAGs based on different parameterizations of the tree extraction procedure. For statistical modeling, we see that extracted TAGs have quite a sparse data problem because of the large number of tree frames they contain. Furthermore, we see that there is a strong inverse correlation between the size of the grammar in terms of lexicalized trees and its supertagging accuracy. We also note there is a tension between the TAG principle of localizing dependencies, which leads to large grammars, and the desire to maximize supertagging accuracy. This tension seems to be assuaged somewhat by the kinds of dependencies that are localized. For example, increasing the number of empty elements in the grammar hurts supertagging accuracy more than increasing the number of "complements" in the grammar, perhaps because the former creates less useful distinctions in the stochastic model for the purpose of supertag disambiguation.

We also present several labor-inexpensive methods for inducing higher-level organization of TAGs. The first is a semi-automatic organization that turns tree frames into feature vectors. The second is a more automatic organization that groups tree frames into tree families using distributional similarity. We note the potential usefulness of such an organization in many areas, including aiding smoothing, easing grammar development, interfacing the extracted grammar to other components.

Insofar as sparse data seems to be an egregious problem with extracted TAGs, we investigate using the higher-level organization of TAGs for the purpose of smoothing to increase supertagging accuracy. We show that they are indeed useful for significantly improving supertagging accuracy. For the purpose of stochastic modeling, this provides evidence that while the use of TAG does incur the disadvantage of sparse data in comparison with currently more prevalent approaches such as Markov grammars (as noted by Collins (1999), for example), it also provides the opportunity for exploring the use of novel smoothing techniques.

A Appendix

This appendix gives a detailed overview of the head percolation table that is used in the tree extraction procedure. First, we describe the purpose of the table in

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

relation to the tree extraction procedure. Second, we provide the unabridged table and describe how it is used. Third, we describe how we tailored the table for its use in automatically extracting TAGs.

The tree extraction procedure depends upon a method for determining the head child η_h of a phrase when given a parent node η^p and its children $\eta_1, \ldots, \eta_m$. In order to accomplish this task, a slight variant of what is called a *head percolation table* is used. The head percolation table was introduced by Magerman (1994). It is a manually constructed, heuristic method for determining the value of η_h. Collins (1999) adopts it for use in the Penn Treebank II. Modifications of this table in particular are made for use here.

The head percolation table is shown in Table 16. The first column is the label of the parent node η^p, one of whose children $\eta_1, \ldots, \eta_m$ is to be selected as the head child. The second column indicates the direction from which the children should be processed. It either indicates that the children should be processed starting from the leftmost child η_1 or from the rightmost child η_m. The third column gives priorities to various possible child labels. For example, for the parent node that is labeled SBAR, S is the child label with first priority, SQ is the child label with second priority, etc. An innovation from previous incarnations of the head percolation table is that sets of children in parentheses indicate children at the same priority level. For example, for the parent node that is labeled NP, the labels NN, NNS, NNP, and NNPS are all at the first priority.

The head percolation table determines the head child as follows. The children $\eta_1, \ldots, \eta_m$ are iteratively scanned according to the direction in the table. To start, the first iteration scans for a child with label(s) of highest priority. In general, the kth iteration scans for a child with label(s) of kth priority. The process stops as soon as a match of a label of some η_i with a label that is currently sought is found. That η_i becomes the head child η_h.

Instead of adopting the head percolation table of Collins directly, certain changes have been made. These changes are justified here.

The head percolation table in Collins (1999) follows that in Magerman (1994) in that there is a total ordering on the priority of child labels. An alteration to that scheme that is used here is that a set of child labels can have the same priority. This is used in the head percolation table entry for parent label NP. It was implemented because there are many examples of base NP[22] bracketings for which there is no clear priority among child labels. For example, in τ_0 and τ_1 in Figure 17 intuitively the rightmost child in both cases should be selected as the head child. This is not possible to accomplish using the scheme of Collins because, given NN and NNS, either NN is always selected as the head child, or NNS always is. In contrast, the alteration allows NN to be selected in some cases and NNS to be selected in others, given that NN and NNS are set at the same priority level.

Other changes to the head percolation table have to deal with editing the entries in the table to have the TAG extraction procedure produce elementary trees that localize

[22] Recall that *base NP* is a bracketing that is rooted at NP such that the root does not properly dominate another NP.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

Table 16. *Head percolation table*

Label of η^p	Direction	Priority list of labels of η
ADJP	right	NNS QP NN $ ADVP JJ VBN VBG ADJP JJR NP JJS DT FW RBR RBS SBAR RB
ADVP	right	RB RBR RBS FW ADVP TO CD JJR JJ IN NP JJS NN
CONJP	right	CC RB IN
FRAG	right	**
INTJ	left	**
LST	right	LS :
NAC	left	NN NNS NNP NNPS NP NAC EX $ CD QP PRP VBG JJ JJS JJR ADJP FW
NP	right	(NN NNS NNP NNPS) (NP NX) NAC EX $ CD QP PRP VBG JJ JJS JJR ADJP FW SYM
NX	right	(NN NNS NNP NNPS) (NP NX) NAC EX $ CD QP PRP VBG JJ JJS JJR ADJP FW SYM
PP	right	IN TO VBG VBN RP FW
PRN	left	**
PRT	right	RP
QP	left	$ IN NNS NN JJ RB DT CD NCD QP JJR JJS
RRC	right	VP NP ADVP ADJP PP
S	right	VP NP-PRD ADJP-PRD ADVP-PRD PP-PRD SBAR ADJP UCP NP
SBAR	right	S SQ SINV SBAR FRAG X
SBARQ	right	SQ S SINV SBARQ FRAG X
SINV	right	VP NP-PRD ADJP-PRD ADVP-PRD PP-PRD S VBZ VBD VBP VB SINV ADJP NP
SQ	right	VP NP-PRD ADJP-PRD ADVP-PRD PP-PRD VBZ VBD VBP VB MD SQ
UCP	right	**
VP	right	VP NP-PRD ADJP-PRD ADVP-PRD PP-PRD VBD VBN VBZ VB MD TO VBG VBP ADJP NP
WHADJP	left	CC WRB JJ ADJP
WHADVP	right	CC WRB
WHNP	left	WDT WP WP$ WHADJP WHPP WHNP
WHPP	right	IN TO FW
X	left	**

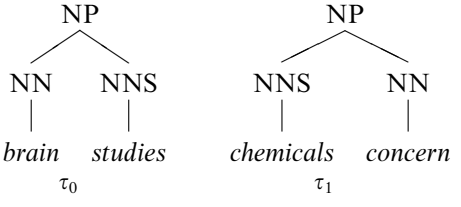


Fig. 17. Examples of base NP bracketings that suggest that NN and NNS should be set at the same priority.

semantic dependencies. One change has been that NPs, ADJPs, and ADVPs which represent predication are given higher priority in the table over verbs. Consequently, the semantic relationship between the NP, ADJP, or ADVP and the NP subject is localized in the same elementary tree, as shown in Figure 18. Another significant change has been that labels signifying *wh*-words such as WHNP and WHADVP are given lower priority than clausal labels such as S and SBAR for the entry in the

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

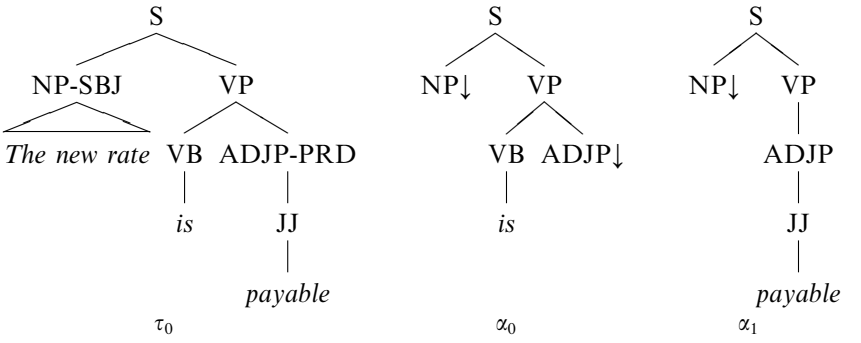


Fig. 18. τ_0 is a Penn Treebank bracketing. α_0 is the initial tree that would be extracted if VB has higher priority than ADJP-PRD. α_1 is the initial tree that would be extracted if ADJP-PRD has higher priority than VB. Note that the dependency between the NP subject and ADJP is localized in α_1, but not in α_0.

head percolation table for clausal labels. This allows an extraposed complement to be localized in the same elementary tree as its predicate.

References

Bangalore, S., Chen, J. and Rambow, O. (2001) Impact of quality and quantity of corpora on stochastic generation. *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, Pittsburgh, PA.

Bangalore, S. and Joshi, A. K. (1999) Supertagging: An approach to almost parsing. *Computational Linguistics*, **25**(2).

Bies, A., Ferguson, M., Katz, K. and MacIntyre, R. (1995) Bracketing guidelines for treebank ii style penn treebank project. Technical report, University of Pennsylvania, Philadelphia, PA.

Briscoe, T. and Carroll, J. (1997) Automatic extraction of subcategorization from corpora. *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP 97)*, Washington, DC.

Buchholz, S. (1998) Distinguishing complements from adjuncts using memory-based learning. *Proceedings of the ESSLLI-98 Workshop on Automated Acquisition of Syntax and Parsing*, Saarbrücken, Germany.

Charniak, E. 2000. A maximum-entropy-inspired parser. *Proceedings of First Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, Seattle, WA.

Chen, J. (2001) *Towards Efficient Statistical Parsing Using Lexicalized Grammatical Information*. PhD thesis, University of Delaware.

Chen, J., Bangalore, S., Collins, M. and Rambow, O. (2002a) Reranking an n-gram supertagger. *Proceedings of the Sixth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6)*, Venice, Italy.

Chen, J., Bangalore, S., Rambow, O. and Walker, M. (2002b) Towards automatic generation of natural language generation systems. *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002)*, Taipei, Taiwan.

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

Chen, J. and Rambow, O. 2003. Use of deep linguistic features for the recognition and labeling of semantic arguments. *Proceeedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, Sapporo, Japan.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

Chiang, D. (2000) Statistical parsing with an automatically-extracted tree adjoining grammar. *Proceedings of the the 38th Annual Meeting of the Association for Computational Linguistics*, pp. 456–463. Hong Kong.

Chiang, D. and Bikel, D. M. (2002) Recovering latent information in treebanks. *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002)*, Taipei, Taiwan.

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

Collins, M. (1997) Three generative lexicalized models for statistical parsing. *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics.*

Collins, M. (1999) *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania.

Dagan, I., Lee, L. and Pereira, F. C. N. (1999) Similarity-based models of word cooccurrence probabilities. *Machine Learning*, **34**, 43–69.

Doran, C. and Srinivas, B. (1994) Bootstrapping a wide-coverage ccg from fb-ltag. Technical Report TALANA-RT-94-01, TALANA, Université Paris 7, Paris, France.

Frank, R. (2002) *Phrase Structure Composition and Syntactic Dependencies*. MIT Press.

Gadzar, G., Klein, E., Pullum, G. K. and Sag, I. A. (1985) *Generalized Phrase Structure Grammar*. Harvard University Press.

Gildea, D. and Hockenmaier, J. 2003. Identifying semantic roles using combinatory categorial grammar. *Proceeedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, Sapporo, Japan.

Haegeman, L. (1994) *Introduction to Government and Binding Theory*. Blackwells.

Hockenmaier, J. and Steedman, M. (2002a) Acquiring compact lexicalized grammars from a cleaner treebank. *Proceedings of the Third International Conference on Language Resources and Evaluation*, Las Palmas.

Hockenmaier, J. and Steedman, M. (2002b) Generative models for statistical parsing with combinatory categorial grammar. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, PA.

Hwa, R. (1998) An empirical evaluation of probabilistic lexicalized tree insertion grammars. *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics*, pp. 557–563. Montreal, Canada.

Hwa, R. (1999) Supervised grammar induction using training data with limited constituent information. *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pp. 73–79. College Park, MD.

Katz, S. M. (1987) Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Trans. Acoustics, Speech and Signal Process.* **35**: 400–401.

Kingsbury, P., Palmer, M. and Marcus, M. 2002. Adding semantic annotation to the penn treebank. *Proceedings of the Human Language Technology Conference*, San Diego, CA.

Lee, L. (1999) Measures of distributional similarity. *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pp. 25–32.

Magerman, D. M. (1994) *Natural Language Parsing as Statistical Pattern Recognition*. PhD thesis, Stanford University.

Manning, C. D. and Schütze, H. (1999) *Foundations of Statistical Natural Language Processing*. MIT Press.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K. and Schasberger, B. (1994) The penn treebank: Annotating predicate argument structure. *Proceedings of the 1994 Human Language Technology Workshop*, pp. 110–115.

Merlo, P. and Leybold, M. (2001) Automatic distinction of arguments and modifiers: the case of prepositional phrases. *Proceedings of the Fifth Workshop on Computational Natural Language Learning (CoNLL-01)*, Toulouse, France.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press

Neumann, G. (1998) Automatic extraction of stochastic lexicalized tree grammars from treebanks. In *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks*, pp. 120–123.

Radford, A. (1988) *Transformational grammar: a first course*. Cambridge University Press.

Rambow, O., Vijay-Shanker, K. and Weir, D. (1995) D-tree grammars. *Proceedings of the 33rd Annual Meeting of the Assocation for Computational Linguistics*, Cambridge, MA.

Resnik, P. (1992) Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. *Proceedings of the 15th International Conference on Computational Linguistics (COLING 92)*, pp. 418–424. Copenhagen, Denmark.

Sarkar, A. (2000) Practical experiments in parsing using tree adjoining grammars. *Proceedings of the Fifth International Workshop on Tree Adjoining Grammars and Related Frameworks*, Paris, France.

Sarkar, A. (2001) Applying co-training methods to statistical parsing. *Proceedings of Second Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, Pittsburgh, PA.

Sarkar, A. and Joshi, A. (1996) Coordination in tree adjoining grammars: Formalization and implementation. *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pp. 610–615. Copenhagen, Denmark.

Schabes, Y. and Waters, R. (1996) Stochastic lexcalized tree-insertion grammar. In: H. Bunt and M. Tomita, editors, *Recent Advances in Parsing Technology*, pp. 281–294. Kluwer.

Schabes, Y. (1992) Stochastic lexicalized tree-adjoining grammars. *Proceedings of the 15th International Conference on Computational Linguistics (COLING 92)*, pp. 426–432. Copenhagen, Denmark.

Schabes, Y. (1993) Lexicalized context-free grammars. Technical Report 93-01, Mitsubishi Electric Research Laboratories, Cambridge, MA.

Schabes, Y., Abeillé, A. and Joshi, A. K. (1988) Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammars. *Proceedings of the 12th International Conference on Computational Linguistics*, Budapest, Hungary.

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

Srinivas, B. (1997) Performance evaluation of supertagging for partial parsing. *Proceedings of the Fifth International Workshop on Parsing Technologies*, pp. 187–198. Cambridge, MA.

Weischedel, R., Meteer, M., Schwartz, R., Ramshaw, L. and Palmucci, J. (1994) Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics*, **19**(2): 359–382.

Xia, F. (1999) Extracting tree adjoining grammars from bracketed corpora. *Fifth Natural Language Processing Pacific Rim Symposium (NLPRS-99)*, Beijing, China.

Xia, F., Han, C. H., Palmer, M¡. and Joshi, A. (2000) Comparing lexicalized treebank grammars extracted from chinese, korean, and english corpora. *Proceedings of the Second Chinese Language Processing Workshop (CLP-2000)*, Hong Kong, China.

Xia, F., Palmer, M. and Joshi, A. (2000) A uniform method of grammar extraction and its applications. *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000)*, Hong Kong, China.

Xia, F., Palmer, M., Vijay-Shanker, K. and Rosenzweig, J. 1998. Consistent grammar development using partial-tree descriptions for lexicalized tree adjoining grammars. *Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks*, pp. 180–183.

XTAG-Group, The (2001) A Lexicalized Tree Adjoining Grammar for English. Technical report, University of Pennsylvania. Updated version available at http://www.cis.upenn.edu/~xtag.

https://doi.org/10.1017/S1351324905003943 Published online by Cambridge University Press