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), 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 (Chandrasekhar 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.

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 Talashashi (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.
Table 1: Treebank tags which appear in this paper

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVP</td>
<td>adverb phrase</td>
</tr>
<tr>
<td>NP</td>
<td>noun phrase</td>
</tr>
<tr>
<td>PP</td>
<td>preposition phrase</td>
</tr>
<tr>
<td>VP</td>
<td>verb phrase</td>
</tr>
<tr>
<td>S</td>
<td>sentence</td>
</tr>
<tr>
<td>CC</td>
<td>coordinating conjunction</td>
</tr>
<tr>
<td>IN</td>
<td>preposition</td>
</tr>
<tr>
<td>JJ</td>
<td>adjectives</td>
</tr>
<tr>
<td>MD</td>
<td>modal</td>
</tr>
<tr>
<td>NN</td>
<td>noun, singular or mass</td>
</tr>
<tr>
<td>NNS</td>
<td>noun, plural</td>
</tr>
<tr>
<td>PRP</td>
<td>pronoun</td>
</tr>
<tr>
<td>RB</td>
<td>adverb</td>
</tr>
<tr>
<td>VB</td>
<td>verb, base form</td>
</tr>
<tr>
<td>VBN</td>
<td>verb, past participle</td>
</tr>
<tr>
<td>VBP</td>
<td>verb, non-3rd singular present</td>
</tr>
<tr>
<td>'</td>
<td>comma</td>
</tr>
<tr>
<td>&quot;</td>
<td>quotation mark</td>
</tr>
<tr>
<td>DTV</td>
<td>dative</td>
</tr>
<tr>
<td>SBJ</td>
<td>subject</td>
</tr>
<tr>
<td>TMP</td>
<td>temporal</td>
</tr>
</tbody>
</table>

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 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, α1, α2 and α3 are initial trees anchored by got, I and production respectively. β1 and β2 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, α2 substitutes at NP0 in α1, β2 adjoins to the NP node in α3, then the whole tree substitutes into the NP1 node in α1, meanwhile, β1 adjoins to the VP node in α1, 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 etree 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 $X^m \rightarrow Y^gX^m-1$, ..., $X^1 \rightarrow XZ^p$ in one tree; a mod-etree corresponds to the rule $W^g \rightarrow WYX^m$ and $X^m$ is further expanded to include its arguments.
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 its parent. 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 $W^0$, 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 $W^0$. 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 ttrees.

**Step A**: Choose the head-child 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 and $R$ 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 output7 is shown in Figure 6. The nodes inserted by the algorithm are in bold font.

4Coordination 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.

5The 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 $X$, $X^1$, ..., $X^m$.

6The 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. phases with -DTV 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.

7The 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.
3.2 Building etrees

Once the ttrees are fully bracketed, the extraction algorithm is straightforward. Let’s 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. The extraction process involves copying nodes $x$ from ttrees to etrees. Let’s denote the copy of $x$ as $x^*$. 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 siblings, 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 6: The fully bracketed structure for the example in Figure 1

$$((S (NP-SBJ (PRP I))\ (VP (VBP 've)) (VP (VBN got)) (NP (JJ limited))) (CC and) (S (NP-SBJ (PRP I)) (VP (RB not)) (VP (RB give)) (NP (PRP you))))$$

Figure 7: The extracted etrees from the ttrees 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 $T_e$ from the path as stated in (1). In $T_m$, replace $mod^*$ with $T_e$.

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 ttrees. 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-modifier pairs in mod-etrees and parent-headchild and head-argument pairs in spine-etrees are allowed in that language. Among the 14 etrees in Figure 7, tree #9 will be correctly

---

8Empty 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 tree to group etrees together (i.e. multi-component LTAGs).

9Each 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.

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

11One 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.
rulled out by our filter because ADVP can not modify PP from the right.\footnote{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 \textit{tree} may cause errors in multiple \textit{etrees}.}

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 \textit{etree} templates; second, it can not rule out frequent but invalid \textit{etrees}. In fact, 11.7\% of invalid \textit{etree} templates occur 10 or more times in the corpus. For example, one of such invalid \textit{etrees} occurs 1832 times. In that \textit{etree}, prepositions modify numbers, such as \textit{about} modifies 160 in the \textit{tree}.

$$(\text{NP (QP (IN about) (CD 160))})$$

\text{(NNS workers)}

The \textit{etree} is created from the \textit{tree} because in this context \textit{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 \textit{etree} templates\footnote{If we abstract away from the individual lexical items in each elementary tree, we get an \textit{elementary tree template}. Each template indicates where the anchor of that tree will be instantiated. Each \textit{etree} is a \textit{(lexical-item, etree template)} pair. e.g. \textit{etree} \#1, \#6 and \#7 in Figure 7 share the same \textit{etree} template.} instead. The results are listed in Table 2. LTAG $G_1$ 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.\footnote{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 \textit{trees} in Treebank) is similar to the size of in $G_0$ (31407 templates from the whole corpus).} The \textit{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 \textit{trees} before extracting \textit{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.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{fig8.png}
\caption{The growth of etree templates}
\end{figure}

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 \textit{trees}, 2286 (4.67\%) can produce all the valid templates.\footnote{We just count the the \textit{trees} that produce the first occurrence of \textit{etrees} as the extraction process goes. If we restrict the length of \textit{trees} to be less than 31 words, 2308 such \textit{trees} can generate all but 35 valid templates.} This implies the size of corpus needed for grammar extraction can be greatly reduced if we could find the right \textit{trees}.

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.\footnote{Several state-of-art parsers (e.g. \cite{Collins, 1997}) are trained on section 2-21, and tested on section 23.} 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 some 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]
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.

Table 2: Etree templates extracted from the Treebank

<table>
<thead>
<tr>
<th># of etree templates(templates)</th>
<th># of valid etree template types</th>
<th>% of valid template types</th>
<th>% of valid template tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTAG G₀</td>
<td>34307</td>
<td>1157</td>
<td>3.37%</td>
</tr>
<tr>
<td>LTAG G₁</td>
<td>6099</td>
<td>2757</td>
<td>45.2%</td>
</tr>
<tr>
<td>LTAG G₂</td>
<td>3014</td>
<td>895</td>
<td>29.7%</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th># of tags</th>
<th>(sw, st)</th>
<th>(uw, st)</th>
<th>(sw, ut)</th>
<th>(uw, ut)</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>POS tags</td>
<td>48</td>
<td>0.44%</td>
<td>2.47%</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LTAG G₁</td>
<td>6099</td>
<td>4.68%</td>
<td>2.45%</td>
<td>0.25%</td>
<td>0.02%</td>
</tr>
<tr>
<td>LTAG G₂</td>
<td>3014</td>
<td>3.42%</td>
<td>2.46%</td>
<td>0.08%</td>
<td>0.01%</td>
</tr>
</tbody>
</table>

References