Extracting Tree Adjoining Grammars from Bracketed Corpora

Fei Xia[*]
Department of Computer and Information Science
University of Pennsylvania
3401 Walnut Street, Suite 400A
Philadelphia PA 19104, USA
fxia@linc.cis.upenn.edu

Abstract

In this paper, we report our work on extracting lexicalized tree adjoining grammars (LTAGs) from partially bracketed corpora. The algorithm first fully brackets the corpora, then extracts elementary trees (*etrees*), and finally filters out invalid *etrees* using linguistic knowledge. We show that the set of extracted *etrees* may not be complete enough to cover the whole language, but this will not have a big impact on parsing.

1 Introduction

Lexicalized Tree Adjoining Grammar (LTAG) is a tree-rewriting formalism. It is more expressive than a context-free grammar (CFG),[1] and therefore a better formalism for representing various phenomena in natural languages. In the last decade, it has been applied to various NLP tasks such as parsing (Srinivas, 1997), machine translation (Palmer et al., 1998), information retrieval (Chandrasekar and Srinivas, 1997), generation (Stone and Doran, 1997; McCoy et al., 1992), and summarization applications (Baldwin et al., 1997). A wide-coverage LTAG for a particular natural language often contains thousands of trees and takes years to build.

There has been work on extracting CFGs (Shirai et al., 1995; Charniak, 1996; Krotov and others, 1998) and lexicalized tree grammars (Neumann, 1998; Srinivas, 1997) from bracketed corpora. In this paper, we propose a new method for learning LTAGs from such corpora.[2]

2 Penn Treebank and LTAG

2.1 Penn Treebank

In this paper, we use the English Penn Treebank as our bracketed corpus, which includes about 1 million words from Wall Street Journal annotated in Treebank II style (Marcus et al., 1994). Its tagset has 85 syntactic labels – 48 POS tags, 27 syntactic category tags such as *NP* for noun phrase, 10 tags for empty categories such as **T** for trace – and 30 function tags(e.g. *SBJ* for the subject in surface structure). Each bracket is labeled with one syntactic category, up to four function tags and reference indices if necessary, cf. (Santorini, 1990) and (Bies et al., 1995). Figure 1 is a simple example. We will use this example throughout the paper. The meaning of the tags used in this example are listed in Table 1.

```
((S (S   (NP-SBJ (PRP I) )
         (VP   (VBP 've)
               (VP (VBN got)
                   (NP     (JJ limited)
                           (NN production) ))))
    (, ,)
    (CC and)
    (S (NP-SBJ (PRP I) )
       (VP (MD can)        (RB not)
           (VP (VB give)
               (NP (PRP it) )
               (PP-DTV (IN to)
                       (NP (PRP you) ))
               (PP-TMP (IN (by)
                       (ADVP (RB then) )))))
    (. .) ))
```

Figure 1: A simple example

2.2 Basics of LTAGs

LTAGs are based on the Tree Adjoining Grammar (TAG) formalism developed by Joshi, Levy, and Takahashi (Joshi et al., 1975; Joshi and Schabes, 1997). The primitive elements of the LTAG formalism are elementary trees. Each elementary tree is associated with at least one lexical item (called *the anchor* of the tree) on its frontier, and the tree provides extended locality over which the syntactic and semantic constraints can be specified.

[*] The author wishes to thank Chung-hye Han, Aravind Joshi, Martha Palmer, Carlos Prolo, Anoop Sarkar, and three anonymous reviewers for many helpful comments.

[1]LTAG is more expressive than CFG formalism both in weak and strong generative capacity, e.g. it can handle cross dependency elegantly.

[2]A related work is (Srinivas, 1997), but its goal is not to learn a new LTAG but to extract the useful information, such as dependency and frequency of trees, for an existing LTAG.

ADVP	adverb phrase
NP	noun phrase
PP	preposition phrase
VP	verb phrase
S	sentence
CC	coordinating conjunction
IN	preposition
JJ	adjectives
MD	modal
NN	noun, singular or mass
NNS	noun, plural
PRP	pronoun
RB	adverb
VB	verb, base form
VBN	verb, past participle
VBP	verb, non-3rd singular present
,	comma
.	period
”	quotation mark
DTV	dative
SBJ	subject
TMP	temporal

Table 1: Treebank tags which appear in this paper

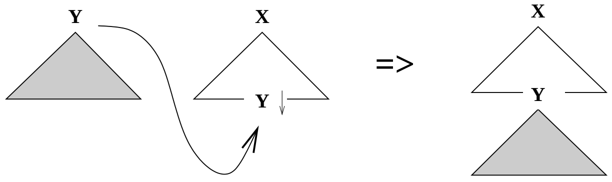

Figure 2: The substitution operation

There are two types of elementary trees: initial trees and auxiliary trees. Each auxiliary tree has a unique leaf node, called the *foot* node, which has the same label as the root. In both types of trees, leaf nodes other than anchors and foot nodes are substitution nodes.

Elementary trees are combined by two operations: substitution and adjunction. In the substitution operation (Figure 2), a substitution node in an elementary tree is replaced by another elementary tree whose root has the same label as the substitution node. In an adjunction operation (Figure 3), an auxiliary tree is inserted into an initial tree. The root

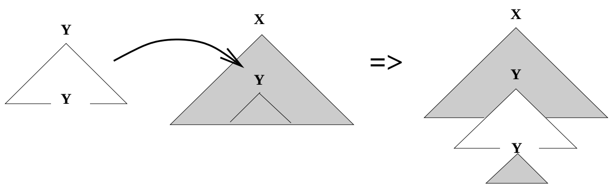

Figure 3: The adjunction operation

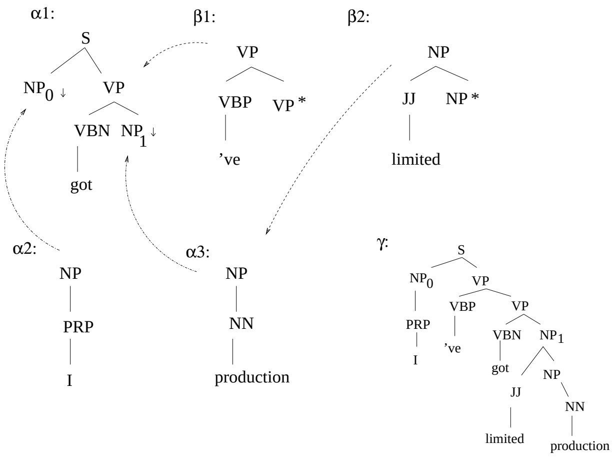

Figure 4: Elementary trees and the derived tree for *I 've got limited production*

and the foot nodes of the auxiliary tree must match the node label at which the auxiliary tree adjoins. The resulting structure of the combined elementary trees is called a *derived tree.*

In Figure 4, $\alpha 1$, $\alpha 2$ and $\alpha 3$ are initial trees anchored by *got*, *I* and *production* respectively. $\beta 1$ and $\beta 2$ are auxiliary trees for verb *'ve* and adjective *limited.* Foot and substitution nodes are marked by $*$, and $\downarrow$ respectively. To parse the sentence *I've got limited production*, $\alpha 2$ substitutes at NP_0 in $\alpha 1$, $\beta 2$ adjoins to the NP node in $\alpha 3$, then the whole tree substitutes into the NP_1 node in $\alpha 1$, meanwhile, $\beta 1$ adjoins to the VP node in $\alpha 1$, thus, forming the derived tree γ. The process is denoted by the dotted lines between trees.

For the sake of clarity, from now on, we will call the bracketed structures in the Penn Treebank *ttrees*, and the elementary trees in LTAGs *etrees.*

2.3 A special form of LTAGs

The LTAG formalism does not impose constraints on the shapes of the *elementary trees* as long as they satisfy the requirements mentioned in Section 2.2. As a result, given a corpus C, there are numerous LTAGs each of which covers C. An LTAG G is said to *cover* a corpus C if each *ttree* in C can be generated by combining the *etrees* in G with substitution and adjunction operations.

To ensure the grammar extracted is compact and linguistically sound, we require that the *etrees* in the grammar have one of the following forms, as shown in Figure 5:[3]

- spine-etree: the anchor of the *etree* is X, which

[3]This description is similar to the rules in X-bar theory: a spine-etree can be seen as a result of combining the rules $X^m \rightarrow Y^k X^{m-1}$, ... $X^1 \rightarrow X Z^p$ in one tree; a mod-etree corresponds to the rule $W^q \rightarrow W^q X^m$ and X^m is further expanded to include its arguments.

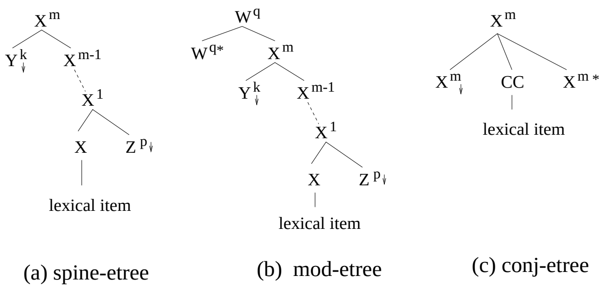

Figure 5: The three forms that extracted *etrees* should belong to

projects to X^1, X^2, all the way to the root X^m. Note: each X^i and its parent X^{i+1} have different labels, e.g. VB and VP. At each level, the relation between X^i and its sisters is a predicate-argument relation. A spine-etree is either an initial tree or a predicative auxiliary tree.

- mod-etree: the root of the *etree* has two children, one is a foot node with the same label W^q, the other node X^m is the root of a spine-etree whose head X is the anchor of the whole mod-etree. X^m is a modifier of W^q. A mod-etree is always an auxiliary tree.
- conj-etree: The anchor of the *etree* is a conjunction. It conjoins two components X^m (one of them is a foot node, the other is a substitution node.) and the root is also labeled as X^m.[4] A conj-etree is always an auxiliary tree.

In Figure 4, α1-α3 are spine-etrees, β1 and β2 are mod-etrees. From now on, *etrees* in the paper refers to elementary trees with one of the three forms just mentioned.

3 Extracting LTAG from the Treebank

The algorithm for grammar extraction can be divided into three steps: fully bracket the *ttrees*, extract *etrees* and filter out invalid *etrees*.

3.1 Fully bracketing the *ttrees*

The *ttrees* from Penn Treebank are partially bracketed: the head/argument/modifier distinction is not marked explicitly, and arguments and modifiers for the same head are both sisters of the head. In the LTAG, arguments appear in spine-etree and modifiers in mod-etree, and each mod-etree takes exactly one modifier. To account for this difference, we need to first fully bracket the *ttrees*. To do that, we start from the root R of the *ttree*:

Step A: Choose the *head-child*[5] *hc* according to a head percolation table. The table lists the possible head-child for each syntactic category (*b-tag* for short), e.g. the VP's head-child can be VBN, VB, VP, etc. A similar method has been used in (Magerman, 1995; Collins, 1997) among others.

Step B: Decide what the relationship between *hc* and its sisters is by comparing the *b-tags* of *hc* and the root R, and check whether one of R's children is a conjunction.

Step C: based on the result of step B, go to one of the following:

(1) predicate-argument relation:
For each sister x of *hc*, fully bracket the subtree rooted at x, then decide whether x is an argument or a modifier of *hc* according to the argument table and tagset table.[6]
If in addition to sisters which are arguments, *hc* also has sisters which are its modifiers, insert an intermediate node R^* with R's b-tag as the new root of *hc* and its arguments, then insert $m-1$ intermediate nodes with R's b-tag between R^* and R, as in (2).

(2) modification relation:
Suppose *hc* has m sisters (all of them are modifiers), insert $m-1$ nodes with R's b-tag between *hc* nd R so that each level has exactly one modifier.

(3) coordination relation:
Use conjunctions to partition the non-conjunction children into m groups. If each group has more than one component, insert a new node with R's b-tag as the new root of the group. Fully bracket each group. If m is larger than 2, insert $m-2$ intermediate nodes with R's b-tag between R and its children so that each level has exactly 2 groups plus one conjunction.

The output[7] is shown in Figure 6. The nodes inserted by the algorithm are in bold font.

[4]Coordination of different syntactic categories can be handled similarly. An alternative to handle coordination is to use mod-etrees: treat one conjoined X^m as the foot node, expand the other X^m and mark the conjunction as a substitution node. The algorithm in Section 3 can be easily modified to accommodate this approach.

[5]The *head-child* differs from the *head* in X-bar theory in that the head of X^m is X, while its head-child can be X, X^1, ..., X^m.

[6]The argument table lists the types of the arguments that each head can take, e.g. VB can take NP, PP and S as arguments. Tagset table specifies whether certain function tags mark modifiers or arguments, e.g. phrases with -TMP are modifiers, those with -DTV are arguments. x is *hc*'s argument if x's *b-tag* is among the possible types of *hc*'s arguments and x's function tag does not indicate it is a modifier.

[7]The bracketing process may eliminate the potential ambiguity which exists in the original *trees*, but in most cases, it will not affect the extracted *etrees*. In this step, we also remove the punctuations.

```
((S (S    (NP-SBJ (PRP I))
          (VP  (VBP 've)
               (VP (VBN got)
                   (NP   (JJ limited)
                         (NP   (NN production))))))
    (CC and)
    (S (NP-SBJ (PRP I))
       (VP (MD can)
           (VP    (RB not)
                  (VP  (VP   (VB give)
                             (NP (PRP it))
                             (PP-DTV (IN to)
                                     (NP (PRP you) )))
                       (PP-TMP    (PP   (IN (by))
                                        (ADVP (RB then)))))))))
```

Figure 6: The fully bracketed structure for the example in Figure 1

3.2 Building *etrees*

Once the *ttrees* are fully bracketed, the extraction algorithm is straightforward.[8] Let's first define a few terms. A *head-path* starting from a node R in a *ttree* is the path from R to a leaf node where each node except R is the head-child of its parent in the *ttree*. A node on the head-path is called a *link node* if its b-tag is the same as the b-tag of its head-child.[9] The extraction process involves copying nodes x from *ttree* to *etree*. Let's denote the copy of x as x^*.[10] To build a set of *etrees* for a *ttree*, we start from the root R of the *ttree* T:

Step A: Choose the head-child hc and decide the relation between hc and its sisters, as mentioned in the previous section.

Step B: Based on the result of step A, go to one of the following according to the relation.
(1) predicate-argument relation (c.f. Figure 5(a)): Find a head-path p from R to a leaf node A in the T. Create a spine-etree T_s in which the copy of the nodes on p is the spine (R^* is the root, A^* is the anchor of T_s.). Copy the children (nodes only, not the whole subtrees) of each non-link nodes x. In T_s, merge x^* with its head-child if x is a link node on p, and mark all leaf nodes except A^* as substitution nodes.
Repeat A-B for x in *ttree* if x is a link node on p or x^* is a substitution node in T_s.

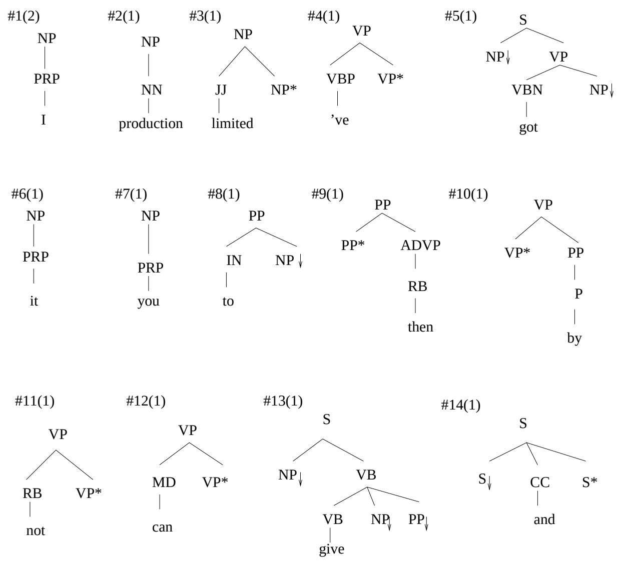

Figure 7: The extracted *etrees* from the *ttree* in Figure 6.

(2) modification relation (c.f. Figure 5(b)): At this stage, hc should have only one sister, call it *mod*. Create a mod-etree T_m in which R^* is the root, hc^* is the foot node, mod^* is hc's sister. Find a head-path p from *mod* to leaf node A in the *ttree*. Build an *etree* T_s from the path as stated in (1). In T_m, replace mod^* with T_s.
Repeat step A-B for x in *ttree* if x is a link node on p or x^* is a substitution node in T_s.
(3) coordination relation (c.f. Figure 5(c)): At this stage, hc should have a conjunction sister *conj* and another sister *coord*. Create a conj-etree T_c in which R^* is the root, hc^* is the foot node, $conj^*$ is the anchor, and $coord^*$ is the substitution node.
Repeat step A-B for *coord*.

Figure 7 shows the *etrees* extracted from the *ttree* examples. The numbers inside the parentheses are the counts of the *etrees*.

3.3 Filter out invalid *etrees*

For a large corpus like Penn Treebank, annotation errors are inevitable. The errors in *ttrees* will result in wrong *etrees*. An *etree* is called *invalid* if it doesn't satisfy some linguistic requirement. Right now, we check whether modifier-modifiee pairs in mod-etrees and parent-headchild and head-argument pairs in spine-etrees are allowed in that language.[11] Among the 14 etrees in Figure 7, tree #9 will be correctly

[8] Empty categories require special treatment. Due to the space limitation, we can not elaborate our strategy in this paper. The main idea is to use co-indices between the traces and the antecedents on the *ttree* to group *etrees* together (i.e. multi-component LTAGs).

[9] Each link node and its children form a recursive structure that will map to mod-etrees or conj-etrees. When we build spine-etrees, those structures should be factored out.

[10] Each node in *ttree* may correspond to up to two nodes in two different *etrees*. We will not get into the details here.

[11] One extra table is required for this task which lists possible modifiers for each b-tag. To check parent-headchild pairs, we can use head percolation table.

ruled out by our filter because ADVP can not modify PP from the right.[12]

An alternative is to use a threshold to throw away infrequent trees. This approach has two drawbacks: first, it will throw away some infrequent but valid *etree* templates; second, it can not rule out frequent but invalid *etrees*. In fact, 11.7% of invalid *etree* templates occur 10 or more times in the corpus. For example, one of such invalid *etrees* occurs 1832 times. In that etree, prepositions modify numbers, such as *about* modifies *160* in the *ttree*

```
(NP (QP (IN about) (CD 160))
    (NNS workers))
```

The *etree* is created from the *ttree* because in this context *about* is often mis-tagged as a preposition in the Treebank. It should be tagged as an adverb.

4 The Experiment

We ran our algorithm on the Penn Treebank II. Because the number of etrees is directly related to the number of words in the corpus, we count the number of *etree templates*[13] instead.The results are listed in Table 2. LTAG G_i is the grammar extracted from the Treebank. G_0 uses the original partially bracketed Treebank, and G_1 and G_2 use the fully bracketed corpus. [14] The *etrees* in G_1 uses Treebank's tagset which has 72 non-punctuation tags, while G_2 uses a reduced tagset (44 non-punctuation tags) where some tags such as VB, VBN, VBP are merged into a single tag. Each occurrence of a template in the corpus is counted as a template token. The table shows that fully bracketing the *ttrees* before extracting *etrees* reduce the number of extracted templates tremendously. Using a smaller tagset further reduces that number. Notice that the number of valid templates actually decreases without full bracketing because many templates extracted from partial bracketed corpus will have both arguments and modifiers as the sisters of predicates, which will be judged as invalid by our filter.

[12]Tree #9 and #10 were created because the word *then* was incorrectly tagged as RB. In this context, it is a temporal noun. Notice an error in *ttree* may cause errors in multiple *etrees*.

[13]If we abstract away from the individual lexical items in each elementary tree, we get an *elementary tree template*. Each template indicates where the anchor of that tree will be instantiated. Each *etree* is a (lexical-item, *etree template*) pair. e.g. *etree* #1, #6 and #7 in Figure 7 share the same etree template.

[14]Neumann (Neumann, 1998) converts sentences in section 02-04 in Penn Treebank to a lexicalized tree grammar. He does not make argument/adjunct distinction, and therefore he does not factor out the recursive structures in the elementary trees. In that sense, his approach is similar to the second step of our algorithm (c.f. Section 3.2). Not surprisingly, his result (11979 elementary tree templates from about 8.8% of *ttrees* in Treebank) is similar to the size of in G_0 (34307 templates from the whole corpus).

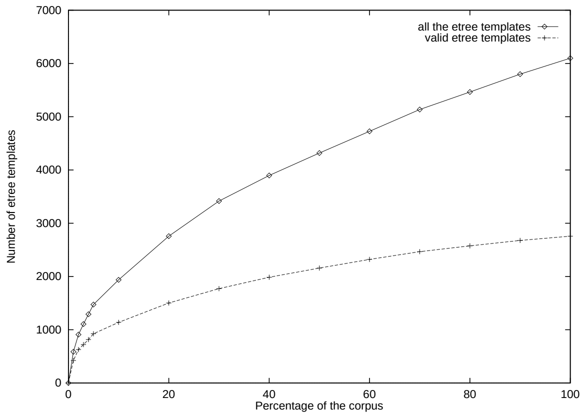

Figure 8: The growth of etree templates

Two questions come to mind. First, from the grammar developer's viewpoint, is the set of extracted templates complete enough to cover the language? The answer is *No*. Figure 8 shows the number of templates converges slowly as the size of the corpus grows, implying there are many unseen templates. On the other hand, interestingly, out of 48889 *ttrees*, 2286 (4.67%) can produce all the valid templates.[15] This implies the size of corpus needed for grammar extraction can be greatly reduced if we could find the right *ttrees*.

We have just showed that the set of templates extracted from one million word is not complete enough to cover the language, the next question is: how often do the unseen templates occur in new data? To answer the question, we calculate the percentage of (word, template) pairs in Treebank section 23 which do not occur in section 2-21.[16] In the unseen pairs, the words can be unseen (*uw*) or seen(*sw*), similarly, the templates can be unseen (*ut*) or seen (*st*), so there are four combinations. (sw, st) means both words and templates have been seen in section 2-21, but not the pair. The percentage of each type is listed in Table 3.3. We also list the percentage of unseen (word, POS tag) pairs in same section for comparison. The table shows the percentage of words with unseen templates is very low, 0.27% for G_1 and 0.09% for G_2, i.e. unseen templates represent only 3.6% of new pairs in G_1, and 1.5% in G_2. So the fact that the extracted template set is incomplete does not seem to have a big impact on parsing. Notice that the percentage of (sw, st) is much higher than (uw, ut) plus (sw, ut), implying some type of smoothing over sets of templates (e.g. the notion of tree families in XTAG English

[15]We just count the the *ttrees* that produce the first occurrence of *etrees* as the extraction process goes. If we restrict the length of *ttrees* to be less than 31 words, 2308 such *ttrees* can generate all but 35 valid templates.

[16]Several state-of-art parsers(e.g. (Collins, 1997)) are trained on section 2-21, and tested on section 23.

	# of etree templates(types)	# of valid etree template types	% of valid template types	% of valid template tokens
LTAG G_0	34307	1157	3.37%	19.5%
LTAG G_1	6099	2757	45.2%	96.1%
LTAG G_2	3014	895	29.7%	96.2%

Table 2: *Etree* templates extracted from the Treebank

	# of tags	(sw, st)	(uw, st)	(sw, ut)	(uw, ut)	total
POS tags	48	0.44%	2.47%	0	0	2.91%
LTAG G_1	6099	4.68%	2.45%	0.25%	0.02%	7.40%
LTAG G_2	3014	3.42%	2.46%	0.08%	0.01%	5.97%

Table 3: The types of unknown (word, template) pairs in Treebank section 23

grammar(XTAG-Group, 1995)) is desirable.

5 Conclusion

We have outlined an algorithm for extracting lexicalized tree adjoining grammars from partially bracketed corpora. This approach has a few advantages over hand-crafted grammars: First, it requires little human effort – the head percolation table etc. should take at most a couple of hours to build – if bracketed corpora are available. Second, the extracted grammar covers the corpus, i.e. the corpus can be seen as a collection of derived trees for the grammar, and therefore the corpus can be used to train statistical LTAG parsers directly.

The number of templates extracted from the Treebank converges slowly, implying there are many new templates outside the corpus. On the other hand, 4.67% of the Treebank generates the same number of valid templates as the whole Treebank. This implies that if there is no bracketed corpus available and we want to build a new one, the sampling of the sentences in the corpus is crucial if our goal is to generate a wide-coverage grammar from it.

References

Breckenridge Baldwin, Christine Doran, Jeffrey Reynar, Michael Niv, B. Srinivas, and Mark Wasson. 1997. EAGLE: An Extensible Architecture for General Linguistic Engineering. In *Proceedings of RIAO97*, Montreal.

Ann Bies, Mark Ferguson, Karen Katz, and Robert MacIntyre. 1995. Bracketing guidelines for treebank ii style penn treebank project.

R. Chandrasekar and B. Srinivas. 1997. Gleaning information from the web: Using syntax to filter out irrelevant information. In *Proceedings of AAAI 1997 Spring Symposium on NLP on the World Wide Web.*

Eugene Charniak. 1996. Treebank grammars. In *Proceedings of AAAI-96.*

Mike Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th ACL.*

Aravind Joshi and Yves Schabes. 1997. Tree adjoining grammars. In A. Salomma and G. Rosenberg, editors, *Handbook of Formal Languages and Automata.* Springer-Verlas, Herdelberg.

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

Alexander Krotov et al. 1998. Compacting the penn treebank grammar. In *Proceedings of ACL-COLING.*

David M. Magerman. 1995. Statistical Decision-Tree Models for Parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics.*

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, et al. 1994. The penn treebank: annotating predicate argument structure. In *In Proc of ARPA speech and Natural language workshop.*

K. F. McCoy, K. Vijay-Shanker, and G. Yang. 1992. A functional approach to generation with tag. In *Proceedings of the 30th ACL.*

Gunter Neumann. 1998. Automatic extraction of stochastic lexicalized tree grammars from treebanks. In *Fourth International Workshop on TAG and Related Frameworks(TAG+4).*

Martha Palmer, Owen Rambow, and Alexis Nasr. 1998. Rapid prototyping of domain-specific machine translation system. In *Proceedings of ATMA-98*, Langhorne, PA, October.

Beatrice Santorini. 1990. Part-of-speech tagging guidelines for the penn treebank project. Technical report, Dept of Computer and Information Science, University of Pennsylvania.

Kiyoaki Shirai, Takenobu Tokunaga, and Hozumi Tanaka. 1995. Automatic extraction of japanese grammar from a bracketed corpus. In *Proceedings of Natural Language Processing Pacific Rim Symposium.*

Bangalore Srinivas. 1997. *Complexity of Lexical Descriptions and Its relevance to Partial Parsing.* Ph.D. thesis, University of Pennsylvania.

Matthew Stone and Christine Doran. 1997. Sentence planning as description using tree adjoining grammar. In *Proceedings of the 35th ACL.*

The XTAG-Group. 1995. A Lexicalized Tree Adjoining Grammar for English. Technical Report IRCS 95-03, University of Pennsylvania.