Treebank Analysis and Search Using an Extracted Tree Grammar

Seth Kulick and Ann Bies

Linguistic Data Consortium
University of Pennsylvania
{skulick,bies}@ldc.upenn.edu*

November 13, 2009

Abstract

We describe here a new approach to the problem of analyzing and comparing two sets of trees that contain annotation for the same data. This is an important problem both for evaluating inter-annotator consistency during treebank construction and also for evaluating parser output as compared to the gold trees. Our approach is based on a decomposition of the trees into small syntactic chunks, inspired by work in Tree Adjoining Grammar. This allows queries to be stated in more meaningful syntactic units, and the resulting system produces confusion matrices showing disagreements on these units across the two sets of trees. There is also a significant potential speed advantage for treebank search, since duplicate information is removed from the treebank, and a search for the syntactic chunks simply becomes a search on integers associated with each sentence.

1 Introduction

A crucial issue when constructing a treebank is to ensure consistency in annotation decisions among the annotators. This is usually done by having multiple annotators annotate the same file, and then comparing the different annotations of that file. One way in which this has been done is to use the same metrics as used for scoring parser output against gold trees. This makes sense, since the two problems can

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

both be viewed as the same, as comparing two sets of trees to determine how well one matches against the other.

For example, the recent revision of the Penn Arabic Treebank [11] reported inter-annotator agreement (IAA) using the `evalb` program.[1] `evalb` determines a single score for the two sets of trees, by comparing matching constituent brackets across the two sets of trees.[2] However, while the `evalb` score can be viewed as a very rough approximation to the consistency of annotation, it is of little use in the important task of pinpointing where the annotators are disagreeing. To some extent this can be improved by breaking down the bracketing scores by bracket type (i.e., an `NP` score, an `SBAR` score, etc.). However, this is still just an approximation to the sorts of decisions actually made during annotation. One other approach that has been taken is to break down the two sets of trees into single-level head-dependency relations, as done for English in [5] and for Arabic in [10, 11].

There are of course systems available for searching a treebank corpus for various structures - e.g., [7, 12, 13]. These are extremely useful and heavily used (among other purposes) in corpus construction for identifying illegal structures that were mistakenly annotated. However, they also do not allow the comparison of structures across two sets of trees. Also, searches are stated in terms of individual nodes in the phrase-structure trees. While this allows great flexibility in searching for structures, it requires a certain amount of trickery to translate a search for meaningful syntactic units into the combination of single nodes.

Currently the analysis of disagreements among annotators in IAA annotation can only be done satisfactorily by painstaking and time-consuming manual comparison of the sets of trees, if the goal is to understand the underlying annotation decisions that led to the reported errors.

We present here an approach to a system for analyzing and evaluating two sets of trees in such a way as to allow search explicitly on these structures of interest, returning an analysis of differences on these structures. We break the trees down into meaningful syntactic units and search and evaluate based on these units, therefore returning information which is more aligned with the decisions made during annotation or mimicked during parsing. This approach is rooted in the long line of research on Tree Adjoining Grammar [9] which has been heavily used for various purposes in NLP but not previously for the purposes described here.

In Section 2 we discuss the particular data set we are using, which provides the examples of tree extraction and query search in the following sections. Section 3 describes the process of decomposing the full trees into the core syntactic units. This is similar to previous work in this tradition, although with some novel aspects, such as including the function tags in the extracted trees. In Section 4 we discuss how search on a treebank is reconceptualized and implemented to work on the tree decomposition of the full trees. In Section 5 we run through some examples of queries and results, and demonstrate the utility of this approach. Section 6 is the

[1] `http://nlp.cs.nyu.edu/evalb/`

[2] It also produces a few secondary numbers, such as the number of crossing brackets.

conclusion and discusses several possibilities for future work.

2 Data Set

There are several corpora under construction which are appropriate candidates for utilizing the work described here. We focus in this paper on the Penn Arabic Treebank, which has recently undergone a substantial revision [11] in its guidelines and for which more data is being annotated. Further, while there have been improvements in Arabic parsing ([10, 11]), there are still many questions as to which aspects of the tree structure are accurately recovered by the parser and which remain problematic.

Given the choice between working with a limited number of IAA files, and the more substantial number of trees available from parsing work, we have chosen to work first with the parse files. This is mostly because of the greater number of samples available for the development of this approach. As discussed above, it is basically the same problem regardless of which pairs of trees we work with.

The corpus we are using consists of the recent Arabic Treebank revisions of parts ATB1, ATB2, and ATB3.[3] For the parsing work, we used a previously-proposed train/dev/test split.[4]

The problem of segmentation of Arabic text into the tokens that match the gold tokens is not a simple one, and previously published work on parsing Arabic (e.g., [10, 11]), simply assumes gold tokenization for input to the parser. We instead use a morphological tagger and tokenizer [15], and for parsing use the Bikel parser.[5] [6]

3 Elementary Tree Extraction

As discussed above, we are aiming for an analysis of the trees that is directly expressed in terms of the syntactic constructions that annotators have in mind during annotation, or are mimicked during parsing. Towards this end we utilize ideas

[3] LDC2008E61 (Arabic Treebank Part 1 v4.0), LDC2008E62 (Arabic Treebank Part 2 v3.0), and LDC2008E22 (Arabic Treebank Part 3 v3.1), respectively.

[4] `http://nlp.stanford.edu/software/parser-arabic-data-splits.shtml`

[5] `http://www.cis.upenn.edu/~dbikel/software.html`

[6] There is an important issue here for parser evaluation that we can only discuss briefly. The `evalb` program depends on the two sets of trees having exactly matching tokenizations, so that the corresponding constituent spans can be compared. The spans cannot be compared when the tokenizations do not match. In fact, while the tagger has an overall tokenization accuracy of about 98.5%, even this level of accuracy is catastrophic for `evalb`, since out of 1739 sentences in the dev section (of length $<= 40$), 338 differ in tokenization and so cannot be evaluated, and for this reason we do not present a parsing score. See [17] for a discussion of a similar problem for Chinese parsing evaluation. A good candidate for overcoming this problem is the Sparseval software [14], which allows the separate specification of how tokens are aligned. This is somewhat orthogonal to the main concerns of this paper, but we note here that this issue does not prevent the evaluation and analysis under our system. See footnote 13 for more detail on this issue.

```
==============================
TOKENS
==============================
<0> wa  {and}                              <1> jar+at {occur+it/they/she]}
<2> Al+masiyr+ap {the+march+[fem.sg.]}     <3> Al+>uxoraY {the+another}
<4> fiy {in}                               <5> muxay~am {refugee camp}
<6> jabAlyiA {Jabaliya}                    <7> li {for/to}
<8> lAji}+iyna {refugee+[masc.pl.gen.]}    <9> $amAl {north}
<10> gaz~+ap {Gaza+[fem.sg.]}              <11> bi {by/with}
<12> mu$Arak+ap {participation+[fem.sg.]}  <13> Ea$ar+At {scores+[fem.pl.]}
<14> Al+>aTofAl {the+children}

================================================================
FULL TREE
================================================================

(S (CONJ <0>wa)
   (VP (PV+PVSUFF_SUBJ:3FS <1>jar+at)
       (NP-SBJ (DET+NOUN+NSUFF_FEM_SG <2>Al+masiyr+ap)
               (DET+ADJ <3>Al+>uxoraY))
       (PP-LOC (PREP <4>fiy)
               (NP
                   (NP (NOUN <5>muxay~am})
                       (NP (NOUN_PROP <6>jabAliyA)))
                   (PP (PREP <7>li)
                       (NP (NOUN+NSUFF_MASC_PL_GEN <8>lAji}+iyna)))
                   (NP-LOC (NOUN <9>$amAl)
                          (NP (NOUN_PROP+NSUFF_FEM_SG <10>gaz~+ap)))))
       (PP-MNR (PREP <11>bi)
               (NP (NOUN+NSUFF_FEM_SG <12>mu$Arak+ap)
                   (NP (NOUN_NUM+NSUFF_FEM_PL <13>Ea$ar+At)
                       (NP (DET+NOUN <14>Al+>aTofAl)))))))

================================================================
EXTRACTED ETREE INSTANCES
================================================================
#     ETREE TEMPLATE                  ANCHORS
1     A1                              (A1) <0> CONJ wa
2     (S (VP A1 NP[t]-SBJ^))          (A1) <1> PV+PVSUFF_SUBJ:3FS jar+at
3     (NP A1)                         (A1) <2> DET+NOUN+NSUFF_FEM_SG Al+masiyr+ap
4     A1                              (A1) <3> DET+ADJ Al+>uxoraY
5     (PP[b]-LOC A1 NP^)              (A1) <4> PREP fiy
6     (NP A1 (NP A2))                 (A1) <5> NOUN muxay~am
                                      (A2) <6> NOUN_PROP jabAliyA
7     (PP A1 NP^)                     (A1) <7> PREP li
8     (NP A1)                         (A1) <8> NOUN+NSUFF_MASC_PL_GEN lAji}+iyna
9     (NP[b]-LOC A1 (NP A2))          (A1) <9> NOUN $amAl
                                      (A2) <10> NOUN_PROP+NSUFF_FEM_SG gaz~+ap
10    (PP[b]-MNR A1 NP^)              (A1) <11> PREP bi
11    (NP A1 (NP A2 (NP A3)))         (A1) <12> NOUN+NSUFF_FEM_SG mu$Arak+ap
                                      (A2) <13> NOUN_NUM+NSUFF_FEM_PL Ea$ar+At
                                      (A3) <14> DET+NOUN Al+>aTofAl
```

Figure 1: An example of Tree Decomposition resulting in extracted elementary trees (etrees) (Translation of Arabic sentence: Another march occurred in the north Gaza Jabaliya refugee camp for refugees with the participation of scores of children.)

from a line of research on decomposing full trees in a treebank into smaller syntactic chunks. Usually based around Tree Adjoining Grammar (TAG) or some variant (loosely referred to as "tree grammars"), this work 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 [3, 4, 16]. However, this approach has not been utilized for searching within a treebank, as far as we know.

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. For our grammar we use a TAG variant with tree-substitution, sister-adjunction, and Chomsky-adjunction ([4]). We do not have space here to review these basic aspects of the extraction in detail, but instead we give an example in Figure 1, and highlight some aspects of our tree extraction that we feel are worthy of note.[7]

The full tree is shown in the middle of Figure 1.[8] Each token is listed as `(POS <index> word)`, where `<index>` is the index of the word in the sentence. To avoid cluttering up the tree structure, we include at the top a separate listing of the glosses for each word.

The extracted elementary trees are shown at the bottom. Each decomposed tree has a particular tree structure, and it is possible (indeed, it is the entire reason for this approach) that the same tree structure is used in more than one decomposed tree fragment. We call each such elementary tree structure an "etree template", and a particular instance of that template, together with the "anchors" (tokens) used in that instance of that template, is called an "etree instance" ("etree" is short for "elementary tree").

For example, the template `(S (VP A1 NP[t]-SBJ^))` is used once in this tree decomposition (although many times in the entire corpus), where `A1` is the anchor of the template, which is associated with a particular word in an etree instance that uses this template. In this case, instance #2 uses this template, and the anchor is the verb at index `<1>`. The `^` indicates that the `NP[t]-SBJ^` node is a substitution node, meaning that another etree instance substitutes into it to reform the original full tree (in this case, etree instance #3).

The template `(NP A1)` is used in two etree instances, #3 with anchor `<2>` and #8 with anchor `<8>`. Templates can have more than one anchor. For example, the template `(NP[b]-LOC A1 (NP A2))` is an example of a two-level *idafa* structure, an extremely common structure in Arabic, in which two or more tokens form a tight syntactic unit. (In this particular case, the entire structure projects with a `LOC` function tag as well.) Etree instance #9 uses this template, with two anchors, the words at indices `<9>` and `<10>`. Instance #11 is an example of a three-level idafa.

A fundamental idea of this approach (as in all TAG-related work) is that the modifiers are separated from non-recursive structures. For example, the two-level

[7] See [8] for an earlier and somewhat different approach to extracting a tree grammar from the Arabic Treebank.

[8] Throughout this paper we use the Buckwalter Arabic transliteration scheme [2].

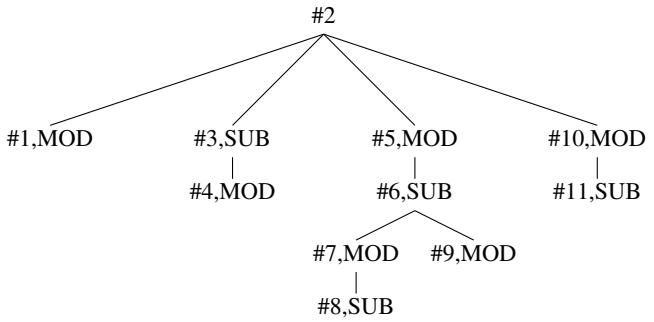

Figure 2: The derivation tree for the extraction in Figure 1

idafa at indices <5>,<6> is separated from its modifier PP and NP-LOC sisters.[9] Some modifiers result in etree templates that are just a single anchor, with no structure. For example, the template A1 is used for instance #1, of just the typical sentence-initial conjunction wa, and also for the adjective <3>, which is separated out from the noun it modifies (<2>).

It is important to note that each etree instance has at least one anchor, and every tree token is an anchor for some etree instance. It therefore becomes possible to examine the properties of an etree instance that a tree token is the anchor for, and to compare corresponding instances for the same tree token across two different annotations, such as gold/parser-output trees or between the trees for two different annotators. This property (traditionally called "lexicalization" in the Tree Adjoining Grammar literature) is taken advantage of for query searching and comparison in Sections 4 and 5.

As usual in tree grammar extraction, the extraction process for each full tree produces not just the collection of etree instances, but also a "derivation tree" that is in effect a record of how the etree instances combine together again to form the original full tree. The derivation tree for the extraction just described is shown in Figure 2, in which the nodes of the tree correspond to the etree instances in Figure 1, and the SUB and MOD indicate the type of attachment (substitution and modification).[10] For example, the derivation tree shows etree instance #4 modifying instance #3, which in turn substitutes into instance #2.

Unlike all earlier work in tree decomposition that we know of, we also include function tags in the extracted trees. While a prominent feature of the Penn Treebank and the Penn Arabic Treebank, they have been mostly ignored in parsing (with some exceptions - e.g., [1, 6]) and in previous tree extraction work.[11]

[9] The "extra" NP that scopes around the span <5,10> in the full original tree is therefore missing from the extracted instances. If recreating the original tree from the extracted trees, it can be added back in to exactly recreate the original tree.

[10] This derivation tree is a bit of a simplification. The complete tree also includes the addresses in each etree template of the locus of substitution or modification, and for sister adjunction, the direction of modification. We leave out these details here.

[11] Except for the use of the function tags as a way to determine the argument/adjunct classification

Since the function tags are an important part of the annotation (see the arguments in [1, 6]), and since they are part of the annotation process (and so something to be checked in the inter-annotator agreement files), we include them in the tree decomposition. We break the tags up into two groups "syntactic" and "semantic". The TAG decomposition views the former as being imposed on a node from the top, and the latter as arising from the bottom, and this controls where the function tag for a node from the original tree is placed in the extracted trees. For example, the `SBJ` function tag (a syntactic tag) in the full tree is placed in the template for instance #2 (with the `[t]-SBJ` meaning that it is a "top" function tag), and it is *not* present on the template for instance #3, and so the word at index `<2>` in etree instance #3 only receives that label from substituting into the `NP[t]-SBJ^` node. In contrast, the `-LOC` and `-MNR` tags are semantic tags, and so "bottom" tags and appear in the extracted trees anchored by the head of the constituent the tag appears with in the original tree.

4 Treebank Search Using Elementary Trees

We carry out the extraction procedure described in the previous section to both versions, gold and parsed, of the dev section, using the data discussed in Section 2.[12]

For the two versions of the dev section, taken together, there are 141499 tokens, with 83196 etree instances, which need only 1551 etree templates. After this extraction process is completed, all of the tokens, etree templates, etree instances, and derivation trees are stored in a MySQL database for later search. We do not have space here to show the database schema, but it is organized with appropriate indexing to allow for very quick access to the etree instances for each sentence, and to the etree template each one in turn links to.

The search for particular syntactic structures, as represented by the etree templates, therefore becomes a search for indexed integers (with integers representing etree instances and etree templates). Searching for these elementary trees is therefore extremely fast, something we are taking advantage of for separate work on searching an entire corpus for syntactic structures, aside from the current issue of searching on two sets of trees together. From the perspective of database organization, the tree extraction can be perhaps be viewed as a type of database "normalization", in which duplicate information is placed in a separate table.

Before any queries are processed, the tree decomposition and database creation, as discussed above, is done. This is only done once, for the two full sets of

in various head-based parsing approaches, or in the earlier TAG-based tree extraction work. However, most parsing work has not been concerned with function tag recovery during parsing, and no TAG-based extraction work has included the function tags in the extracted trees.

[12]The extraction of the parse output requires that function tags be included in the output, as in either [1, 6]. This is because the function tags are used for the tree decomposition process. However, it does not currently use empty categories in the parse output, as in [6] and other work, although that will be incorporated in future work.

trees, and is used for all future query searches.

Currently queries are specified as conditions over elementary trees. We do not currently utilize the derivation tree in the search as well, to query on how etrees connect with each other. However, this is very important for future work, as discussed in Section 6.

Further, the queries can be logically grouped into sets, for cases in which we want to see how particular etree instances might satisfy one query instead of another. For example, we might want to see if the parser, or the annotators, are confusing `LOC` and `ADV` function tags, or treating a particular token as part of a three-level idafa instead of a two-level idafa.

We take advantage of the "lexicalized" property of the tree grammar, as discussed in Section 3, to allow us to construct confusion matrices showing how corresponding tokens across two different annotations compare with regard to satisfaction of the queries of interest. This is possible because we can associate each token with satisfaction results for various queries based on the etree instance that the tree token belongs to. Even queries examined on their own, without reference to other queries, form a 2x2 confusion matrix, in which the query is compared against the lack of satisfaction of that query. We show examples of these query results and confusion matrices in the next section.

For a given set of queries, the following sequence occurs:

- The etree templates are searched to determine which match a given query. It is possible (and likely) that a query will select more than one of the 1551 etree templates. For example, a query specifying the structure (SBAR WHNP S) will find cases of such SBAR trees regardless of what is below the S node (intransitive or transitive verb, verbal noun, etc.).

- The etree instances are then searched to determine which take a template that satisfies a given query.

- These first two steps result in the following situation. Each etree instance is categorized as satisfying a given query or not. Due to the "lexicalized" property of the tree grammar, as just discussed, each treebank token is linked to an etree instance, and so it immediately follows that for any treebank token, it is easy to determine whether it is in a syntactic context that satisfies a given query.

- The two sets of trees are gone through in parallel, token by token.[13] Each corresponding token in the two sets of trees is checked for which queries it satisfies. The query results are stored in confusion matrices as mentioned above.

[13] As noted in footnote 6, there is an issue here with differing tokenizations in the gold and parser data. More precisely, we are gathering data on the original whitespace-delimited tokens, which themselves are broken up into multiple tokens for the trees, perhaps differently for the gold and parsed data. This allows the matching of tokens across the two sets of trees, thus overcoming the problems with `evalb` mentioned in footnote 6. We do not show in this paper the two levels of tokens.

gold\ parsed	N	query 4	total
N		194	194
query 4	88	677	765
	88	871	959

Table 1: Confusion matrix showing results of query 4. `N` indicates absence of satisfaction of query 4. The cell `(N,N)` is not included since we are only interested in cases in which at least one query was satisfied.

gold\ parsed	N	1	2	3	total
N		6	0	131	137
query 1	16	28	0	8	52
query 2	3	0	1	1	5
query 3	215	2	0	313	530
total	234	36	1	453	724

Table 2: Confusion matrix showing results of queries 1, 2, and 3.

5 Data Set and Example Queries

We give four examples of searches that we are currently using. They are specified as partial tree information, and are matched against the etree templates. The `A$` indicates the anchor of an elementary tree that is associated with that query.[14]

1 - `(NP[b]-LOC A$)` - the anchor projects to a NP-LOC

2 - `(NP[b]-DIR A$)` - the anchor projects to a NP-DIR

3 - `(NP[b]-ADV A$)` - the anchor projects to a NP-ADV

4 - `(NP A (NP A (NP A$)))` - a three-level idafa

In addition, we consider query 4 by itself, resulting in a 2x2 confusion matrix for cases in which a three-level idafa was found vs. the absence of a three-level idafa. Queries 1, 2, and 3 are grouped together in a confusion matrix, along with absence of satisfaction of query 1, 2, or 3.

Table 1 shows the confusion matrix for query 4, the three-level idafa. The `N` row indicates the absence of query 4 in the gold tree, while the `N` column indicates the absence of query 4 in the parsed tree. We do not include the cell `(N,N)` because those would be cases in which the query under consideration is missing from both the gold and parsed trees for some token, and so are not relevant for this table.

The cell `(4,4)` indicates that there are 677 cases where both sets of trees agreed on the same three-level idafa. However, there are 88 cases where the gold tree had a three level idafa and the parsed tree did not, and 194 cases of the inverse. Figure 3 shows an example of an entry from cell `(N,4)` in Table 1, in which the token `<10>` in the parsed tree is the anchor of a three-level idafa, thus satisfying query 4, while the corresponding anchor in the gold tree does not. Note that it is a simple

[14] i.e., if an elementary tree has more than one anchor, we want only one anchor (word) to trigger that query, so that the elementary tree is not counted twice.

```
        GOLD TREE                                          PARSED TREE
        =========                                          ===========
(PP (PREP                                                   (PP (PREP
     <7>li,for/to)                                              <7>li)
    (NP (NOUN+NSUFF_MASC_PL_GEN                                 (NP (NOUN
         <8>lAji}+iyna,refugee+[masc.pl.gen.])                      <8>lAji}+iyna)
(NP-LOC (NOUN+CASE_DEF_ACC                                         (NP (NOUN
         <9>$amAl+a,north/North+[def.acc.])                             <9>$amAl)
        (NP (NOUN_PROP+NSUFF_FEM_SG+CASE_DEF_GEN                       (NP (NOUN_PROP
             <10>gaz~+ap+i,Gaza+[fem.sg.]+[def.gen.])                         <10>gaz~+ap)
```

Figure 3: The token <10>gaz~+ap in the parsed tree is an example of a token that results in the cell (N,4) in Table 1, since it is the anchor of a three-level idafa while the corresponding token in the gold tree is not. At the same time, the token <9> in the gold tree results in an entry in cell (1,N) in Table 2 since it is the anchor of a NP-LOC structure, while the corresponding token <9> in the parsed tree is not.

matter in this approach to report on multi-level etree structures that are larger than just one-level relations in the trees.

Table 2 shows the confusion matrix for queries 1, 2, and 3. For example, the cell (1,1) indicates that there are 28 cases in which both the gold and parsed trees had an (NP-LOC A) structure for a token A. Cell (1,3) indicates that there are 8 cases in which the gold tree satisfied query 1 (NP-LOC) for some token, while the parser output tree satisfied query 3 (NP-ADV) for that same token.

Of particular interest to us is that cell (1,N) indicates that there are 16 cases in which the gold tree had a token projecting to NP-LOC while the parsed tree did not project to LOC, DIR, or ADV. One such case is the same pair of tree fragments in Figure 3, in which token <9> in the gold tree heads an NP-LOC while this is not the case in the parsed tree.[15]

These examples are of necessity just a sampling of the searches currently being used. The ability to define conditions on etrees and relate the results to particular tokens makes it easy to look for properties across the pairs of trees. It is extremely useful to be able to define queries on meaningful syntactic units, such as the idafas or, in other queries not shown here, properties of relative clause constructions, properties of etrees headed by verbs (such as valency), and so on. These types of searches are harder to do in the earlier work on tree analysis discussed in Section 1, which are limited to reporting on one-level head/dependency relations.

[15]This case is also of interest also because it shows that the parser is not taking advantage of morphological evidence showing that the three-level idafa analysis is wrong. The tight coupling of words in an idafa triggers various morpological and phonological effects, one of which is that the "n" in non-final words in the idafa is dropped, and so the suffix "iyna" in token <8> clearly indicates that the idafa in the parse output must be incorrect.

6 Conclusions and Future Work

We have described a new approach to treebank search that allows queries to be directly stated, and reports generated, using meaningful units of tree fragments. Future work will take place in three overlapping directions:

(1) The work has so far focused on Arabic parsed data. This needs to be extended for IAA analysis (which, again, is basically the same problem), and furthermore, for other languages, in particular English, both for parsing analysis and for current treebank construction.

(2) As discussed in Section 4, there are some notable potential speed advantages to this approach, and we intend to utilize and test this for the more standard problem of searching on a complete corpus for various configurations, in addition to the search on parallel trees as described here.

(3) However, by far the most important aspect of future work is extending the query definition to allow for searches on how etrees combine in the derivation tree. This will allow us to quantify various aspects of modifier attachment, always a concern for both annotation and parsing. Because we are using trees of meaningful syntactic structures as the basic units of search, we will be able to quantify the results concerning such questions as "how often do the annotators agree on the core structures, but disagree on the attachment of various modifiers into those structures?"

While searching in the derivation tree is necessary for fully expressing the searches we are interested in, it will add a layer of complexity to the search procedure. The worst-case scenario is that requiring arbitrary searches over the derivation tree will bring up here the usual issues concerning how to search trees in a database that we have so far avoided in this approach by "normalizing" the corpus by extracting out the etrees. However, we suspect that many of the required searches will need to examine only very local slices of the derivation tree, namely just parent/head/sister relations, which are those used to represent modification in the derivation tree. This should be the case because in the derivation tree, each node already represents a chunk of syntactic structure, instead of just a a single node. If this hypothesis holds, and arbitrary hierarchical searching on trees is avoided for queries, we will be able to search for queries we are interested in, while avoiding any significant speed penalty.

References

[1] Don Blaheta. *Function Tagging*. PhD thesis, Brown, 2003.

[2] Tim Buckwalter. Arabic morphological analyzer version 2.0. LDC2004L02, 2004. Linguistic Data Consortium.

[3] John Chen. *Towards Efficient Statistical Parsing Using Lexicalized Grammatical Information*. PhD thesis, University of Delaware, 2001.

[4] David Chiang. Statistical parsing with an automatically extracted tree adjoining gramar. In *Data Oriented Parsing*. CSLI, 2003.

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

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

[7] Sumukh Ghodke and Steven Bird. Querying linguistic annotations. In *Proceedings of the Thirteenth Australasian Document Computing Symposium*, pages 69–72, Hobart, Australia, 2008.

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

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

[10] Seth Kulick, Ryan Gabbard, and Mitchell Marcus. Parsing the Arabic Treebank: Analysis and improvements. In *Proceedings of TLT 2006*. Treebanks and Linguistic Theories, 2006.

[11] Mohamed Maamouri, Ann Bies, and Seth Kulick. Upgrading and enhancing the Penn Arabic Treebank: A GALE challenge. In Joseph Olive, editor, *in progress for publication (book describing work in GALE program)*. 2009.

[12] Jiri Mirovsky. Netgraph - making searching in treebanks easy. In *Proceedings of the Third International Joint Conference on Natural Language Processing*, pages 945–950, Hyderabad, India, 2008.

[13] Beth Randall. Corpus search. http://sourceforge.net/projects/corpussearch/.

[14] B. Roark et al. Sparseval: Evaluation metrics for parsing speech. In *Fifth International Conference on Language Resources and Evaluation*, Genoa, Italy, 2006.

[15] Rushin Shah. The LDC Standard Arabic Morphological Tagger. talk at Linguistic Data Consortium.

[16] Fei Xia. *Automatic Grammar Generation From Two Different Perspectives*. PhD thesis, University of Pennsylvania, 2001.

[17] Nianwen Xue. Evaluating the impact of Chinese word segmentation on syntactic parsing, 2009. Book chapter for Global Automatic Language Exploitation.