A Treebank Query System Based on an Extracted Tree Grammar

Seth Kulick and Ann Bies
Linguistic Data Consortium
University of Pennsylvania
3600 Market St., Suite 810
Philadelphia, PA 19104
{skulick,bies}@ldc.upenn.edu

Abstract

Recent work has proposed the use of an extracted tree grammar as the basis for treebank analysis and search queries, in which queries are stated over the elementary trees, which are small chunks of syntactic structure. However, this work was lacking in two crucial ways. First, it did not allow for including lexical properties of tokens in the search. Second, it did not allow for using the derivation tree in the search, describing how the elementary trees are connected together. In this work we describe an implementation that overcomes these problems.

1 Introduction

(Kulick and Bies, 2009) describe the need for treebank search that compares two sets of trees over the same tokens. Their motivation is the problem of comparing different annotations of the same data, such as with inter-annotator agreement evaluation during corpus construction. The typical need is to recognize which annotation decisions the annotators are disagreeing on. This is similar to the problem of determining where the gold trees and parser output differ, which can also be viewed as two annotations of the same data.

As they point out, for this purpose it would be useful to be able to state queries in a way that relates to the decisions that annotators actually make, or that a parser mimics. They provide examples suggesting that (parent, head, sister) relations as in e.g. (Collins, 2003) are not sufficient, and that what is needed is the ability to state queries in terms of small chunks of syntactic structure.

Their solution is to use an extracted tree grammar, inspired by Tree Adjoining Grammar (Joshi and Schabes, 1997). The “elementary trees” of the TAG-like grammar become the objects on which queries can be stated. They demonstrate how the “lexicalization” property of the grammar, in which each elementary tree is associated with one or more token, allows for the queries to be carried out in parallel across the two sets of trees.

However, the work was lacking in two crucial ways. First, it did not allow for including lexical properties of a token, such as its Part-of-Speech tag, together with the elementary tree search. This made it impossible to formulate such queries as “find all ADVP elementary trees for which the head of the tree is a NOUN NUM”. Even more seriously, there was no way to search over the “derivation tree”, which encodes how the extracted elementary trees combine together to create the original tree. This made it impossible to carry out searches such as “find all verb frames with a PP-LOC modifying it”, and in general to search for the crucial question of where annotators disagree on attachment decisions.

In this paper we describe how we have solved these two problems.

2 Tree Extraction

Following (Kulick and Bies, 2009), we draw our examples from the Arabic Treebank. For our gram-
We will use "etree" as shorthand for "elementary tree".

3This derivation tree is slightly simplified, since with sister-adjunction it includes more information to indicate the direction and order of attachment.
LEX : (L1) text="fiy"
ETREE: (E1) (S (VP A$ NP[t]-SBJˆ{dta:1}))
     (E2) (PP A${lex:L1} NP^)
DTREE: (D1) E2
     (D2) (E1 E2{dta:1})

Figure 4: Examples of one lexical restriction, two etree queries, and two dtree queries

The section of ATB we are working with has 402,246 tokens, resulting in 319,981 etree instances
and only 2804 etree templates, which gives an indication of the huge amount of duplication of structure
in a typical treebank representation. From the perspective of database organization, the representation
of the etree templates can be perhaps be viewed as a type of database “normalization”, in which duplicate
information is placed in a separate table.

3 Query Processing

We now describe the algorithm used for searching on the database with the extracted tree grammar, focusing
on how the algorithm now allows searching based on the derivation tree and lexical information.

Queries are specified as "etree queries" and "dtree queries". Sample queries are shown in Figure 4. The query processing is as follows:

Step 1:
The etree templates are searched to determine which match a given etree query.\footnote{Each etree query has a "distinguished" anchor marked A$ that indicates the anchor (word) of an etree template that is associated with that query. The reason for that is that if an etree template has more than one anchor, we only want one to trigger that query, so that the etree is not counted twice.} This is a simple tree matching between each template and query, all of which are small small trees. It is within this tree matching that several of the typical relations can be specified, such as precedence and dominance. A table stores the information on which templates match which queries.

In addition, the Etree queries can now include two new properties. First, they can include a specification for a lexical restriction, such as lex:L1 in E2 in Figure 4. However, step 1 of the query processing does not actually check this, since it is simply going through each template, without examining any anchors, to determine which have the appropriate structure to match a query. Therefore, we store in another table the information that for a (template, query) to match it must be the case that an anchor at a particular address in that template satisfies a particular lexical restriction. It in effect produces specialized information for the given template as to what additional restrictions apply for that (template, query) pair to succeed as a match, in each etree instance that uses that etree template. For example, in this case the stored information specifies that an etree instance with template (PP A NP^) matches the query E2 if the instance has an anchor with the text fiy at address 1.1 (the anchor A).

Similarly, the etree query can include a specification dta (as in E1), for "derivation tree address", indicating that the corresponding address in each matching template needs to be stored for later reference in derivation tree searching. In this case, the template for etree instance #1 will match etree query E1, with the additional information stored that the address 1.1.2 will be used for later processing.

An important point here is that this additional information is not necessarily the same for the different templates that otherwise match a query. For example, the two templates

\begin{align*}
(1) & \quad (S (VP A NP[t]-SBJ<1.1.2>)) \\
(2) & \quad (SBAR (S (VP A NP[t]-SBJ<1.1.1.2>)))
\end{align*}

both match query E1, but for (1) the stored address dta:1 is 1.1.2, while for (2) the stored address is 1.1.1.2. The same point holds for the address of the anchor with a lexical restriction.

Step 2:
For a given query, the matching etree instances are found. First it finds all etree instances such that the (template, query) is a match for the instance’s etree template. It then filters this list by checking the lexical restriction, if any, for the anchor at the appropriate address in the etree instance, using the information stored from step 1. In the above example, this will select etree instance #4 as satisfying query E2, since the template for instance #4 was determined in step 1 to match E2, and the particular instance #4
also satisfies the lexical restriction in query \( E_2 \).

**Step 3:**
The final results are reported using the dtree queries. Some dtree queries are singletons naming an etree query, such as \( D_1 \), indicating that the dtree query is simply that etree query. In this example, any etree instance that satisfies the etree query \( E_2 \) is reported as satisfying the dtree query \( D_1 \).

The dtree query can also specify nodes in a derivation tree that must satisfy specified etree queries and also be in a certain relationship in the derivation tree. For example, dtree query \( D_2 \) in Figure 4 specifies that the query is for two nodes in a parent-child relationship in the derivation tree, such that the parent node is for an etree instance that satisfies etree query \( E_1 \), and the child is an instance that satisfies etree query \( E_2 \). Furthermore, the address in the derivation tree is the same as the address \( \text{dta:1} \) that was identified during Step 1. Note that the address is located on the parent tree during Step 1, but appears in the derivation tree on the child node.

Steps 1 and 2 identify etree instance #1 as satisfying etree query \( E_1 \), with \( \text{dta:1} \) stored as address \(<1.1.2>\) for the template used by instance #1. These steps also identified etree instance #4 as satisfying etree query \( E_2 \). Step 3 now determines that etree instances #1 and #4 are in a derivation tree relationship that satisfies dtree query \( D_2 \), by checking for a parent-child relationship between them with the address \(<1.1.2>\).

So dtree query \( D_1 \) is finding all PP etrees headed by "fly", and dtree query \( D_2 \) is finding all clauses with a subject after the verb, with a PP attaching next to the subject, where the PP is headed by "fly".

We consider the distinguished anchor (see footnote 4) for a dtree query to be the distinguished anchor of the parent node. The earlier work on comparing two sets of trees (Kulick and Bies, 2009) can then use this to report such searches as "the annotators agree on the same verbal structure, but one has a PP modification and the other does not".

## 4 Conclusion and Future Work

Our immediate concern for future work is to work closely with the ATB team to ensure that the desired queries are possible and are integrated into the work on comparing two sets of trees. We expect that this will involve further specification of how queries select etree templates (Step 1), in interesting ways that can take advantage of the localized search space, such as searching for valency of verbs.

We are also working on evaluation of the speed of this system, in comparison to systems such as (Ghodke and Bird, 2008) and Corpus Search. The search algorithm described above for derivation tree searches can be made more efficient by only looking for relevant etree instances in the context of walking down the derivation tree. In general, while searching for etree instances is very efficient, even with lexical restrictions, complex searches over the derivation tree will be less so. However, our hope, and expectation, is that the vast majority of real-life dtree queries will be local (parent, child, sister) searches on the derivation tree, since each node of the derivation tree already encodes small chunks of structure.

**Acknowledgements**

We thank Aravind Joshi, Anthony Kroch, Mitch Marcus, and Mohamed Maamouri for useful discussions. This work was supported in part by the Defense Advanced Research Projects Agency, GALE Program Grant No. HR0011-06-1-0003 (both authors) and by the GALE program, DARPA/CMO Contract No. HR0011-06-C-0022 (first author). The content of this paper does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

**References**

David Chiang. 2003. Statistical parsing with an automatically extracted tree adjoining grammar. In *Data Oriented Parsing*. CSLI.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29:589–637.

Sumukh Ghodke and Steven Bird. 2008. Querying linguistic annotations. In *Proceedings of the Thirteenth Australasian Document Computing Symposium*.

A.K. Joshi and Y. Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Volume 3*.

Seth Kulick and Ann Bies. 2009. Treebank analysis and search using an extracted tree grammar. In *Proceedings of The Eighth International Workshop on Treebanks and Linguistic Theories*.

---

\(^5\) It is also possible to specify the nature of that relationship by the attachment type, substitution or modification.

\(^6\) [http://corpussearch.sourceforge.net](http://corpussearch.sourceforge.net).