Using Derivation Trees for Treebank Error Detection

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

Abstract

This work introduces a new approach to checking treebank consistency. Derivation trees based on a variant of Tree Adjoining Grammar are used to compare the annotation of word sequences based on their structural similarity. This overcomes the problems of earlier approaches based on using strings of words rather than tree structure to identify the appropriate contexts for comparison. We report on the result of applying this approach to the Penn Arabic Treebank and how this approach leads to high precision of error detection.

1 Introduction

The internal consistency of the annotation in a treebank is crucial in order to provide reliable training and testing data for parsers and linguistic research. Treebank annotation, consisting of syntactic structure with words as the terminals, is by its nature more complex and thus more prone to error than other annotation tasks, such as part-of-speech tagging. Recent work has therefore focused on the importance of detecting errors in the treebank (Green and Manning, 2010), and methods for finding such errors automatically, e.g. (Dickinson and Meurers, 2003b; Boyd et al., 2007; Kato and Matsubara, 2010).

We present here a new approach to this problem that builds upon Dickinson and Meurers (2003b), by integrating the perspective on treebank consistency checking and search in Kulick and Bies (2010). The approach in Dickinson and Meurers (2003b) has certain limitations and complications that are inherent in examining only strings of words. To overcome these problems, we recast the search as one of searching for inconsistently-used elementary trees in a Tree Adjoining Grammar-based form of the treebank. This allows consistency checking to be based on structural locality instead of n-grams, resulting in improved precision of finding inconsistent treebank annotation, allowing for the correction of such inconsistencies in future work.

2 Background and Motivation

2.1 Previous Work - DECCA

The basic idea behind the work in (Dickinson and Meurers, 2003a; Dickinson and Meurers, 2003b) is that strings occurring more than once in a corpus may occur with different "labels" (taken to be constituent node labels), and such differences in labels might be the manifestation of an annotation error. Adopting their terminology, a "variation nucleus" is the string of words with a difference in the annotation (label), while a "variation n-gram" is a larger string containing the variation nucleus.

(1) a. `(NP the (ADJP most important) points)`

b. `(NP the most important points)`

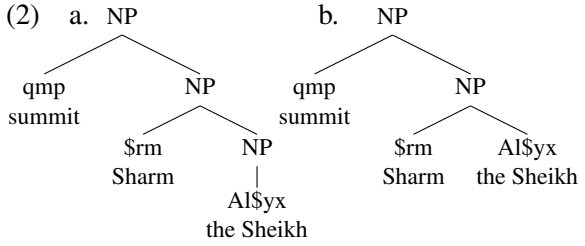
For example, suppose the pair of phrases in (1) are taken from two different sentences in a corpus. The "variation nucleus" is the string `most important`, and the larger surrounding n-gram is the string `the most important points`. This is an example of error in the corpus, since the second annotation is incorrect, and this difference manifests itself by the nucleus having in (a) the label ADJP but in (b) the default label NIL (meaning for their system that the nucleus has no covering node).

Dickinson and Meurers (2003b) propose a "non-

fringe heuristic", which considers two variation nuclei to have a comparable context if they are properly contained within the same variation n-gram - i.e., there is at least one word of the n-gram on both sides of the nucleus. For the the pair in (1), the two instances of the variation nucleus satisfy the non-fringe heuristic because they are properly contained within the identical variation n-gram (with `the` and `points` on either side). See Dickinson and Meurers (2003b) for details. This work forms the basis for the DECCA system.[1]

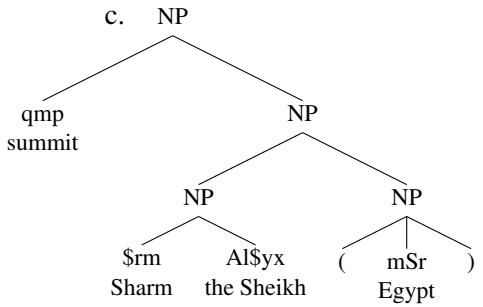
2.2 Motivation for Our Approach

(2) a.



b.

c.



We motivate our approach by illustrating the limitations of the DECCA approach. Consider the trees (2a) and (2b), taken from two instances of the three-word sequence `qmp $rm Al$yx` in the Arabic Treebank.[2] There is no need to look at any surrounding annotation to conclude that there is an inconsistency in the annotation of this sequence.[3] However, based on (2ab), the DECCA system would not even identify the three-word sequence `qmp $rm Al$yx` as a nucleus to compare, because both instances have a NP covering node, and so are considered to have the same label. (The same is true for the two-word subsequence `$rm Al$yx`.)

Instead of doing the natural comparison of the inconsistent structures for the identical word sequences as in (2ab), the DECCA approach would instead focus on the single word `Al$yx`, which has a NP label in (2a), while it has the default label NIL in (2b). However, whether it is reported as a variation depends on the irrelevant fact of whether the word to the right of `Al$yx` is the same in both instances, thus allowing it to pass the non-fringe heuristic (since it already has the same word, `$rm`, on the left).

Consider now the two trees (2bc). There is an additional NP level in (2c) because of the adjunct `( mSr )`, causing `qmp $rm Al$yx` to have no covering node, and so have the default label NIL, and therefore categorized as a variation compared to (2b). However, this is a spurious difference, since the label difference is caused only by the irrelevant presence of an adjunct, and it is clear, without looking at any further structure, that the annotation of `qmp $rm Al$yx` is identical in (2bc). In this case the "non-fringe heuristic" serves to avoid reporting such spurious differences, since if `qmp $rm Al$yx` did not have an open parenthesis on the right in (b), and `qmp` did not have the same word to its immediate left in both (b) and (c), the two instances would not be surrounded by the same larger variation n-gram, and so would not pass the non-fringe heuristic.

This reliance on irrelevant material arises from using on a single node label to characterize a structural annotation and the surrounding word context to overcome the resulting complications. Our approach instead directly compares the annotations of interest.

3 Using Derivation Tree Fragments

We utilize ideas from the long line of Tree Adjoining Grammar-based research (Joshi and Schabes, 1997), based on working with small "elementary trees" (abbreviated "etrees" in the rest of this paper) that are the "building blocks" of the full trees of a treebank. This decomposition of the full tree into etrees also results in a "derivation tree" that records how the elementary trees relate to each other.

We illustrate the basics of TAG-based derivation we are using with examples based on the trees in (2). Our grammar is a TAG variant with

[1] `http://www.decca.osu.edu/`.

[2] In Section 4 we give the details of the corpus. We use the Buckwalter Arabic transliteration scheme (Buckwalter, 2004).

[3] While the nature of the inconsistency is not the issue here, (b) is the correct annotation.

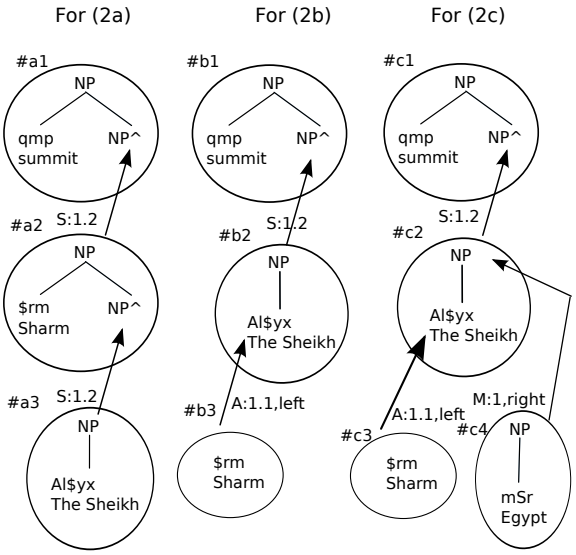


Figure 1: Etrees and Derivation Trees for (2abc).

tree-substitution, sister-adjunction, and Chomsky-adjunction (Chiang, 2003). Sister adjunction attaches a tree (or single node) as a sister to another node, and Chomsky-adjunction forms a recursive structure as well, duplicating a node. As typically done, we use head rules to decompose a full tree and extract the etrees. The three derivation trees, corresponding to (2abc), are shown in Figure 1.

Consider first the derivation tree for (2a). It has three etrees, numbered a1, a2, a3, which are the nodes in the derivation tree which show how the three etrees connect to each other. This derivation tree consists of just tree substitutions. The ^ symbol at node `NP^` in a1 indicates that it is a substitution node, and the `S:1.2` above a2 indicates that it substitutes into node at Gorn address 1.2 in tree a1 (i.e., the substitution node), and likewise for a3 substituting into a2. The derivation tree for (2b) also has three etrees, although the structure is different. Because the lower NP is flat in (2b), the rightmost noun, `Al$yx`, is taken as the head of the etree b2, with the degenerate tree for `$rm` sister-adjoining to the left of `Al$yx`, as indicated by the `A:1.1,left`. The derivation tree for (2c) is identical to that of (2b), except that it has the additional tree c4 for the adjunct `mSr`, which right Chomsky-adjoins to the root of c2, as indicated by the `M:1,right`.[4]

[4]We leave out the irrelevant (here) details of the parentheses in the derivation tree.

This tree decomposition and resulting derivation tree provide us with the tool for comparing nuclei without the interfering effects from words not in the nucleus. We are interested not in the derivation tree for an entire sentence, but rather only that slice of it having etrees with words that are in the nucleus being examined, which we call the derivation tree fragment. That is, for a given nucleus being examined, we partition its instances based on the covering node in the full tree, and within each set of instances we compare the derivation tree fragments for each instance. These derivation tree fragments are the relevant structures to compare for inconsistent annotation, and are computed separately for each instance of each nucleus from the full derivation tree that each instance is part of.[5]

For example, for comparing our three instances of `qmp $rm Al$yx`, the three derivation tree fragments would be the structures consisting of (a1, a2, a3), (b1, b2, b3) and (c1, c2, c3), along with their connecting Gorn addresses and attachment types. This indicates that the instances (2ab) have different internal structures (without the need to look at a surrounding context), while the instances (2bc) have identical internal structures (allowing us to abstract away from the interfering effects of adjunction).

Space prevents full discussion here, but the etrees and derivation trees as just described require refinement to be truly appropriate for comparing nuclei. The reason is that etrees might encode more information than is relevant for many comparisons of nuclei. For example, a verb might appear in a corpus with different labels for its objects, such as NP or SBAR, etc., and this would lead to its having different etrees, differing in their node label for the substitution node. If the nucleus under comparison includes the verb but not any words from the complement, the inclusion of the different substitution nodes would cause irrelevant differences for that particular nucleus comparison.

We solve these problems by mapping down the

[5]A related approach is taken by Kato and Matsubara (2010), who compare partial parse trees for different instances of the same sequence of words in a corpus, resulting in rules based on a synchronous Tree Substitution Grammar (Eisner, 2003). We suspect that there are some major differences between our approaches regarding such issues as the representation of adjuncts, but we leave such a comparison for future work.

System	nuclei	n-grams	instances
DECCA	24,319	1,158,342	2,966,274
Us	54,496	not used	605,906

Table 1: Data examined by the two systems for the ATB

System	nuclei found	non-duplicate nuclei found	types of inconsistency
DECCA	4,140	unknown	unknown
Us-internal	9,984	4,272	1,911

Table 2: Annotation inconsistencies reported for the ATB

representation of the etrees in a derivation tree fragment to form a "reduced" derivation tree fragment. These reductions are (automatically) done for each nucleus comparison in a way that is appropriate for that particular nucleus comparison. A particular etree may be reduced in one way for one nucleus, and then a different way for a different nucleus. This is done for each etree in a derivation tree fragment.

4 Results on Test Corpus

Green and Manning (2010) discuss annotation consistency in the Penn Arabic Treebank (ATB), and for our test corpus we follow their discussion and use the same data set, the training section of three parts of the ATB (Maamouri et al., 2008a; Maamouri et al., 2009; Maamouri et al., 2008b). Their work is ideal for us, since they used the DECCA algorithm for the consistency evaluation. They did not use the "non-fringe" heuristic, but instead manually examined a sample of 100 nuclei to determine whether they were annotation errors.

4.1 Inconsistencies Reported

The corpus consists of 598,000 tokens. Table 1 compares token manipulation by the two systems. The DECCA system[6] identified 24,319 distinct variation nuclei, while our system had 54,496. DECCA examined 1,158,342 n-grams, consisting of 2,966,274 instances (i.e., different corpus positions of the n-grams), while our system examined 605,906 instances of the 54,496 nuclei. For our system, the number of nuclei increases and the variation n-grams are eliminated. This is because all nuclei with more than one instance are evaluated, in order to search for constituents that have the same root but different internal structure.

The number of reported inconsistencies is shown in Table 2. DECCA identified 4,140 nuclei as likely errors - i.e., contained in larger n-grams, satisfying the non-fringe heuristic. Our system identified 9,984 nuclei as having inconsistent annotation - i.e., with at least two instances with different derivation tree fragments.

4.2 Eliminating Duplicate Nuclei

Some of these 9,984 nuclei are however redundant, due to nuclei contained within larger nuclei, such as `$rm Al$yx` inside `qmp $rm Al$yx` in (2abc). Eliminating such duplicates is not just a simple matter of string inclusion, since the larger nucleus can sometimes reveal different annotation inconsistencies than just those in the smaller substring nucleus, and also a single nucleus string can be included in different larger nuclei. We cannot discuss here the full details of our solution, but it basically consists of two steps.

First, as a result of the analysis described so far, for each nucleus we have a mapping of each instance of that nucleus to a derivation tree fragment. Second, we test for each possible redundancy (meaning string inclusion) whether there is a true structural redundancy by testing for an isomorphism between the mappings for two nuclei. For this test corpus, eliminating such duplicates leaves 4,272 nuclei as having inconsistent annotation. It is unknown how many of the DECCA nuclei are duplicates, although many certainly are. For example, `qmp $rm Al$yx` and `$rm Al$yx` are reported as separate results.

4.3 Grouping Inconsistencies by Structure

Across all variation nuclei, there are only a finite number of derivation tree fragments and thus ways in which such fragments indicate an annotation inconsistency. We categorize each annotation inconsistency by the inconsistency type, which is simply a set of numbers representing the different derivation

[6] We worked at first with version 0.2 of the software. However this software does not implement the non-fringe heuristic and does not make available the actual instances of the nuclei that were found. We therefore re-implemented the algorithm to make these features available, being careful to exactly match our output against the released DECCA system as far as the nuclei and n-grams found.

tree fragments. We can then present the results not by listing each nucleus string, but instead by the inconsistency types, with each type having some number of nuclei associated with it.

For example, instances of `$rm Al$yx` might have just the derivation tree fragments (a2, a3) and (b2, b3) in Figure 1, and the numbers representing this pair is the "inconsistency type" for this (nucleus, internal context) inconsistency. There are nine other nuclei reported as having an inconsistency based on the exact same derivation tree fragments (abstracting only away from the particular lexical items), and so all these nuclei are grouped together as having the same "inconsistency type". This grouping results in the 4,272 non-duplicate nuclei found being grouped into 1,911 inconsistency types.

4.4 Precision and Recall

The grouping of internal checking results by inconsistency types is a qualitative improvement in consistency reporting, with a high precision.[7] By viewing inconsistencies by structural annotation types, we can examine large numbers of nuclei at a time. Of the first 10 different types of derivation tree inconsistencies, which include 266 different nuclei, all 10 appear to real cases of annotation inconsistency, and the same seems to hold for each of the nuclei in those 10 types, although we have not checked every single nucleus. For comparison, we chose a sample of 100 nuclei output by DECCA on this same data, and by our judgment the DECCA precision is about 74%, including 15 duplicates.

Measuring recall is tricky, even using the errors identified in Green and Manning (2010) as "gold" errors. One factor is that a system might report a variation nucleus, but still not report all the relevant instances of that nucleus. For example, while both systems report `$rm Al$yx` as a sequence with inconsistent annotation, DECCA only reports the two instances that pass the "non-fringe heuristic", while our system lists 132 instances of `$rm Al$yx`, partitioning them into the two derivation tree fragments. We will be carrying out a careful accounting of the recall evaluation in future work.

[7] "Precision" here means the percentage of reported variations that are actually annotation errors.

5 Future Work

While we continue the evaluation work, our primary concern now is to use the reported inconsistent derivation tree fragments to correct the annotation inconsistencies in the actual data, and then evaluate the effect of the corpus corrections on parsing. Our system groups all instances of a nucleus into different derivation tree fragments, and it would be easy enough for an annotator to specify which is correct (or perhaps instead derive this automatically based on frequencies).

However, because the derivation trees and etrees are somewhat abstracted from the actual trees in the treebank, it can be challenging to automatically correct the structure in every location to reflect the correct derivation tree fragment. This is because of details concerning the surrounding structure and the interaction with annotation style guidelines such as having only one level of recursive modification or differences in constituent bracketing depending on whether a constituent is a "single-word" or not. We are focusing on accounting for these issues in current work to allow such automatic correction.

Acknowledgments

We thank the computational linguistics group at the University of Pennsylvania for helpful feedback on a presentation of an earlier version of this work. We also thank Spence Green and Chris Manning for supplying the data used in their analysis of the Penn Arabic Treebank. This work was supported in part by the Defense Advanced Research Projects Agency, GALE Program Grant No. HR0011-06-1-0003 (all 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

Adriane Boyd, Markus Dickinson, and Detmar Meurers. 2007. Increasing the recall of corpus annotation error detection. In *Proceedings of the Sixth Workshop on Treebanks and Linguistic Theories (TLT 2007)*, Bergen, Norway.

Tim Buckwalter. 2004. Buckwalter Arabic morphological analyzer version 2.0. Linguistic Data Consortium LDC2004L02.

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

Markus Dickinson and Detmar Meurers. 2003a. Detecting errors in part-of-speech annotation. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL-03)*, pages 107–114, Budapest, Hungary.

Markus Dickinson and Detmar Meurers. 2003b. Detecting inconsistencies in treebanks. In *Proceedings of the Second Workshop on Treebanks and Linguistic Theories (TLT 2003)*, Sweden. Treebanks and Linguistic Theories.

Jason Eisner. 2003. Learning non-isomorphic tree mappings for machine translation. In *The Companion Volume to the Proceedings of 41st Annual Meeting of the Association for Computational Linguistics*, pages 205–208, Sapporo, Japan, July. Association for Computational Linguistics.

Spence Green and Christopher D. Manning. 2010. Better Arabic parsing: Baselines, evaluations, and analysis. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 394–402, Beijing, China, August. Coling 2010 Organizing Committee.

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.

Yoshihide Kato and Shigeki Matsubara. 2010. Correcting errors in a treebank based on synchronous tree substitution grammar. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 74–79, Uppsala, Sweden, July. Association for Computational Linguistics.

Seth Kulick and Ann Bies. 2010. A TAG-derived database for treebank search and parser analysis. In *TAG+10: The 10th International Conference on Tree Adjoining Grammars and Related Formalisms*, Yale.

Mohamed Maamouri, Ann Bies, Seth Kulick, Fatma Gaddeche, Wigdan Mekki, Sondos Krouna, and Basma Bouziri. 2008a. Arabic treebank part 1 - v4.0. Linguistic Data Consortium LDC2008E61, December 4.

Mohamed Maamouri, Ann Bies, Seth Kulick, Fatma Gaddeche, Wigdan Mekki, Sondos Krouna, and Basma Bouziri. 2008b. Arabic treebank part 3 - v3.0. Linguistic Data Consortium LDC2008E22, August 20.

Mohamed Maamouri, Ann Bies, Seth Kulick, Fatma Gaddeche, Wigdan Mekki, Sondos Krouna, and Basma Bouziri. 2009. Arabic treebank part 2- v3.0. Linguistic Data Consortium LDC2008E62, January 20.