A TAG-derived Database for Treebank Search and Parser Analysis

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, in which queries are stated over the elementary trees, which are small chunks of syntactic structure. In this work we integrate search over the derivation tree with this approach in order to analyze differences between two sets of annotation on the same text, an important problem for parser analysis and evaluation of inter-annotator agreement.

1 Introduction

In earlier work (Kulick and Bies, 2009; Kulick and Bies, 2010; Kulick et al., 2010) we have described the need for a treebank search capability that compares two sets of trees over the same tokens. Our motivation is the problem of comparing different annotations of the same data, such as determining where gold trees and parser output differ. Another such case is that of comparing inter-annotator agreement files during corpus construction. In both cases the typical need is to recognize which syntactic structures the two sets of trees are agreeing or disagreeing on.

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. We refer to this earlier work for arguments 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.

The solution we take is to use an extracted tree grammar, inspired by Tree Adjoining Grammar (Joshi and Schabes, 1997). The "elementary trees" and the derivation trees of the TAG-like grammar are put into a MySql database, and become the objects on which queries can be stated. The "lexicalization" property of the grammar, in which each elementary tree is associated with one or more tokens, allows for the the queries to be carried out in parallel across the two sets of trees.

We show here how this approach can be used to analyze two types of errors that occur in parsing the Arabic Treebank. As part of this analysis, we show how search over the derivation tree, and not just for the elementary trees, can be used as part of this analysis of parallel annotations over the same text.

2 Elementary Tree Extraction

The work described and all our examples are taken from the Arabic Treebank, part 3, v3.2 (ATB3-v3.2) (Maamouri et al., 2010).

As discussed above, we are aiming for an analysis of the trees that is directly expressed in terms of the core syntactic constructions. Towards this end we utilize ideas from the long line of TAG-based research that aims to identify the smaller trees that are the "building blocks" of the full trees of that treebank, and that are then used for such purposes as training parsers or as a basis for machine translation systems (Chen, 2001; Chiang, 2003; Xia, 2001). However, as far as we know this approach has not been utilized for searching within a treebank, until the current line of work.

As in the earlier TAG work we use head rules to decompose the full trees and then extract out the "elementary trees", which are the small syntactic chunks. This decomposition of the full tree results

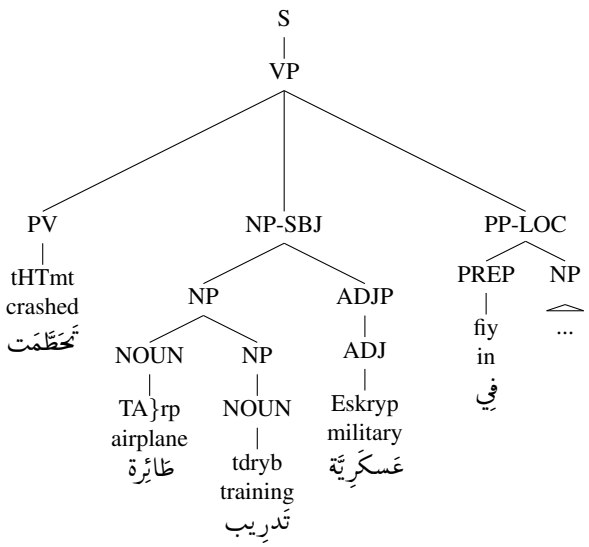

Figure 1: Sample tree

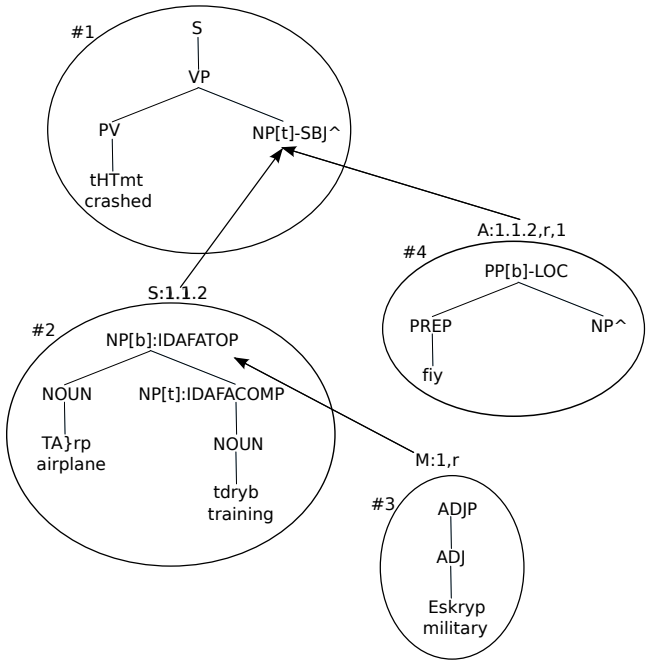

Figure 2: Elementary Trees and Derivation Tree for the Tree Decomposition in Figure 1

not just in these elementary trees, but also records how the elementary trees relate to each other, and therefore how they can be recombined to form the original full tree. For our grammar we use a TAG variant with tree-substitution, sister-adjunction, and Chomsky-adjunction (Chiang, 2003).

A small example is shown in Figures 1 and 2.[1] The full tree is shown in Figure 1, and the extracted elementary trees[2] and derivation tree in Figure 2. (The ^ symbol at the node `NP[t]-SBJ` in tree #1 indicates that it is a substitution node.) The extracted trees are the four trees numbered #1–#4. These trees are in effect the nodes in the derivation tree showing how the four elementary trees connect to each other.

We briefly mention three unusual features of this extraction, and refer the reader to (Kulick and Bies, 2009) for detail and justification.[3]

1. The function tags are included in the tree extraction, with the syntactic tags such as `SBJ` treated as a top feature value, and semantic tags such as `LOC` treated as a bottom feature value, extending the traditional TAG feature system (Vijay-Shanker and Joshi, 1988) to handle function tags.

2. Etree #2 consists of two anchors, rather than splitting up the tree decomposition further. This is because this is an instance of the idafa (”construct state”) construction in Arabic, in which two or more words are grouped tightly together.

3. During the extraction process, additional information is added to the nodes in some cases, as further attributes for the “top” and “bottom” information, parallel to the function tag information. In this case, the root of etree #2 has the “bottom” attribute `IDAFATOP`, meaning that it is the top of an idafa structure, and the lower `NP` has the “top” attribute `IDAFACOMP`, meaning that it is the complement within an idafa structure.[4] Such added attributes can be used by the search specifications, as will be done here.

The derivation tree for this tree decomposition

[1] We use the Buckwalter Arabic transliteration scheme `http://www.qamus.org/transliteration.htm` for the Arabic.

[2] We will sometimes use ”etree” as shorthand for ”elementary tree”.

[3] See (Habash and Rambow, 2004) for an earlier and different approach to extracting a TAG from the ATB. As they point out, there is no one correct way to extract a TAG.

[4] At the root node, the `IDAFATOP` information is in the “bottom” attribute because it is part of the structure from below. The `IDAFACOMP` is a “top” attribute because it is a consequence of being a child of the higher node.

shows how the relations of substitution, sister-adjunction, and Chomsky-adjunction relate the etree instances. For example, etree instance #2 substitutes at address 1.1.2[5] of etree instances #1, as indicated by the `S:1.1.2` above instance #2. Etree instance #3 Chomsky-adjoins at the root of etree instance #2, as indicated by the `M:1,r` above instance #3. The `M` indicates Chomksy-adjunction, the `1` indicates the root, and the `r` indicates that it is to the right.[6] Etree instance #4 sister-adjoins to node 1.1.2 in Etree #1, as indicated by the `A:1.1.2,r,1`, becoming a sister of node `NP[t]-SBJ^`. The `A` indicates sister-adjunction, and the `r` is again the direction, and the `1` indicates the ordering, in case there was more than one such sister-adjunction at a node.

It is of course often the case that the same elementary tree structure will be repeated in different elementary trees extracted from a corpus. To make our terminology precise, we call each such structure an "etree template", and a particular instance of that template, together with the "anchors" (tokens) used in that instance of the template, is called an "etree instance". For example, in tree #2, the template is `(NP[b]:IDAFATOP A1 (NP[t]:IDAFACOMP A2))`, where `A1` and `A2` stand for the anchors, and this particular etree instance has that template and the anchors `(NOUN TA}rp)` and `(NOUN tryb)`.

3 Query Processing

We are concerned here with showing the analysis of parallel sets of annotations on the same text, and as mentioned in Section 1, we compare gold and parser output. However, we are interested in exploring differences between different parser runs, in which aspects of the parser model are changed. Therefore we use a training/dev/test split[7], and work here with the dev section. We do not include here all the details of the parser setup, since that is not the focus here,[8] but we work with two settings. For both "Run 1" and "Run 2", the parser is supplied with the gold tags for each word. For "Run 2 ", the parser is forced to use the given tags for every word. For "Run 1", the parser can use its own tags, based on the training data, for words that it has seen in training. We are interested in exploring some of the consequences of this difference.

We therefore carry out the extraction procedure described in the previous section on each of three versions of trees for the same tokens: (a) the gold dev section trees, (b) the parser output for Run 1, and (c) the parser output for Run 2. Each has (the same) 17882 tokens. The gold version has 14370 etree instances using 611 etree templates, Run 1 has 14208 etree instances and 489 etree templates, and Run 2 has 14215 etree instances using 497 etree templates. This gives some 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 tree structure information is placed in a separate table.

A significant aspect of this decomposition of the parse output is that the tree decomposition relies upon the presence of function tags to help determine the argument status of nodes, and therefore what should be included in an elementary tree. We therefore use a modification of Bikel parser as described in (Gabbard et al., 2006), so that the output contains function tags. However, inaccuracy in the function tag recovery by the parser could certainly affect the formation of the elementary trees resulting from Runs 1 and 2. We do not include empty categories for the parser output, while they are present in the Gold trees.[9] There are 929 etree templates in total, combining those for the three versions, with those for Run 1 and Run 2 overlapping almost entirely.

The extracted tokens, etree templates, etree instances, and derivation trees are stored in a MySQL database for later search. The derivation tree is implemented with a simple "adjacency list" representation, as is often done in database representations of

[5] The addresses are Gorn addresses, with the root as 1.

[6] Since we do not store such directional information in the actual tree adjoining in in the traditional TAG way, by including the appropriate root and foot node, the directional information needs to be specified in the derivation tree.

[7] http://nlp.stanford.edu/software/parser-arabic-data-splits.shtml. We also include only sentences of length $<= 40$.

[8] We used the Bikel parser, available at `www.cis.upenn.edu/~dbikel/software.html`

[9] On a brief inspection, this is likely the reason for the greater number of templates used for the gold version of the data, since the templates then include the empty categories as well.

Lexical restrictions:

```
L1: text="Ean"
```

Etree queries:

```
E1) [NP{b:IDAFATOP}  (A1,)]
    [NP{t:IDAFACOMP} (A2,)]
E2) [NP{b:IDAFATOP}  (A1,)]
    [NP{t:IDAFAMID}  (A2,)]
    [NP{t:IDAFACOMP} (A2,)]
...
E6 [VP (A1,PP[t]-CLR^{dta:1})]
E7 [PP (A1{lex:L1},)]
```

Dtree queries:

```
# two-level idafa (NP A1 (NP A2))
D1) E1
# three-level idafa
# (NP A1 (NP A2 (NP A3)))
D2) E2
# four-level idafa
# (NP A1 (NP A2 (NP A3 (NP A4))))
D3) E3
# five-level idafa
D4) E4
# six-level idafa
D5) E5
# VP with PP substituting into PP-CLR
D6) (sub{dta:1} E6 E7)
```

Figure 3: Examples of Etree and Dtree queries

hierarchical structure. We do not have space here to show the database schema, but it is organized with appropriate indexing so that a full tree is represented by a derivation tree, with integers pointing to the etree instance, which in turn use integers to represent the etree template in that etree instance and also point to the anchors of that etree instance.

This tree extraction and database setup only needs to be done once, as a preliminary step, for all of the queries on the corpus, as stored in the database. We now illustrate how queries can be specified, and describe the algorithm used for searching on the database with the extracted tree grammar.

3.1 Query Specification

Queries are specified as "Etree queries" and "Dtree queries". Sample queries are shown in Figure 3. Etree queries determine a set of etree instances, by specifying conditions on the structure of a etree instance (and therefore on the etree template that the etree instance uses), and, optionally, lexical constraints on the anchor(s) of that etree instance. The Dtree queries specify a relationship in the derivation tree of etree instances that satisfy certain etree queries.

Each Etree query is in the form of a list of pairs, where each pair is `(node-label, children-of-node-label)`, where the node labels identify nodes on the spine from the root down. We forgo a rigorous definition here of the query language here in favor of focusing on the example queries.

Etee query E1 specifies that an etree instance is a match for E1 if it has a path with a `NP{b:IDAFATOP}` node and then another node `NP{t:IDAFACOMP}`. Each such node further has a child that is an anchor, `A1` for the first, and `A2` for the second. There are no lexical restrictions specified for these anchors, so any etree instance with an etree template that satisfies that condition satisfies E1. Etree query E2 is similar except that it matches a three-level idafa, using the attribute `IDAFAMID` to do so. By repeating the number of nodes in the spine with `IDAFAMID`, idafas of various sizes can be found, as in Etree queries E3-E5, which we leave out.

Etree query E6 specifies that an etree instance is a match for E6 if it has a VP node, with children `A1` and substitution node `PP-CLR^`. Etree query E7 simply finds all templates with node `PP`, for which the anchor satisfies lexical restrction `L1`, which is specified to mean that its text is `Ean`.

Some Dtree queries, such as `D1-D5`, are as simple as possible, corresponding to a single node in the derivation tree, and are identical to a specified Etree query. Here, `D1-D5` just return the results of Etree queries `E1-E5`, respectively. Other Dtree queries involve two nodes in the derivation, such as `D6`, which specifies that is is selecting pairs of Etree instances, one satisfying Etree query `E6`, and the other `E7`, with the latter substituting into the former. This substitution has to be at a certain location in the parent etree instance, and `dta:1` (for "derivation tree address') is this location. It arises from the search of etree templates for the parent query, here `E6`, in a manner described in the following Step 1.

3.2 Step 1: Etree Template Search

The etree templates are searched to determine which match a given etree query. For the current data, all

929 etree templates are searched to determine which match queries E1–E7. It's currently implemented with simple Python code for representing the templates as a small tree structure. While this search is done outside of the database representation, the resulting information on which templates match which queries is stored in the database.[10]

This step does not search for any lexical information, such as `lex:L1` in E7. That is because this step is simply searching the etree templates, not the etree instances, which are the objects that contain both the etree template and lexical anchor information. So this step is going through each template, without examining any anchors, to determine which have the appropriate structure to match a query. However, in order to prepare for the later steps of finding etree instances, 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, or that a particular address in that template will be used in a derivation tree search.

This additional information is not necessarily the same for different templates that otherwise match a query. For example, the two templates

```
(1) (S (VP A1 NP[t]-SBJ^ PP[t]-CLR^))
(2) (SBAR WHNP^
          (S (VP A1 (NP[t]-SBJ (-NONE- *T*))
                    PP[t]-CLR^)))
```

both match query E6, but for (1) the stored address `dta:1` is 1.1.3, while for (2) the stored address is 1.2.1.3, the address of `PP[t]-CLR^` in each template. Likewise, the stored information specifies that an etree instance with the template `(PP A1 NP^)` matches the query E7 if the anchor has the text `Ean`.

This step in effect produces specialized information for the given template as to what additional restrictions apply for that (query,template) pair to succeed as a match, in each etree instance that uses that etree template.

To summarize, this step finds all (Etree query, etree *template*) matches, and for each case stores the additional lexical restriction or `dta` information for that pair. This information is then used in the following steps to find the etree *instances* that match a given Etree query Eq, by also checking the lexical restriction, if any, for an etree instance that has a template that is in the pair (Eq, template), and by using in a derivation tree search the `dta` information for that pair.

3.3 Step 2: Dtree Search and Etree Instances

For each Dtree query, it first finds all etree instances that satisfy the etree query (call it Eroot here) contained in the root of the Dtree query. This is a two-part process, by which it first finds etree instances such that the (Eroot, etree template) is a match for the instance's etree template, which is the information found in Step 1. 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.

For single-node Dtree queries, such as D1−D5 this is the end of the processing. For two-node Dtree queries, such as D6, it descends down the derivation tree. This is similar to the two-part process just described, although the first step is more complex. For the Etree query specified by the child node (call it Echild here), it finds all etree instances such that (Echild, etree template) is a match for the instance's etree template, and, in addition, that the etree instance is a child in the derivation tree for a parent that was found to satisfy Eroot, and that the address in the derivation tree is the same as the address `dta` that was identified during Step 1 for the template of the parent etree instance. Note that the address is located on the parent tree during Step 1, but appears in the derivation tree on the child node.

4 Search over Pairs of Trees

As discussed in the introduction, one of the motivations for this work is to more easily compare two sets of trees for structures of interest, arising from either two annotators or gold and parser output. We construct confusion matrices showing how corresponding tokens across two different annotations compare with regard to satisfaction of the queries of interest. We do this by associating each token with satisfaction results for queries based on the etree instance that the tree token belongs to (this is related to the

[10] While there are several ways to optimize this tree matching, we have not made that a priority since the search space is so small.

Gold:

```
(PP-PRP (PREP <17>li)                                          for/to
        (NP (NOUN+CASE_DEF_GEN <18>waDoE+i)                    laying down
            (NP
                (NP (NOUN+CASE_INDEF_GEN  <19><iTAr+K)         framework
                    (ADJ+CASE_INDEF_GEN   <20>EAm~+K)          general
                (SBAR ....
```

Run1:

```
(PP-PRP (PREP <17>li)
        (NP
            (NP (NOUN <18>waDoE+i)
                (NP (NOUN <19><iTAr+K)
                    (NP (NOUN <20>EAm~+K)
            (SBAR
                 ...
```

Run2:

```
(PP-PRP (PREP <17>li)
          (NP
              (NP (NOUN <18>waDoE+i)
                  (NP
                      (NP (NOUN <19><iTAr+K)
                          (ADJ <20>EAm~+K)
                      (SBAR
```

Figure 4: Token <18> is an example entry from cell (D1, D2) in Table 1, showing that the gold tree satisfies query D1 (a two-level idafa) while the Run 1 parse tree satisfies, incorrectly, query D2 (a three-level idafa). Token <18> in Run 2 correctly satisfies query D2.

"lexicalization" property of TAG). To prevent the same query from being counted twice, in case it is satisfied by an etree instance with more than one anchor, we associate just one "distinguished anchor" as the token that counts as the satisfying that instance of the query.[11] Similarly, for a Dtree query such as D6 that is satisfied by two etree instances together, each one of which would have its own distinguished anchor, we use just the anchor for the child etree instance. For D6, this means that the token that is associated with the satisfaction of the query is the preposition in the child elementary tree.

As discussed in Section 3, we have three sets of trees to compare over the same data, (a) the gold, (b) Run 1, and (c) Run 2. We constructed confusion matrices measuring (a) against (b), (a) against (c), and (b) and against (c). The latter is particularly helpful of course when identifying differences between the two parser runs. However, due to space reasons we only present here sample confusion matrices for (a) the gold vs. (b) Run 1, although our examples also show the corresponding tree from Run 2.

It is often the case that some queries are logically grouped together in separate confusion matrices. For the queries in Figure 3, we are interested in comparing the idafa queries (D1−D5) against each other, with the PP−CLR case (D6) in isolation.

gld\Rn1	N	D1	D2	D3	D4	D5	Total
N	0	66	30	6	0	0	102
D1	81	1389	13	3	1	0	1487
D2	21	4	285	1	1	0	312
D3	1	0	1	42	0	0	44
D4	0	0	0	0	5	0	5
D5	0	0	0	0	0	1	1
Total	103	1459	329	52	7	1	1951

Table 1: Confusion matrix showing results of queries D1-D5 for Gold trees and Run 1

Table 1 shows the confusion matrix for queries D1−D5 for the gold vs. Run 1. The row N contains cases in which the token for the gold tree did not satisfy any of query D1−D5, and likewise the column N contains cases in which the token for the parse output did not satisfy any of queries D1−D5. The cell (N,N) would consist of all tokens which do not satisfy any of queries D1−D5 for either the gold or the parse, and so are irrelevant and not included.

For example, the cell (1, 2) consists of cases in which the token in the gold tree is a distinguished anchor for an elementary tree that satifies query D1, while the corresponding token in the parse output is a distinguished anchor for an elementary tree that satisfies query D2. An example of an entry from this cell is shown in Figure 4. The token

[11] This is just the anchor that is the head.

Gold:

```
(S
   (VP (PV+PVSUFF_SUBJ:3MS  <13>Eab˜ar+a)                    express + he
       (NP-SBJ (DET+NOUN+CASE_DEF_NOM  <14>Al+|bA'+u)         the fathers/ancestors
       (PP-CLR (PREP <15>Ean)                                 from/about/of
               (NP
                   (NP (NOUN+CASE_DEF_GEN  <16>qalaq+i)       unrest/concern/apprehension
                       (NP (POSS_PRON_3MP  <17>him)           their
                   (PP (PREP <18>min)                         from
                       (NP ...
```

Run1:

```
(S
  (VP (PV <13>Eab˜ar+a)
      (NP-SBJ
           (NP
               (NP (DET+NOUN <14>Al+|bA'+u)
               (PP (PREP <15>Ean)
                   (NP
                       (NP (NOUN <16>qalaq+
                           (NP (POSS_PRON <17>him)
                       (PP (PREP <18>min)
                           (NP ...)
```

Run2:

```
(S
  (VP (PV <13>Eab˜ar+a)
      (NP-SBJ (DET+NOUN <14>Al+|bA'+u)
      (PP-CLR (PREP <15>Ean)
              (NP (NOUN <16>qalaq+i)
                  (NP (POSS_PRON <17>him)
      (PP (PREP <18>min)
           (NP ....)
```

Figure 5: Token <15> is an example entry from cell (D6,N) in Table 2, showing that the gold tree satisfies query D6 (a verbal structure with a PP−CLR argument that is headed by Ean), while the Run 1 parse tree fails to satisfy this. Token <15> in the Run 2 parse does correctly satisfy query D6.

<18>waDoE+i satisfies query D1 in the gold tree because it is the distinguished anchor for the two-level idafa structure consisting of tokens <18> and <19>:

```
(NP (NOUN+CASE_DEF_GEN <18>waDoE+i)
    (NP (NOUN+CASE_INDEF_GEN <19><iTAr+K)))
```

The modifier at <20> does not interfere with this identification of the two-level idafa structure, since it is a modifier and therefore a separate etree instance in the derivation tree.

However, token <18> in the Run 1 output is the distinguished anchor for a 3-level idafa, consisting of the tokens at <18>, <19>, <20>. Note that these identifications are made separately from the incorrect attachment level of the SBAR (at <19> in the gold tree, at <18> in Run 1), which is a separate issue from the idafa complexity, which is of concern in this query. One can see here the effect of the parser choosing the wrong tag for token <20>, a NOUN instead of ADJ, which causes it to mistakenly build an extra level for the idafa structure. Figure 4 also shows the corresponding part of the parse output for Run 2 (in which the parser is forced to use the given tags), which is correct. Therefore if we also showed the confusion matrix for gold/Run 2, token <18> would be an entry in cell (D1,D1). Also, in a confusion matrix for Run1/Run2 it would appear in cell (D2,D1). (This is analogous to comparing annotations by two different anotators.)

gold\Run 1	N	D6	Total
N	0	152	152
D6	76	258	334
Total	76	410	486

Table 2: Confusion matrix showing results of query D6 for Gold trees and Run 1

Table 2 shows the confusion matrix for Dtree query D6, which simply scores the satisfaction of D6 compared with a lack of satisfaction. An example entry from cell (D6,N) is shown in Figure 5, in which token <15>Ean in the gold tree is the distinguished anchor for the child etree instance that satisfies query D6, while the corresponding token <15>Ean in Run 1 does not. For Run 2, token <15> does satisfy query D6, although the attachment of the PP headed by <18>min is incorrect, a separate issue.

5 Future Work

Our immediate concern for future work is to use the approach described for inter-annotator agreement and work closely with the ATB team to ensure that the queries necessary for interannotator comparisons can be constructed in this framework and integrated into the quality-control process. 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 also aim to provide information on where two annotators agree on the core structure, but disagree on attachment of modifiers to that structure, a major problem for corpus annotation consistency.

However, there are many topics that need to be explored within this approach. We conclude by mentioning two.

(1) We are not using classic TAG adjunction, and thus cannot handle any truly (i.e., not auxiliaries) long-distance dependencies. Related, we are not properly handling coindexation in our extraction. The consequences of this need to be explored, with particular attention in this context to extraction from within an idafa construction, which is similar to the extraction-from-NP problem for TAG in English.

(2) We are also particularly interested in the relation between query speed and locality on the derivation tree. In general, while searching for etree instances is very efficient, complex searches over the derivation tree will be less so. However, our hope, and expectation, is that the 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. We plan to evaluate the speed of this system, in comparison to systems such as (Ghodke and Bird, 2008) and Corpus Search[12].

Acknowledgements

We thank David Graff, 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.

[12] `http://corpussearch.sourceforge.net`.

References

John Chen. 2001. *Towards Efficient Statistical Parsing Using Lexicalized Grammatical Information*. Ph.D. thesis, University of Delaware.

David Chiang. 2003. Statistical parsing with an automatically extracted tree adjoining gramar. In *Data Oriented Parsing*. CSLI. http://www.isi.edu/~chiang/papers/chiang-dop.pdf.

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

Ryan Gabbard, Seth Kulick, and Mitchell Marcus. 2006. Fully parsing the Penn Treebank. In *HLT-NAACL*, pages 184–191.

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

Nizar Habash and Owen Rambow. 2004. Extracting a Tree Adjoining Grammar from the Penn Arabic Treebank. In *Traitement Automatique du Langage Naturel (TALN-04)*, Fez, Morocco.

A.K. Joshi and Y. Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*, pages 69–124. Springer, New York.

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

Seth Kulick and Ann Bies. 2010. A treebank query system based on an extracted tree grammar. In *HLT-NAACL (short paper)*.

Seth Kulick, Ann Bies, and Mohammed Maamouri. 2010. A quantitative analysis of syntactic constructions in the Arabic Treebank. Presentation at GURT 2010 (Georgetown University Roundtable).

Mohamed Maamouri, Ann Bies, Seth Kulick, Sondos Krouna, Fatma Gaddeche, and Wajdi Zaghouani. 2010. Arabic treebank part 3 - v3.2. Linguistic Data Consortium LDC2010T08, April.

K. Vijay-Shanker and A. K. Joshi. 1988. Feature structures based tree adjoining grammars. In *COLING*.

Fei Xia. 2001. *Automatic Grammar Generation From Two Different Perspectives*. Ph.D. thesis, University of Pennsylvania.