Evaluating the Coverage of LTAGs on Annotated Corpora

Fei Xia and Martha Palmer

Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104, USA
{fxia,mpalmer}@linc.cis.upenn.edu

Abstract
Lexicalized Tree Adjoining Grammars (LTAGs) have been applied to many NLP applications. Evaluating the coverage of a LTAG is important for both its developers and its users. In this paper, we describe a method, which estimates a grammar’s coverage on annotated corpora by first automatically extracting a Treebank grammar from the corpus and then calculating the overlap between the two grammars. We used the method to test the coverage of the XTAG grammar, which is a large-scale hand-crafted grammar for English, on the English Penn Treebank, and the result shows that the grammar can cover at least 97.2% of template tokens in the Treebank. This method has several advantages: first, the whole process is semi-automatic and requires little human effort; second, the coverage can be calculated at sentence level or more fine-grained levels, third, the method provides a set of new templates that can be added to the grammar to improve its coverage. Fourth, there is no need to parse the corpus.

1. Introduction

A Lexicalized Tree Adjoining Grammar (LTAG) consists of a finite set of lexicalized trees (elementary trees) and composition operations of substitution and adjunction. LTAGs have been applied to many NLP applications. Evaluating the coverage of a LTAG is important for both its developers and its users.

Previous evaluations (Doran et al., 1994; Srinivas et al., 1998) of LTAGs used unannotated data (i.e. a set of sentences without syntactic bracketing). The data are first parsed by a LTAG parser and the coverage of the grammar is measured as the percentage of sentences in the data that get at least one parse. For more discussion on this approach, see (Prasad and Sarkar, 2000).

In this paper, we propose a new evaluation method that takes advantage of large annotated corpora (i.e. Treebanks) and a grammar extraction tool (Xia, 1999). The tool extracts LTAGs from Treebanks automatically. Using the tool, the coverage of a hand-crafted grammar can be measured by the overlap of the grammar and the Treebank grammar. This method has several advantages. First, the whole process is semi-automatic and requires little human effort; Second, the coverage can be calculated at either sentence level or elementary tree level, which is more fine-grained. Third, the method provides a list of elementary trees that can be added to the grammar to improve its coverage. Fourth, there is no need to parse the whole corpus, which could have been very time-consuming.

2. LTAG formalism

LTAGs are based on the Tree Adjoining Grammar 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 (etrees for short). Each etree 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. There are two types of etrees: 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. Leaf nodes other than anchors and foot nodes are substitution nodes.

Etrees are combined by operations substitution and adjunction. In the substitution operation (Figure 1), a substitution node in an etree is replaced by another etree whose root has the same label as the substitution node. In an adjunction operation (Figure 2), 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 etrees is called a derived tree.

![Figure 1: The substitution operation](image)

![Figure 2: The adjunction operation](image)

Figure 3: Etrees and the derived tree for the sentence who worried about the flood

In Figure 3, the top five structures are the etrees anchored by words in a wh-question who worried about the flood. Foot and substitution nodes are marked by *, and ↓ respectively. The arrows between the trees illustrate the combining process, and γ is the derived tree.

3. The XTAG grammar and the English Penn Treebank

In this paper, we will report our experiments on evaluating the coverage of the XTAG grammar on the English Penn Treebank. The XTAG grammar (XTAG-Group, 1998) is a large-scale Tree Adjoining Grammar for English, which has been developed at University of Pennsylvania since the early 1990s. The current XTAG grammar has about 1.8 million etrees and has 1004 tree templates.\(^1\)

The English Penn Treebank (Marcus et al., 1994) has about 1 million words from the Wall Street Journal. The sentences in the Treebank are bracketed with syntactic structures. The average sentence length is 23 words.

4. Methodology

The main idea of our evaluation method is as follows: given a Treebank \(T\) and a grammar \(G_h\), if we use the grammar extraction tool to extract a Treebank grammar, \(G_t\), from \(T\), then the coverage of \(G_t\) can be measured as the percentage of \(T\) which are covered by the intersection of \(G_t\) and \(G_h\). The Treebank and \(G_h\) may choose different analyses for certain syntactic constructions. As a result, although some constructions are covered by both grammars, the corresponding elementary trees in these grammars would look very different. To account for this, our method has several steps:

1. Extract a Treebank grammar from \(T\). Let \(G_t\) be the set of templates in the Treebank grammar.

2. Put into \(G_t\) all the templates in \(G_t\) which match some templates in \(G_h\).

3. Check each template in \(G_t - G_t\) and decide whether the construction represented by the template is handled differently in \(G_h\). If so, put the template in \(G_t''\).

The coverage of \(G_h\) on \(T\) is measured as \(\frac{\text{count}(G_t' \cup G_h'')}{\text{count}(G_t)}\). The templates in \(G_t - G_t' - G_h''\) are the ones that are truly missing from \(G_h\). They should be checked and the plausible ones can be added to \(G_h\) to improve \(G_h\)'s coverage. In this paper, we are focusing on general syntactic structures in two grammars, not on the completeness of lexicons. Therefore, for grammar coverage we use templates, instead of etrees. The method can be easily extended to compare etrees. The next three sections will describe each step of the evaluation method.

\(^1\)If we remove the anchor from each etree, we get tree templates. Each etree can be seen as a (word, template) pair.
5. LexTract and a Treebank grammar

We have built a grammar development tool, called LexTract, for grammar extraction. The architecture of LexTract is shown in Figure 4, with the components relevant to the grammar evaluation task in boldface.

Figure 4: Architecture of LexTract

By design all the etrees extracted from the Treebank by LexTract fall into one of three types according to the relations between the anchor of the etree and other nodes in the tree, as in Figure 5.

We will use an example to illustrate the main steps of the extraction algorithm. The input to the algorithm is a bracketed sentence from the Penn Treebank (we call it a ttree), as in Figure 6(a).

The ttree is partially bracketed in that arguments and modifiers for the same head are siblings of the head. In LTAG, arguments appear in spine-etrees and modifiers in mod-etrees, and each mod-etree takes exactly one modifier. To account for this difference, the algorithm first fully brackets the ttrees by adding intermediate nodes so that at each level the siblings have one of three relations: predicate-argument relation, modification relation, or coordination relation. The fully bracketed sentence is shown in Figure 6(b). The nodes inserted by the algorithm are circled. We map the Treebank tagset to the XTAG tagset to compare the Treebank grammar with the XTAG.

The next step builds an etree set \( E \) from each fully bracketed ttree. Recursive structures become mod-etrees or conj-etrees, and the remaining structures become spine-etrees. If we treat each node in a ttree as a (top, bottom) pair, \( E \) in fact forms a decomposition of the ttree. The ttree in Figure 7 is the same as the one in Figure 6(b) and as \( \gamma_1 \) in Figure 3 except that in the former some nodes are split into (top, bottom) pairs. The ttree yields five etrees, the same ones as in Figure 3. Notice that the two subtrees of \( \alpha_2 \) are separated by the auxiliary tree \( \beta_1 \).

We ran the algorithm on the Penn English Treebank II and extracted 2890 templates.

6. Matching templates in the two grammars

To calculate the coverage of the XTAG grammar, we need to find out how many templates in the Treebank grammar match some templates in the XTAG grammar. We define two types of matching: \( t\text{-match} \) and \( e\text{-match} \). From now on, we use XTAG and ExtG to stand for the XTAG grammar and the extracted grammar respectively.

---

2Recall that when a pair of etrees are combined during parsing, the root of one etree is merged with a node in the other etree. Splitting nodes into top and bottom pairs during the decomposition of a fully bracketed ttree is the reverse process of merging nodes during parsing. For the sake of simplicity, we show the top and the bottom parts of a node \( X \), denoted as \( X.t \) and \( X.b \) respectively, only when the two parts will end up in different etrees.
Figure 7: The etree set is a decomposition of the ttree.

6.1. t-match

We call two trees t-match (t for tree) if they are identical barring the type of information present only in one grammar, such as feature structures and subscripts in XTAG and frequency information in ExtG. In Figure 8, XTAG trees in 8(a) and 8(b) t-match the ExtG tree in 8(c).

Figure 8: An example of t-match

XTAG also differs from ExtG in that XTAG includes multi-anchor trees to handle idioms (Figure 9(a)), light verbs (Figure 9(b)) and so on. In each of these cases, the multi-anchors form the predicate. These trees are the same as the spine-tree in Figure 5(a) except that some nodes of the XTAG trees (e.g. NP1 in Figure 9(a) and its counterpart Zp in Figure 5) are expanded. By having multi-anchors, each tree can be associated with semantic representations directly (as shown in Figure 9), which is an advantage of LTAG formalism. ExtG does not have multi-anchor trees because semantics is not marked in the Treebank and consequently the extraction algorithm can not distinguish idiomatic meanings from literal meanings. Since expanded subtrees are present only in XTAG, we disregard them when comparing templates.

Figure 9: Templates in XTAG with expanded subtrees t-match the one in ExtG when the expanded subtrees are disregarded

6.2. c-match

t-match requires two trees to have exactly the same structure barring expanded subtrees, therefore, it does not tolerate minor annotation differences between the two grammars. For instance, in XTAG, relative pronouns such as which and the complementizer that occupy distinct positions in the etree for relative clauses, whereas the Penn Treebank treats both as pronouns and therefore they occupy the same position in ExtG, as shown in Figure 10. Because the circled subtrees will occur in every tree for relative clauses and wh-movement, all these trees will not t-match their counterparts in the other grammar. Nevertheless, the two trees share the same subcategorization frame (NP V NP), the same subcategorization chain3 S → VP → V and the same modification pair (NP, S). To capture this kind of similarity, we decompose a mod etree into a tuple of (subcat frame, subcat chain, modification pair). Similarly, a spine etree is decomposed into (subcat frame, subcat chain) pair, and a conj etree into (subcat frame, subcat chain, coordination sequence). Two etrees are said to c-match (c for component) if they are decomposed into the same tuples. According to this definition, in Figure 10 the two templates c-match.

---

3A subcategorization chain is a subsequence of the spine in a spine etree where each node on the chain is a parent of some argument(s) in the subcategorization frame. The nodes on a subcategorization chain roughly correspond to various lexical projections in GB-theory.
### 6.3. Matching results

So far, we have defined two types of matching. Notice that both types of matching are not one-to-one. Table 1 lists the numbers of matched templates in two grammars. The last row lists the frequencies of the matched ExtG templates in the Treebank. For instance, the second column says 162 templates in XTAG \( t \text{-match} \) 54 templates in ExtG, and these 54 templates account for 54.6% of the template tokens in the Penn Treebank.

One of the major differences between the XTAG and the Treebank annotation is that an adjective modifies a noun directly in the former whereas in the latter an adjective projects to an AP which in turn modifies an NP, as shown in Figure 11. Similarly, in XTAG an adverb modifies a VP directly, whereas in the Treebank an adverb sometimes projects to an ADVP first. If we disregard these annotation differences, the percentage of matched template tokens increases from 59.9% to 82.1%, as shown in Table 2. The magnitude of the increase is due to the high frequency of templates with nouns, adjectives and adverbs.

### Table 1: Matched templates and their frequencies

<table>
<thead>
<tr>
<th></th>
<th>( t \text{-match} )</th>
<th>( c \text{-match} )</th>
<th>matched subtotal</th>
<th>unmatched subtotal</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTAG</td>
<td>162</td>
<td>314</td>
<td>476</td>
<td>528</td>
<td>1004</td>
</tr>
<tr>
<td>ExtG</td>
<td>54</td>
<td>133</td>
<td>187</td>
<td>2703</td>
<td>2890</td>
</tr>
<tr>
<td>frequency</td>
<td>54.6%</td>
<td>5.3%</td>
<td>59.9%</td>
<td>40.1%</td>
<td>100%</td>
</tr>
</tbody>
</table>

### Table 2: Matched templates when certain annotation differences are disregarded

<table>
<thead>
<tr>
<th></th>
<th>( t \text{-match} )</th>
<th>( c \text{-match} )</th>
<th>matched subtotal</th>
<th>unmatched subtotal</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTAG</td>
<td>173</td>
<td>324</td>
<td>497</td>
<td>507</td>
<td>1004</td>
</tr>
<tr>
<td>ExtG</td>
<td>81</td>
<td>134</td>
<td>215</td>
<td>2675</td>
<td>2890</td>
</tr>
<tr>
<td>frequency</td>
<td>78.6%</td>
<td>3.5%</td>
<td>82.1%</td>
<td>17.9%</td>
<td>100%</td>
</tr>
</tbody>
</table>

### Figure 10: An example of \( c \text{-match} \)

(a) in XTAG

(b) in ExtG

### Figure 11: Templates for adjectives modifying nouns

(a) in XTAG

(b) in ExtG

### 7. Classifying unmatched templates

The previous section shows that 17.9% of the template tokens do not match any template in the XTAG grammar. This is due to several reasons:

**T1: incorrect templates in ExtG** These templates result from Treebank annotation errors, and therefore, are not in XTAG.

**T2: coordination in XTAG** The templates for coordinations in XTAG are generated on-the-fly while parsing (Sarkar and Joshi, 1996), and are not part of the 1004 templates. Therefore, the \( \text{conj	ext{-trees}} \) in ExtG, which account for 3.4% of the template tokens in the Treebank, do not match any templates in XTAG.

**T3: alternative analyses** XTAG and ExtG often choose different analyses for the same phenomenon. For example, the two grammars treat reduced relative clauses differently. As a result, the templates used to handle those phenomena do not match each other by our definition.
T4: constructions not covered by XTAG

Some of such constructions are the unlike coordination phrase (UCP), parenthetical (PRN), fragment (FRAG) and ellipsis.

For the first three types, the XTAG grammar can handle the corresponding constructions although the templates used in two grammars look very different and do not match according to our definition.

To find out what constructions are not covered by XTAG, we manually classify 289 of the most frequent unmatched templates in ExtG according to the reason why they are absent from XTAG. These 289 templates account for 93.9% of all the unmatched template tokens in the Treebank. The results are shown in Table 3, where the percentage is with respect to all the tokens in the Treebank. From the table, it is clear that most unmatched template tokens are due to alternative analyses (T3) adopted in the two grammars. Combining the results in Table 2 and 3, we conclude that 97.2% of template tokens in the Treebank are covered by XTAG, while another 1.7% are not. Because the remaining 2386 unmatched templates in ExtG have not been checked, it is not clear how many of the remaining 1.1% template tokens are covered by XTAG.  

<table>
<thead>
<tr>
<th>type</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>freq</td>
<td>1.1%</td>
<td>3.4%</td>
<td>10.6%</td>
<td>1.7%</td>
<td>16.8%</td>
</tr>
</tbody>
</table>

Table 3: Classifications of 289 unmatched templates

8. Conclusion

We have presented a method for evaluating the coverage of a LTAG grammar on an annotated corpus. It first uses LexTract to automatically extract a Treebank grammar, then the templates in the Treebank grammar are matched with the ones in the grammar to be evaluated, next the unmatched templates in the Treebank grammar are classified so that we can determine how many of them are due to missing constructions in the latter grammar. We have tested the method with the XTAG grammar and the English Penn Treebank and the result shows that the XTAG grammar can cover at least 97.2% of the template tokens in the Treebank.

This method has several advantages: first, the whole process is semi-automatic and requires little human effort; second, the coverage can be calculated at sentence level, template level and sub-structure level; third, the method provides a list of templates that can be added to the grammar to improve its coverage; fourth, there is no need to parse the whole corpus, which could have been very time-consuming.

9. References


