Further Developments in Treebank Error Detection Using Derivation Trees

Seth Kulick, Ann Bies, Justin Mott

Linguistic Data Consortium
University of Pennsylvania
Philadelphia, PA 19104
{skulick,bies,jmott}@ldc.upenn.edu

Abstract

This work describes how derivation tree fragments based on a variant of Tree Adjoining Grammar (TAG) can be used to check treebank consistency. Annotation of word sequences are compared both for their internal structural consistency, and their external relation to the rest of the tree. We expand on earlier work in this area in three ways. First, we provide a more complete description of the system, showing how a naive use of TAG structures will not work, leading to a necessary refinement. We also provide a more complete account of the processing pipeline, including the grouping together of structurally similar errors and their elimination of duplicates. Second, we include the new experimental external relation check to find an additional class of errors. Third, we broaden the evaluation to include both the internal and external relation checks, and evaluate the system on both an Arabic and English treebank. The evaluation has been successful enough that the internal check has been integrated into the standard pipeline for current English treebank construction at the Linguistic Data Consortium

Keywords: Quality control, treebanking, Tree Adjoining Grammar

1. Introduction

Treebank annotation, consisting of syntactic structure with words as the terminals, is by its nature more complex and thus more prone to error than many other annotation tasks, such as part-of-speech tagging. However, as with other types of annotation, it is crucial that annotation consistency be high in order to provide reliable training and testing data for parsers and linguistic research. 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).

In Kulick et al. (2011), we introduced a new approach to this problem that improves upon Dickinson and Meurers (2003b) by decomposing the full syntactic tree into smaller units, utilizing ideas from Tree Adjoining Grammar (TAG) (Joshi and Schabes, 1997). This allows the comparison to be based on meaningful syntactic units instead of string n-grams. In this paper we develop this earlier work in three ways:

1. We expand on the earlier description. First, we explain how a naive use of TAG structures will not work, and give our solution to this problem. Second, we give a more complete account of the entire processing pipeline, placing the use of the TAG-based comparison in the context of how the tree fragments to be compared are determined. We now call this comparison an "internal relation check", to distinguish it from the following new search.

2. We discuss a class of errors that the earlier work - the internal relations check - would not find, and how we have experimented with a new error search, the "external relation check".

3. We broaden the evaluation, to include both the internal and external checks as well as additional large English corpora.

The evaluation has been successful enough that the "internal" check has now been integrated into the standard pipeline for current English treebank annotation at the Linguistic Data Consortium (Bies et al., 2012).

2. Background

In this section we present an overview of the system as described in Kulick et al. (2011). The goal of our system is to detect sequences that are annotated in inconsistent ways by comparing local syntactic units.

The example in Figure 1 is from the Penn Arabic Treebank (ATB) (Maamouri et al., 2010), using the Buckwalter (2004) transliteration. The three trees in the top half of the figure each represent a differently annotated instance of the string `qmp $rm Al$yx`. The annotation in (A) and (B) is inconsistent because (A) is fully right-branching, whereas (B) is not. In contrast, although (B) and (C) look dissimilar, the annotation in (C) is consistent with that in (B), with regard to `qmp $rm Al$yx`. Undoing the adjunction in (C) and removing the modifier `mSr` yields a tree that is identical to (B).

Following the TAG approach, we decompose the full phrase structure into smaller syntactic chunks called elementary trees (henceforth, e-trees), with certain operations composing them with one another. Our method uses the three operations in Figure 2: (1) substitution, which substitutes one tree for a target node in another tree, (2) adjunction, which attaches one tree to a target node in another tree by creating a copy of the target node[1], and (3) sister adjunction, which attaches one tree (often only a single node) as a sister to a target node in another tree. (The operations in Figure 2 are somewhat different from those used in standard TAG, and the formalism underlying our system is properly speaking a variant of Tree Insertion Grammar, closely related to the system in Chiang (2003).)

[1]Also known as Chomsky adjunction.

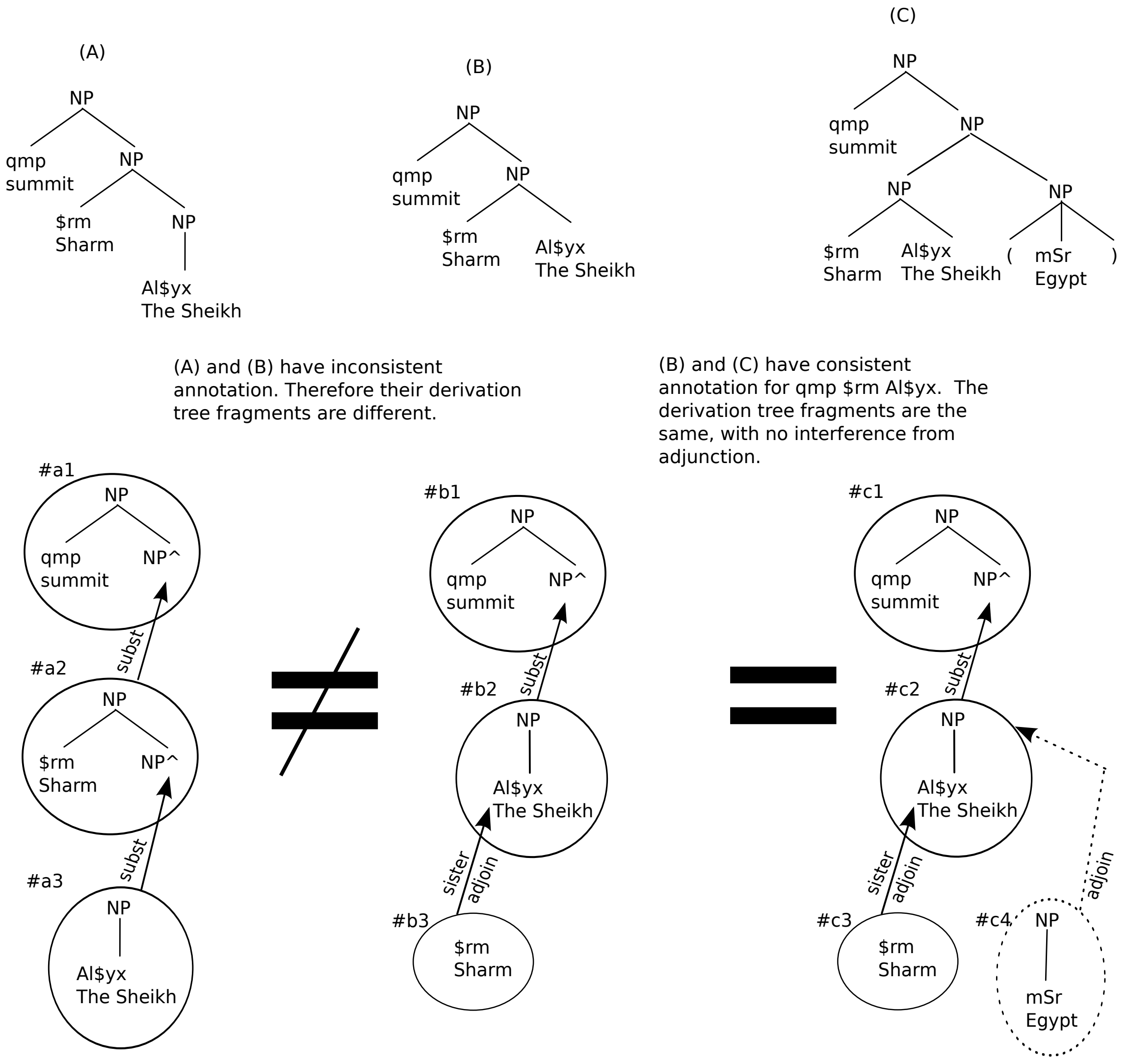


Figure 1: Examples of inconsistent and consistent annotation

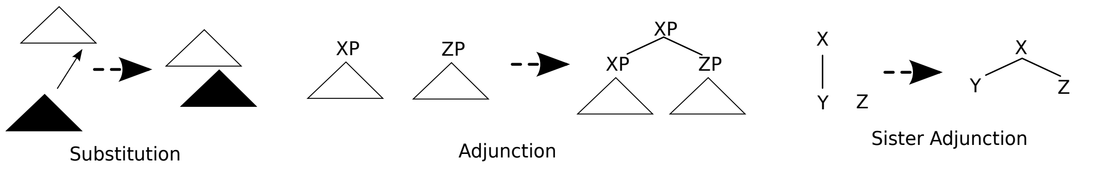


Figure 2: Three TAG Operations

The relationship of the e-trees underlying a full phrase structure tree to each other is recorded in a derivation tree, which specifies how the e-trees combine to form the original full tree. The bottom half of Figure 1 shows the derivation trees corresponding to the phrase structure trees in the top half. Without walking through all the details, each node in the derivation tree is an e-tree, related to its parent node in the derivation tree by one of the three composition operations. In general, we are not concerned with derivation trees for entire sentences, but rather derivation tree fragments: just those nodes (i.e., e-trees) that contain a word in the string being compared. These derivation tree fragments are the basis for our system's comparison of different instances of the same string.

For example, the derivation tree fragments for our three instances of `qmp $rm Al$yx` are the structures consisting of (a1, a2, a3), (b1, b2, b3) and (c1, c2, c3), along with the operations relating them. The structures in (A) and (B) are reported as inconsistent, since their derivation tree fragments (a1, a2, a3) and (b1, b2, b3) are different, with a difference in both the e-trees and the composition operations relating them. But (B) and (C) are reported as consistent, since the e-trees for the string, (b1, b2, b3) and (c1, c2, c3), as well as the operations relating them are identical. Because we are evaluating `qmp $rm Al$yx`, the modifier in (C) is not relevant to the evaluation, and because of the TAG tree decomposition, it correctly does not interfere with the comparison.

In Kulick et al. (2011), we compare our approach to that

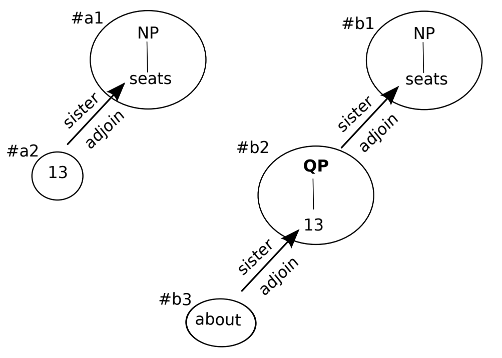


Figure 3: Derivation tree fragments corresponding to (1ab)

of the DECCA system[2], based on Dickinson and Meurers (2003b). We discuss there the advantages of basing a comparison on tree fragments instead of string n-grams.

We would like to stress that there is no one correct way to extract elementary trees from the full phrase structure tree. The decomposition can be done in different ways, based on choices of which (parent,head) relationships to use, or of what constitutes an argument, or even whether substitution nodes should be used at all (Shen et al., 2008). We have chosen a fairly traditional approach, although it is possible that we may modify this in the future to more closely fit the needs of this work. The reason that we make heavy use of sister adjunction is because this allows us to create a derivation tree that can be used to exactly reconstruct the original phrase structure tree as it exists in the treebank. Due to the focus of this work, on identifying inconsistencies in the existing treebank, we did not want to follow some earlier TAG work that, in order to make heavier use of recursive adjunction, first manipulated the structure of the phrase structure trees before doing the extraction.

3. Reduced derivation tree fragments

3.1. The problem

The naive use of the derivation tree fragments as just described does not however give entirely appropriate results. The reason is that the e-trees might encode more information than is relevant for the comparison of annotation for a different instances of a string. We solve this problem by (automatically) mapping down the representation of the e-trees in a derivation tree fragment to form a "reduced" derivation tree fragment, optimized for annotation comparison.

We describe here two main aspects of this problem, as briefly mentioned in Kulick et al. (2011). These are both related to the fact that the tree decomposition does not lose any information, and so every node in the original tree must be in some e-tree in the resulting decomposition.

(A) Because of rules in the annotation guidelines, particularly regarding single vs. multi-word constituents, structures can differ in extra structure in a way that does not matter for error detection. For example, consider the nucleus `13 seats` appearing as an instance in two sentences, with the two constituents in (1ab).

[2]`http://decca.osu.edu/`

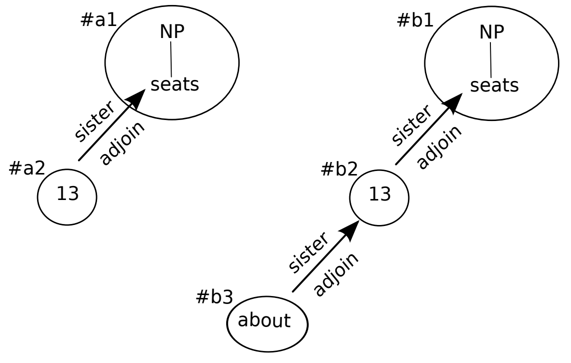
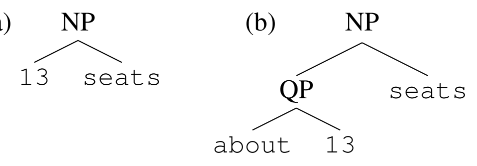


Figure 4: Reduced derivation tree fragments corresponding to (1ab)

(1) (a)


In (1b), the QP node is present because it is the covering node for a multi-word constituent. However, in (1a), the QP node is not present because it would be the covering node for a single-word constituent, and by the Penn Treebank (and similar) guidelines (Bies et al., 1995), the QP is left "implicit" in such cases.

The result in terms of the e-trees and derivation tree is shown in Figure 3. The two derivation tree fragments for the string `13 seats` would be (a1,a2) and (b1,b2) (b3 is not included because it is not included in the string). Although the difference between a2 and b2 would cause these two derivation tree fragments to be characterized as inconsistent, the difference is not in fact an annotation error.

(B) The nature of the e-tree extraction process might lead to nodes being included in an e-tree that are irrelevant for the nucleus being examined. 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 e-trees, 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.[3]

3.2. The solution

We solve these problems by mapping down the representation of the e-trees 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 e-tree may be reduced in one way for one nucleus, and then a different way for a different nucleus. This is done for each e-tree in a derivation tree fragment.

First, all substitution nodes that do not include words in the nucleus are deleted. This is case `(B)`. Second, start-

[3]It is interesting to note that extracting the elementary trees in a different way, as in (Shen et al., 2008), would avoid this problem since substitution nodes are not used at all. We will explore this option in the future.

ing from the root, all nodes with only one child are deleted, with the procedure stopping when a node is not deleted. This is case `(A)`, and would eliminate the QP node from e-tree b2 in Figure 3. After this reduction, the two instances of the nucleus `13 seats`, one arising from `(NP 13 seats)` and the other arising from `(NP (QP about 13) seats)`, would have the same reduced derivation tree fragments, namely (a1,a2) and (b1,b2) in Figure 4.

In addition, we also include the POS tags for each word included in the nucleus (and therefore in the derivation tree fragment).[4] This allows us to check for consistency of POS tags at the same time as checking the structure, because whenever the POS tags on words affect the tree decomposition, this will result in differing derivation trees.[5] Examples showing the ability to find POS errors will be seen in Section 6.

4. Internal check of nuclei

The reduced derivation tree fragments described in Section 3 are the basis for the "internal" check of each nucleus found. We call this an "internal" check because it checks for consistency of the internal structure of the reduced derivation tree fragments for all instances of some nucleus.

4.1. Overall processing

The overall procedure is as follows:[6]
(1) All constituents are identified. The set of strings that make up these constituents is the set of nuclei.
(2) For each nucleus (a string of words), we find all other instances of that nucleus (whether they are constituents or not) in other trees.
(3) For each nucleus, for each instance, we form the derivation tree fragment for that instance (reduced as appropriate for the nucleus). We identify the highest node in the highest e-tree in the derivation tree fragment. We call this our "internal context" (meaning the context for the internal check).
(4) For each nucleus, we partition the instances based on the internal context.
(5) We compare the derivation tree fragments for the instances in each partition. If there is any difference, then there is an inconsistency for that partition group (as defined by the internal context) for that nucleus, and thus for the nucleus as a whole.

[4]There are some other details of this e-tree reduction that we do not discuss in detail here. The most notable one is that we replace Gorn addresses to indicate substitution/modification locations with the name of the node that is the locus of substitution/ modification. The reduced derivation tree fragment comparisons therefore abstract away from irrelevant node address differences which can be caused by additional material on top (e.g., SBAR instead of S when comparing instances of a nucleus containing a verb).

[5]This therefore also captures in a natural way the POS-checking work in Dickinson and Meurers (2003a).

[6]Steps (1) and (2) are identical to that in the DECCA system, while all the following steps are different.

4.2. Example

Consider again the three instances of the nucleus `qmp $rm Al$yx` in Figure 1. They have the same internal context NP, and for exactly this reason are compared as part of the same (nucleus, internal context) partition. This comparison allows us to capture a reported inconsistency due to the differences in the derivation tree fragments.

An important benefit of viewing inconsistencies as differing derivation tree fragments is that it lets us group together different nuclei as having the same type of annotation inconsistency. We illustrate this in the context of full actual results, in Section 6.1.1.

4.3. Categorizing duplicate nuclei

It is often the case that a nucleus is contained within a larger nucleus. For example, while the nucleus `qmp $rm Al$yx` will contain, as just discussed, a comparison of the instances in Figure 1, the smaller nucleus `$rm Al$yx` will also trigger such a comparison. In particular, the larger nucleus, within the group for the internal context NP, will compare the derivation tree fragments (a1,a2,a3) and (b1,b2,b3) from Figure 1, thus finding the difference in annotation.[7] The smaller nucleus, within the group for the internal context NP for the smaller nucleus, will compare the derivation tree fragments (a2,a3) and (b2,b3), also finding the difference in annotation.

Obviously they are finding the same difference in annotation, and it would be redundant and annoying to report both nuclei as having inconsistent annotation. However, eliminating such duplicate information from the report is not just a matter of testing whether the smaller nucleus is contained as a string in the larger nucleus. It can happen that a larger nucleus will contain structural inconsistencies that are not just redundant to the structural information in a smaller, enclosed nucleus.

For each such pair of nuclei, we base the test for duplicate information on the instances and their derivation tree fragments for each (nucleus, internal context) partition group as discussed in steps (4, 5) in Section 4.1. Step (5), the examination of the derivation tree fragments for the instances, can be viewed as a mapping from instances to derivation tree fragments. If there is an isomorphism of the mappings in the (larger nucleus, internal context) group and the (smaller nucleus, internal context) group, then the information is redundant. For example, the two instances of the larger nucleus `qmp $rm Al$yx` map, respectively, to (a1,a2,a3) and (b1,b2,b3). The two instances of smaller nucleus `$rm Al$yx` map, respectively, to (a2,a3) and (b2,b3). This is an isomorphism between the mappings, and so the larger nucleus is considered redundant, and listed as being in the same "group" as the smaller nucleus.

5. External relation check

The consistency checking described in Sections 2 and 3 will miss cases in which the annotation inconsistency is not because of a structural difference internal to the derivation tree

[7]Actually it compares the reduced derivation tree fragments, as discussed in Section 3.

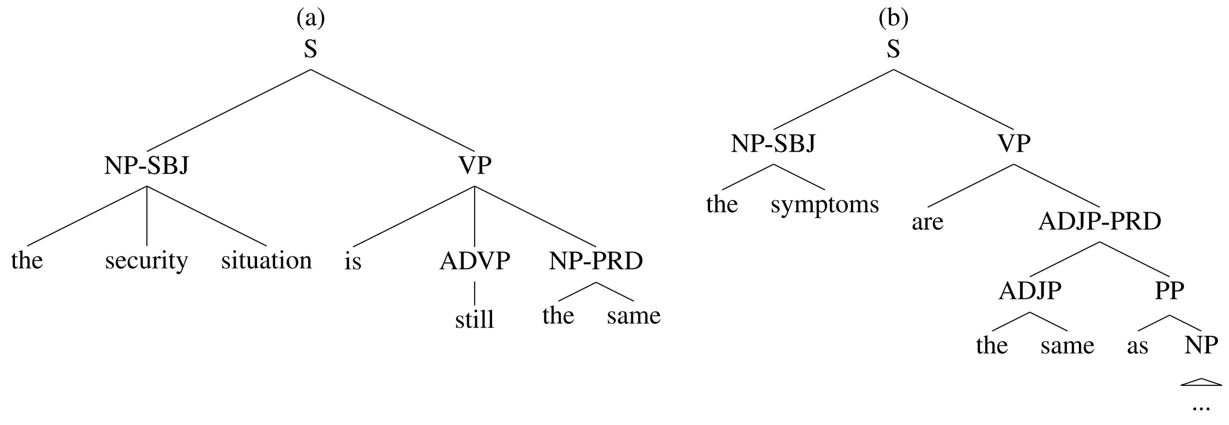


Figure 5: Phrase structure trees for `the security situation is still the same` and `the symptoms are the same as ...`

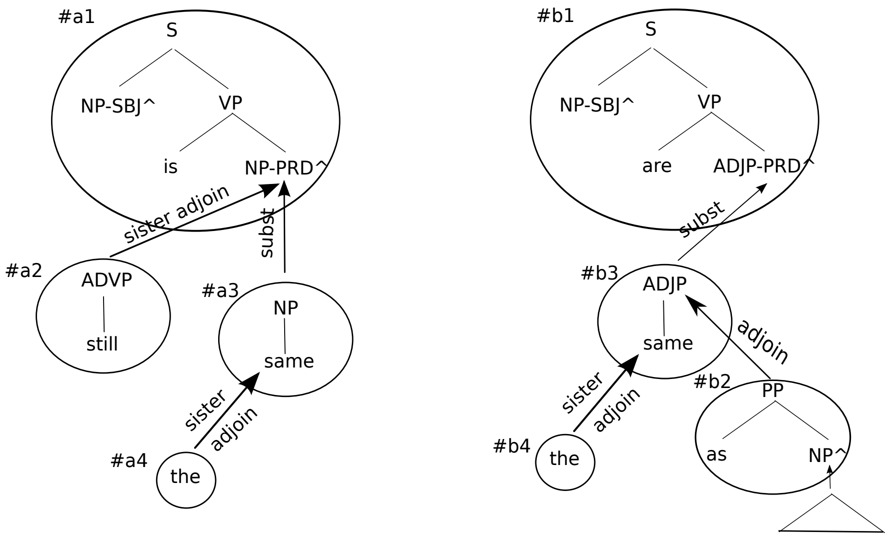


Figure 6: Derivation tree fragments for the phrase structure trees in Figure 5.

fragment, but rather in the constituent labels covering different instances of a string. Therefore we also implement an "external relation check" to consider how that node interacts with the rest of the tree, regardless of the internal structure it may contain, in contrast to the "internal check" of Sections 2 and 3, which essentially checks for consistent node label projection.

We use the same idea of a derivation tree fragment as for the internal check, except that instead of comparing the structures of the corresponding fragments, we compare only the information at the highest node label in the fragments. Since the same string can project to different constituents, we only compare instances of that string if they have the same "context", which is a shorthand description of how the derivation tree fragment interacts with the full derivation tree - e.g., whether it substitutes into another e-tree, or modifies (adjoins into) it, etc.

Figure 5 shows the phrase structure trees for the sentences `the security situation is still the same` and `the symptoms are the same as...`. The NP `the same` is inconsistently annotated, being a NP in (a) and an ADJP in (b). Figure 6 shows the derivation tree structures for these two trees, and the derivation tree fragment for the nucleus string `the same` consists of the e-trees (a3,a4) and (b3,b4), respectively.
In both cases the context for `the same` is substitution into another e-tree. The two instances are therefore compared by our external relation check, and the NP/ADJP difference is flagged as an inconsistency. The presence of `still` in one but not the other prevents Dickinson and Meurers (2003b) from finding this difference, but because our system deals with structural locality, it is able to find this inconsistency.

6. Results on test corpora

We developed our system using a small test corpus of 30000 words, a pre-release version of an English corpus of translated Arabic broadcast news (Bies et al., 2009). Example (1) and the derivation tree extractions in Figure 6 were taken from this corpus.[8]
For evaluation, we used two corpora. The first is the Penn Arabic Treebank (ATB), for which we used the same versions and subset as used for the discussion of annotation

[8]We do not discuss this corpus further since it was so small.

Check	Nuclei found	Non-duplicate nuclei found	Types of inconsistency
internal	9984	4272	1911
external	191	unknown	n/a

Table 1: Annotation inconsistencies reported for the ATB

consistency in Green and Manning (2010).[9] Their work is ideal for this comparison, since they provide what can be considered "gold" errors. with a manual examination of a sample of 100 nuclei to determine whether they were in fact annotation errors. For our second corpus, we used a subset of the English treebank newswire section of the Ontonotes 4.0 release (Weischedel et al., 2011).

6.1. Penn Arabic Treebank results

The corpus consists of 598,000 tokens. Our system found 54,496 nuclei, consisting of 605,906 instances.[10] The number of reported inconsistencies is shown in Table 1. Our system identified 9984 nuclei as having inconsistent annotation using the internal check, and another 191 using the external check. However, as described in Section 4.3, some of these are redundant, due to nuclei contained within larger nuclei, and eliminating such duplicates leaves 4272 nuclei as having inconsistent annotation.

6.1.1. Inconsistencies grouped by structure

For the purposes of understanding and correcting inconsistencies, it is advantageous to examine together different nuclei with the same structural inconsistency, even with different lexical strings. Our grouping by inconsistency type allows for different nuclei (i.e., different strings) with the same structural patterns to be viewed as a single group, thus making it much easier to examine the results.

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 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 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 types". This grouping results in the 4272 non-duplicate nuclei found by the internal check being grouped into 1911 inconsistency types.

This grouping of results into inconsistency types if a crucial of using of integrating this work into an annotation project, since it facilitates more efficient manual examination of the results.

[9]This is the usual training section of three sections of ATB123 (Maamouri et al., 2008a; Maamouri et al., 2009; Maamouri et al., 2008b). Note however that these are not the current versions of the corpora.

[10]The nuclei can different lengths, and it is possible for a single word to be therefore be included as an instance of more than one nucleus.

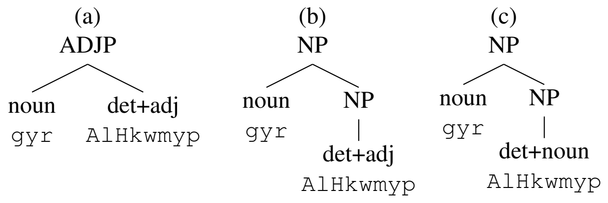


Figure 7: Three instances of `gyr AlHkwmyp` (*non-governmental*)

Check	Nuclei found	Non-duplicate nuclei found	Types of inconsistency
internal	3609	3012	1186
external	859	unknown	n/a

Table 2: Annotation inconsistencies reported for Ontonotes

6.2. Precision and recall

The grouping of internal checking results by inconsistency types allows for a high precision in reporting inconsistency results.[11] Because we can view 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.

Measurement of recall is ongoing at the present, along with the experimental external relation check precision and recall. Initial results however are promising and will inform future work. Consider the examples examples (a,b,c) in Figure 7, which (Green and Manning, 2010) found. The inconsistency between (b) and (c) was found by the internal check, since they are both in the NP internal context for the nucleus `gyr AlHkwmyp`, but the POS tag on `AlHkwmyp` is different, which resulted in a difference in the e-tree representation. The inconsistency between (a) and (bc) was found by the external check, because of the ADJP/NP difference, since the context (substitution into another e-tree) was the same in all three cases.

6.3. Ontonotes English treebank results

The corpus excerpt consists of 524,845 tokens. Our system found 30,497 nuclei, consisting of 619,415 instances. Our system identified 3,609 nuclei as having inconsistent annotation using the internal check, reducing to 3,012 when taking into account duplicates, with 1186 inconsistency types. The external check identified another 859 nuclei.

Preliminary investigation shows though that the internal

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

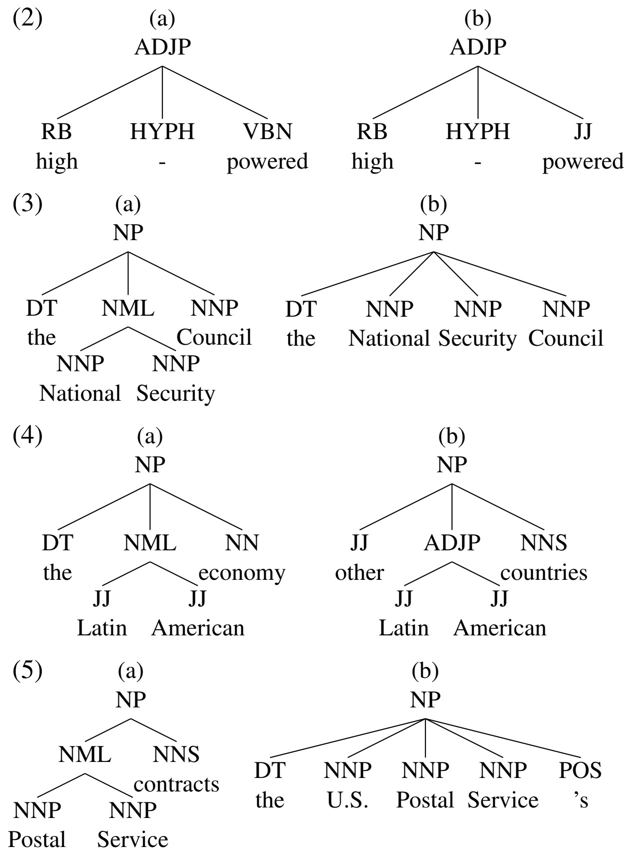


Figure 8: Example system output for Ontonotes

check is finding many POS inconsistencies[12]. An example is shown in (2). As with the results in the ATB, the differing derivation tree fragments are grouped together as an inconsistency type (abstracting from the actual words, although including the POS tags), and the nuclei `well - developed` and `well - known` pattern the same as `high - powered`, and so are reported as having the same annotation inconsistency type.

We also find structural inconsistencies, relating to the use of the NML node label. For example, of the four cases of `the National Security Council` appearing with the internal context NP, three have the annotation (3a), and one has the annotation (3b).[13] While there is clearly much more to do to evaluate the precision (and recall, to the extent that we can) of our results, we note that DECCA does not find any of the Ontonotes inconsistency examples described in this section.

The increased number of external check nuclei reported compared to the ATB needs explanation. Some of the nuclei do appear to be annotation inconsistencies. For example, the nucleus `Latin American` is annotated as either NML or ADJP in similar contexts in (4). However, about half the cases are due to annotation differences caused by the POS (possessive) part-of-speech tag. For example, `Postal Service` is reported as inconsistent, as in (5), even though the guidelines state that the presence of the `(POS 's)` cause the entire NP to be flat, as in (5b). We can adjust the definition of external context to differentiate this properly, but we are reluctant to do this, since we find such cases interesting as contexts that might be difficult for a parser. In the future we might choose different notions of external context depending on different purposes.

6.4. Manual Adjudication

As mentioned in the introduction, we are currently using this work in the ongoing English treebank annotation pipeline at LDC. While the precision results in the evaluation are good, they of course cannot be assumed to be perfect in a real-life application. We therefore include an extensive manual adjudication process in the annotation pipeline to distinguish actual errors from spurious results, and to correct the actual errors as necessary in the trees. We are also currently working on improved software infrastructure to integrate the quality control output with the tree annotation software.

7. Future work

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). There are two main differences that we can see, pending a future comparison: (1) we extract out interference from adjuncts, for the reasons discussed in Section 2, and (2) we focus on identifying inconsistent annotations of word sequences, for which the corpus developers can then select the correct annotation.[14]

While we are continuing the evaluation work, in particular for the external check, our primary concern now is to improve the integration of the internal check into the annotation pipeline, as discussed in Section 6.4. There are also additional potential methods of integrating this work into an annotation pipeline. For example, instead of running this process on the result of the manual annotation, it could be run first on the output of the parser, before any manual annotation, allowing the possibility of automatically identifying parser errors. There are also many possibilities for further improving this work, such as backing off from reliance on word identity to something less brittle, perhaps relying on POS tags.

We are also intending to use this approach for inter-annotator agreement analysis during treebank construction, comparing multiple annotations of the same text. This should be a natural fit since the identical strings used for nuclei are present by definition.

8. Acknowledgements

This work was supported in part by the Defense Advanced Research Projects Agency, GALE Program Grant No. HR0011-06-1-0003. The views, opinions and/or findings contained in this article/presentation are those of the author/presenter and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency

[12]Without "gold" errors, it is difficult to score recall at all.

[13]Four other nuclei have the same inconsistency.

[14]As discussed in Section 6.1.1, we do group nuclei with the same structural inconsistencies together, but such identical inconsistencies are always linked to specific nuclei.

or the Department of Defense. We would also like to thank Mohamed Maamouri, Colin Warner, Aravind Joshi, and Mitch Marcus for valuable conversations and feedback.

9. References

Ann Bies, Mark Ferguson, Karen Katz, and Robert MacIntyre. 1995. Bracketing guidelines for Treebank II-style Penn Treebank project. Technical Report MS-CIS-95-06, University of Pennsylvania.

Ann Bies, Justin Mott, and Colin Warner. 2009. English Translation treebank - EATB part3 v2.0. Linguistic Data Consortium LDC2009E55, June.

Ann Bies, Justin Mott, Colin Warner, and Seth Kulick. 2012. Google Web Treebank, v1.0. Linguistic Data Consortium LDC2012R22.

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, Ann Bies, and Justin Mott. 2011. Using derivation trees for treebank error detection. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 693–698, Portland, Oregon, USA, June. Association for Computational Linguistics.

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.

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

Libin Shen, Lucas Champollion, and Aravind Joshi. 2008. LTAG-spinal and the Treebank: A new resource for incremental, dependency and semantic parsing. *Language Resources and Evaluation*, 42(1):1–19.

Ralph Weischedel, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, Mohammed El-Bachouti, Robert Belvin, and Ann Houston. 2011. OntoNotes 4.0. Linguistic Data Consortium LDC2011T03.