Improving Dependency Label Accuracy using Statistical Post-editing: A Cross-Framework Study

Abstract

We present a statistical post-editing method for modifying the dependency labels in a dependency analysis. We test the method using two English datasets, three parsing systems and three labelled dependency schemes. We demonstrate how it can be used both to improve dependency label accuracy in parser output and highlight problems with and differences between constituency-to-dependency conversions.

1 Introduction

The quality of dependency analyses produced by automatic parsing is usually evaluated using both *attachment accuracy* and *label accuracy*. A parsing system's attachment accuracy reflects its ability to recover structure correctly, i.e. dependencies between heads and dependents. Label accuracy, on the other hand, reflects the system's ability to correctly determine the nature of these dependencies. In order to ascertain *who* did *what* to *whom*, the dependency labels are crucial since they allow us to distinguish between grammatical roles (subjects versus objects, indirect objects versus adverbial modifiers, etc.). In this paper we focus on dependency labels and present a post-editing method for boosting label accuracy.

The idea behind the method is to automatically capture systematic error patterns characterised by local features. A set of parser output dependency analyses is compared to a set of gold standard analyses and a label revision model is learned which can then be applied to new dependency analyses. We experiment with two feature sets to condition the probability of a label. The first makes use of lexical information and the second includes more structural context. We find that both feature sets are effective on their own but are more so when we backoff to the non-lexicalised feature set in the event that the lexicalised feature set does not make a prediction.

The method is designed to fix labelling errors rather than attachment errors, and in that it differs from the tree revision rules of Attardi and Ciaramita (2007). Label and attachment post-editing can be viewed as complementary techniques and in practice may potentially be combined within one system. To our knowledge, this is the first post-editing method to target dependency label accuracy.

In order to fully demonstrate the strengths and weaknesses of the post-editing method, we apply it to two datasets, three parsers and three labelled dependency schemes. In theory, the method is language-independent, although, in this study, we concentrate on English. Our two main datasets are the Wall Street Journal Section of the Penn Treebank (Marcus et al., 1994) and QuestionBank (Judge et al., 2006). We employ two dependency parsers and one constituency parser. The dependency parsers are trained directly on dependency trees produced by applying constituency-to-dependency conversion to Penn Treebank constituency trees. The constituency parser, on the other hand, is trained on the Penn Treebank constituency trees and its output is converted to dependency trees using the same conversion procedure. The dependency parsers we employ are MaltParser (Nivre et al., 2006) and MSTParser

(McDonald et al., 2005), and the constituency parser is the two-stage Charniak and Johnson reranking parser (Charniak and Johnson, 2005). The use of more than one labelled dependency scheme is desirable not only because there is no one standard dependency scheme for English but also because it allows us to highlight some of the differences between the various schemes. The three schemes we employ are LTH (Johansson and Nugues, 2007), Stanford (de Marneffe et al., 2006; de Marneffe and Manning, 2008) and LFGDEP (Cetinoglu et al., 2010).

The post-editing method results in improved labelled attachment scores for the Charniak and Johnson parser and the three dependency schemes. For two of the schemes, the improvements are statistically significant (89.92 → 91.12 for LTH and 90.67 → 90.88 for LFGDEP). The method does not work as well for the two dependency parsers. Because the Charniak and Johnson parser has higher unlabelled attachment accuracy than MaltParser and MSTParser, it is able to benefit more from the method since label modifications can only be learned from correctly attached dependencies. We also find that the post-editor works when trained on the same data on which the parser was trained. This is an encouraging practical result since it demonstrates that improvements may be achieved at no additional annotation cost.

The difference between the Stanford scheme and the LTH and LFGDEP schemes is that the Stanford scheme has been designed to be applied to constituency trees which do not contain function tags or empty nodes. The other two conversions work better when applied to trees containing this information and so there is an inherent mismatch between gold constituency trees (which contain tags and traces) and constituency parser output (which doesn't). We show that the post-editing method can be used to recover some of this missing information and that it is also effective when used in conjunction with an automatic function labeller.

The paper is organised as follows: we begin by discussing related work in Section 2; Our datasets, parsing systems and labelled dependency schemes are described in Section 3, and the post-editing method itself is described in Section 4. Our experiments with the post-editing method are presented and discussed in Section 5. Finally, Section 6 contains some suggestions for future work.

2 Related Work

Attardi and Ciaramita (2007) and Keith and Novak (2005; 2011) present techniques for automatic correction of dependency trees. The basic idea behind these approaches and the approach described here is the same — correction rules are learned from training data consisting of parser output for which gold standard analysis are available. The difference is that previous techniques learn how to modify the structure of the dependency tree, whereas our technique learns how to modify the labels on individual dependency arcs. The more general idea of statistical post-editing has also been applied to machine translation output (Simard et al., 2007).

Dickinson (2008; 2010) has explored the use of automated techniques to signpost potential anomalies in parse trees by identifying atypical cases in both attachments and labelling. Our method, though originally designed for post-editing, can be also applied similarly to this. That is, the relabelling technique can be used, not only as a post-editing correction step, but also as a type of diagnostic to signal differences between two sets of dependency trees, and hence, potential problems with either parser output or gold standards.

Bryl et al. (2009) presented a way of restoring the missing dependency labels in LFG-based statistical machine translation output. Atomic features of LFG f-structures, such as case, number, etc., were used as features for a Naive Bayes classifier. Though the problem is similar to ours, the approach is not readily reusable for our purpose, because such atomic features (many of which are highly relevant for guessing the correct label) are not used in the kind of parsers we explore in our work.

3 Data and Tools

3.1 Datasets

We employ two datasets in this work, the Wall Street Journal Section of the Penn Treebank (Marcus et al., 1994) and QuestionBank (Judge et al., 2006), a set of 4,000 manually parse-annotated questions from a TREC question answering task.[1] Both datasets con-

[1] Questions occur relatively infrequently in the WSJ dataset (Clark et al., 2004).

tain constituency trees which have been produced by an automatic parser and then corrected by hand. Note that the trees in the *WSJ* dataset contain more information than the trees in *QuestionBank*, namely empty nodes which capture long-distance dependencies and function labels on non-terminal categories.

We use *WSJ22* as our post-editing training/development set and *WSJ23* as our test set. We use sentences 2001-3000 from *QuestionBank* as our post-editing training/development set and sentences 3001-4000 as our test set. For the remainder of the paper, we use the term *QuestionDev* to refer to this development set and the term *QuestionTest* to refer to the test set.

3.2 Parsing Systems

We evaluate the post-editing method using one constituency parser and two dependency parsers, both trained on Sections 2-21 of the WSJ section of the Penn Treebank (Marcus et al., 1994). We prefer the Charniak and Johnson parser mainly because of its accuracy. We employ MaltParser and MSTParser because, although they are not the most accurate dependency parsers available, they are very widely used and they exemplify the two main approaches to statistical dependency parsing, namely, transition-based dependency parsing and maximum-spanning-tree dependency parsing.

The Charniak and Johnson parser The Charniak parser (Charniak, 2000) is a generative constituency parser which uses a head-lexicalised smoothed PCFG which is conditioned on the parse history and whose probability model is fine-tuned for English. In our experiments, we use the reranking version in which the n-best list returned by the generative parser is re-ordered using a discriminative reranker trained on features extracted from the complete trees (Charniak and Johnson, 2005).

MaltParser MaltParser is a multi-lingual transition-based dependency parsing system (Nivre et al., 2006). During training, a classifier learns to predict a parsing action at a particular parsing configuration using information from the parse history and the remaining input string. During parsing, the classifier is used to deterministically construct a dependency tree. For our experiments, we use the *stacklazy* parsing algorithm, which can handle non-projective structures (Nivre et al., 2009). Following Attardi and Ciaramiata (2007) and Zhang and Clark (2008), we train a linear classifier which models interactions between features using feature conjunctions. MaltParser expects POS-tagged input — we use SVMTool (Gimenez and Marquez, 2004) to perform POS tagging.

MSTParser Instead of predicting parsing actions, MSTParser (McDonald et al., 2005; McDonald, 2006) comes from the family of dependency parsers which learn to predict entire dependency trees. The parser finds the maximum spanning tree in a multi-digraph using one of several algorithms described in McDonald (2006). For our experiments, we use the second-order approximate non-projective parsing model introduced in McDonald and Pereira (2006). Labelling is carried out at the same time as the tree structure is predicted.

3.3 Labelled Dependency Schemes

Stanford The Stanford dependency scheme represents parser output as labeled bilexical dependencies, and it has been designed with real-world applications in mind (de Marneffe et al., 2006; de Marneffe and Manning, 2008). Stanford dependencies have been used in a variety of NLP applications including recognising textual entailment, information extraction, biomedical information extraction, sentiment analysis and grammatical error detection. Stanford dependencies can produce dependencies in different formats. In our study we focus on *basic* dependencies, because we want to be able to compare with two other representations both of which assume that representations are trees that include all tokens. Stanford dependencies do not use traces and function tags during the conversion and the resulting trees are projective.

LTH In contrast to the Stanford conversion tool, the LTH tool (Johansson and Nugues, 2007) makes crucial use of function tag and trace information in constituency trees. The resulting dependencies – which were used in the CoNLL 2007 dependency parsing shared task (Nivre et al., 2007) – are designed to be useful in downstream semantic processing. The LTH dependency scheme has the richest set of labels of the representations used in this study and, because it tries to take trace information into account, has a higher proportion of non-projective dependencies. Johansson and Nugues (2007) demonstrate that they are harder for parsers to accurately produce than the simpler conversions previously used by dependency parsers (Yamada and Matsumoto, 2003), but that their use leads to improved performance on the task of semantic role labeling.

LFGDEP Cetinoglu et al. (2010) introduce a dependency scheme that takes as a basis a linguistically motivated Lexical Functional Grammar (LFG) f-structure and changes it so that it is a dependency tree. It uses the LFG Annotation Algorithm (AA) which generates LFG f-structures from Penn Treebank style trees (Cahill et al., 2008). In order to use the output of the AA to train the dependency parser, LFG f-structures are converted to dependency trees. The conversion includes substantial modifications to the f-structure representation, namely, representing each token in the f-structure, removing dependencies that cause multiple heads and avoiding multiple roots. This dependency scheme has a lower number of labels than the Stanford and LHT dependencies. The trees can be non-projective but the proportion of non-projectivity is not as high as LTH.

4 Dependency Label Post-Editing

The new dependency label for the ith arc in a dependency structure, $l_{i,new}$, is predicted as follows:

$$l_{i,new} = \arg\max_{l_{gold}} \hat{P}(l_{i,gold}|f_{i,1}, f_{i,2}, ...)$$

where $l_{i,gold}$ is the gold (correct) dependency label of the ith dependency arc in the structure, $f_{i,1}, f_{i,2}$, etc. are features extracted from the parser output, and $\hat{P}$ is the approximation of the given probability calculated on a training dataset for which gold standard parses are available. If several labels receive equal probability estimates, the “do not change” outcome is given priority. With this method, we make no assumption about feature independence[2] and instead calculate the probability approximation directly:

$$\hat{P}(l_{i,gold}|f_{i,1}, f_{i,2}, ...) = \frac{count(l_{i,gold}, f_{i,1}, f_{i,2}, ...)}{count(f_{i,1}, f_{i,2}, ...)}$$

Only correctly attached (in accordance with the gold standard) dependency arcs are used for training. We additionally request that the denominator of this fraction is not less then 2; in other words, that a decision is made on the basis of at least two relevant samples in the training data. It means, that for some cases no decision is made. This allows us to combine several post-editing transformations in a queue. If, for the given case, a post-editor with a longer feature list refuses to make a decision, another post-editor with a shorter feature list may be given a chance.

We employ a combination of two post-editing transformations, with feature sets as follows (all features are taken from the parser output; so, for example, “the dependency label of the arc in question” is the piece of data which might be replaced as a result of the transformation):[3]

1. **Lexicalised feature set:** the dependency label, the POS tag of the dependent word, and the surface form of the dependent and head words of the arc in question (see left tree in Figure 1)

2. **Non-lexicalised feature set:** the dependency label, the POS tag of the dependent word and the dependency label of the parent dependency arc of the arc in question (see right tree in Figure 1)

5 Experiments

We learn post-editing label modification rules for WSJ and QuestionBank by employing leave-one-out cross-validation using the respective development sets. The resulting rules are then applied to the test sets *WSJ23* and *QuestionTest*. For the WSJ dataset,

[2] In preliminary experiments, Naive Bayes was also tried on the same features (described later in the section) and produced very discouraging results. Together with some correct modifications this method made a huge amount of wrong ones, signalling that Naive Bayesian assumption is too strong for these features and leads to over-generalisation. Therefore, other methods based on the independence assumption are also not promising, though some kind of combined approach may succeed.

[3] We settle on these two feature sets after experimenting on our development sets.

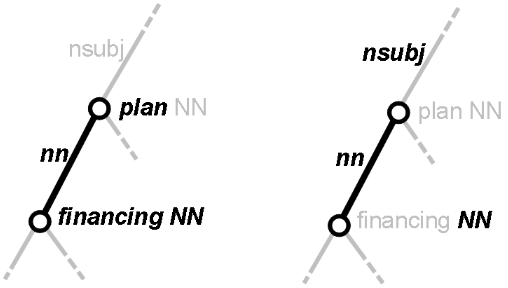

Figure 1: Lexicalised and unlexicalised features sets

we also experiment with using the full parser training data to train the post-editor. For some experiments, we apply an automatic function labeller, FunTag (Chrupała et al., 2007), to the output of the Charniak and Johnson parser, and to the QuestionBank gold trees (which have not been labelled with function tags). We use the CoNLL evaluation metrics of labelled attachment score (LAS) and unlabeled attachment score (UAS).

5.1 WSJ Results

The results for the WSJ dataset are shown in Tables 1 - 3. For each parser type, the baseline scores are provided first, followed by the post-editing scores, where the post-editor is trained using leave-one-out cross-validation on *WSJ22*. The post-editor results when the training set is *WSJ2-21* are given in the third row. The scores are provided both for *WSJ22* and for *WSJ23*. Labeled attachment scores also include the number of correct modifications minus the number of wrong modifications.

We can see from Tables 1 - 3 that LTH benefits the most from post-editing. It is followed by LFGDEP and then Stanford. The reason for these large differences in correction balances between the conversion schemes is due to their design decisions. The parser outputs do not contain function tags and LTH suffers from the lack of this information. LFGDEP is less dependent on them and Stanford is almost insensitive. This explanation is confirmed by using FunTag. When function labels are provided by FunTag, the order of balances remains the same, but the correction balance drops dramatically for LFGDEP and even more for LTH, while the already small correction balances decreases slightly for Stanford dependencies.

For the Stanford scheme, the most successful post-editing rule is the one in which generic `dep` relations are converted to more informative `npadvmod`[4] relations. Using FunTag eliminates the problem almost without a need for post-editing. Training the post-editing tool with a larger data set does not affect the results.

For LTH, relations incorrectly labelled as `VMOD` are converted to various other relations including `ADV`, `SUBJ` and `OBJ`. The correction type breakdown is different for C&J and C&J with FunTag. The `VMOD` corrections appear to cease altogether with FunTag, but actually FunTag only transforms `VMOD` into `DEP` in most of the cases. It still needs to be corrected and it is successfully handled by the post-editing tool. In most frequent sub-cases of `VMOD => SBJ/OBJ` conversions, the post-editing tool converts them to the correct label before using FunTag. When the post-editing tool is trained on *WSJ2-21* instead of *WSJ22*, it makes fewer modifications — the number of incorrect modifications in particular drops, and this explains the increase in correction balance. The type of the corrections is almost the same, but how they are corrected differs. When the post-editor is trained on *WSJ22*, the non-lexicalised feature set is used in modifications. The same modifications are carried out based on the the lexicalised feature set when the size of the training data increases. On *WSJ23*, correct modifications increase, and, more importantly, incorrect modifications drop dramatically. As a result the balance increases by 0.5 % absolute, a statistically significant improvement.

Looking at the breakdown of results in Table 3, we see that, for the LFGDEP dependency scheme, the post-editing rules succeed in correctly converting adjuncts to obliques and complements to adjuncts. Very few instances of these corrections remain after using FunTag. Post-editing corrects only `topicrel => subj` in the C&J FT configuration. This covers sentences with a relative pronoun which acts both as a subject and a relative topic. Due to design decisions (there is only one head of a dependent and a grammatical function has a higher priority than a discourse function), LFGDEP prefers to

[4] noun phrase adverbial modifier

	WSJ 22		WSJ 23	
Parser	**UAS**	**LAS**	**UAS**	**LAS**
C&J	94.18	91.52	94.21	91.76
C&J post-editor-WSJ22	94.18	91.82 (128 - 26 = 102)	94.21	91.94(20 - 9 = 11)
C&J post-editor-WSJ2-21	94.18	91.80 (118 - 21 = 97)	94.21	91.98(20 - 7 = 13)
C&J FT	94.18	91.94	94.21	92.03
C&J FT post-editor-WSJ22	94.18	91.99 (31 - 14 = 17)	94.21	92.06(109 - 20 = 89)
C&J FT post-editor-WSJ2-21	94.18	91.95 (11 - 10 = 1)	94.21	92.06(129 - 17 = 112)
Malt	90.61	87.98	90.28	87.68
Malt post-editor-WSJ22	90.61	87.93 (11 - 26 = -15)	90.28	87.67(15 - 23 = -8)
Malt post-editor-WSJ2-21	90.61	87.95 (12 - 16 = -4)	90.28	87.68(11 - 8 = 3
MST	91.33	88.76	90.74	88.36
MST post-editor-WSJ22	91.33	88.74 (14 - 26 = -12)	90.74	88.35(22 - 27 = -5)
MST post-editor-WSJ2-21	91.33	88.73 (9 - 16 = -7)	90.74	88.35(7 - 10 = -3)

Table 1: Parser accuracy scores for WSJ 22 and WSJ 23 when Stanford Dep. is used

	WSJ 22		WSJ 23	
Parser	**UAS**	**LAS**	**UAS**	**LAS**
C&J	92.21	65.32	91.91	64.31
C&J post-editor-WSJ22	92.21	82.57 (6313 - 25 = 6288)	91.91	81.52(8803 - 18 = 8785)
C&J post-editor-WSJ2-21	92.21	84.54 (7112 - 95 = 7017)	91.91	84.46(10377 - 32 = 10345)
C&J FT	93.99	89.66	93.86	89.82
C&J FT post-editor-WSJ22	93.99	90.87 (530 - 92 = 438)	93.86	90.68(659 - 233 = 426)
C&J FT post-editor-WSJ2-21	93.99	90.89 (483 - 26 = 457)	93.86	91.12(710 - 31 = 679)
Malt	90.84	87.18	90.80	87.58
Malt post-editor-WSJ22	90.84	87.22 (87 - 96 = -9)	90.80	87.31(46 - 209 = -163
Malt post-editor-WSJ2-21	90.84	87.17 (21 - 24 = -3)	90.80	87.61(32 - 15 = 17)
MST	92.24	88.8	91.89	88.9
MST post-editor-WSJ22	92.24	88.81 (78 - 78 = 0)	91.89	88.7(40 - 146 = -106)
MST post-editor-WSJ2-21	92.24	88.77 (8 - 19 = -11)	91.89	88.91(9 - 6 = 3)

Table 2: Parser accuracy scores for WSJ 22 and WSJ 23 when LTH is used

keep the subj relation. Gold trees have the subject information due to traces and coindexation, so LFGDEP correctly picks the subj relation. Parse trees lack this information hence, only topicrel can be assigned. The other remaining correction is subj => adjunct, which highlights a systematic error made by LFGDEP. Using a larger training data does not change the type of modifications and slightly increases the correction balance.

Post-editing does not help the dependency parsers for any of the conversion schemes. A closer look reveals that the kind of errors made by the dependency parsers are not systematic enough to aid the post-editing tool in learning anything with the existing feature sets. Take for instance the non-lexicalised feature set which includes the parent label as a feature: when the C&J parse trees are converted to dependency trees using LDFDEP, there are 3070 labelling errors, and 57% of these have the correct parent. For Malt, there are 2742 labelling errors, but only 26% of them have the correct parent. Therefore, the post-editing training data for Malt contains, not only fewer training instances due to lower attachment accuracy, but also more noise than the corresponding training data for C&J.[5] The same explanation applies to MST.

5.2 QuestionBank Results

The QuestionBank results in Table 4 are interesting because they highlight the different ways the post-editing method can be used. The method works better for QuestionBank than for the WSJ dataset because, for all three parsers, it succeeds in transforming the parser output so that it more closely re-

[5] We experimented with different feature sets on Malt but did not get a significant improvement.

	WSJ 22		WSJ 23	
Parser	**UAS**	**LAS**	**UAS**	**LAS**
C&J	92.22	87.35	91.67	87.61
C&J post-editor-WSJ22	92.22	88.77 (678 - 104 = 574)	91.67	88.48 (691 - 196 = 495)
C&J post-editor-WSJ2-21	92.22	89.44 (978 - 148 = 830)	91.67	89.33 (1190 - 235 = 955)
C&J FT	92.85	90.83	92.49	90.67
C&J FT post-editor-WSJ22	92.85	90.99 (97 - 23 = 74)	92.49	90.71 (87 - 53 = 34)
C&J FT post-editor-WSJ2-21	92.85	91.02 (108 - 14 = 94)	92.49	90.88 (145 - 20 = 125)
Malt	89.20	87.19	89.42	87.55
Malt post-editor-WSJ22	89.20	87.18 (26 - 29 = 3)	89.42	87.45 (14 - 62 = -48)
Malt post-editor-WSJ2-21	89.20	87.19 (15 - 15 = 0)	89.42	87.56 (15 - 11 = 4)
MST	91.02	89.12	90.75	88.94
MST post-editor-WSJ22	91.02	89.11 (20 - 21 = -1)	90.75	88.86 (9 - 56 = -47)
MST post-editor-WSJ2-21	91.02	89.11 (2 - 5 = -3)	90.75	88.94 (4 - 3 = -1)

Table 3: Parser accuracy scores for WSJ 22 and WSJ 23 when LFGDEP is used

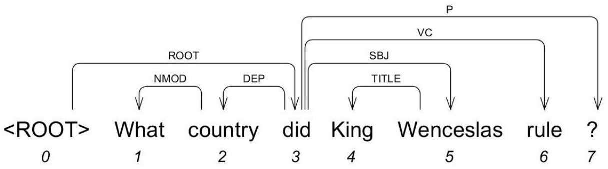

Figure 2: The incorrect gold dependency tree converted by the LTH scheme

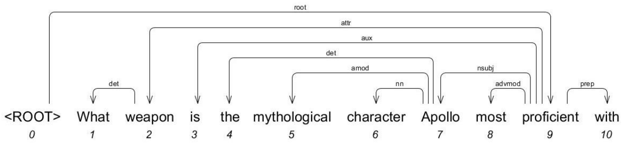
root
attr
aux
det
det
amod
nn
nsubj
advmod
prep
<ROOT> What weapon is the mythological character Apollo most proficient with
0 1 2 3 4 5 6 7 8 9 10

Figure 3: The incorrect gold dependency tree converted by Stanford dependencies

sembles the gold standard. However, we have to be careful here since the QuestionBank gold standard is even less “gold” than the WSJ gold standard for three reasons: 1) it has undergone not one but two automatic procedures, constituency-to-dependency conversion *and* automatic function labelling (recall that the manually annotated QuestionBank constituency trees contain neither functional labels nor traces), 2) no attempt is made to insert traces into the constituency trees before conversion to dependency trees, and 3) the three constituency-to-dependency converters and the automatic function labeller have been developed using Penn Treebank trees and so they are not expected to perform as well on questions. Examining the QuestionBank results in more detail we find problems with the individual converters as well as problems with parser output.

The LTH converter particularly suffers when applied to *QuestionDev*. The most common ”correct” relabelling rules for the two dependency parsers involve a label being converted to the generic `DEP` label. In order to investigate these suspicious relabelling rules, we inspect the gold standard LTH *QuestionDev* dependency trees and find that these dependency trees are in fact incorrect (see, for example, the tree in Figure 2). It is interesting that we discover this problem by looking at the dependency parser relabellings — in this case, the post-editing method is making the dependency parser output worse and this could be because the dependency parsers are trained on dependency trees which were produced from constituency trees containing traces and so their output is more accurate than the QuestionBank gold standard. Examination of the post-editing results highlights a similar (albeit much smaller) problem with the Stanford converter: the correct `cop` dependency label for the copular verb in a question such as *Which X is Y?* is replaced by the incorrect `aux` dependency label because the gold Stanford dependency trees are themselves incorrect. The tree in Figure 3 is an example of an incorrect gold Stanford tree.

There are also many instances in which the gold data is correct and the post-editing method succeeds in correcting labelling errors in parser output. For example, the Stanford relabelling rules manage to

	QuestionDev		QuestionTest	
Parser	**UAS**	**LAS**	**UAS**	**LAS**
C&J	88.47	72.1	88.70	72.46
C&J post-editor-QDev	88.47	81.38 (1017 - 152 = 865)	88.70	81.83 (1041 - 161 = 880)
C&J FT	90.00	82.7	90.43	83.54
C&J FT post-editor-QDev	90.00	85.73 (383 - 109 = 274)	90.43	86.51 (394 - 119 = 275)
Malt	84.89	71.75	85.56	72.61
Malt post-editor-QDev	84.89	78.95 (809 - 155 = 654)	85.56	79.73 (836 - 172 = 664)
MST	85.16	73.06	85.94	74.35
MST post-editor-QDev	85.16	79.52 (751 - 116 = 635)	85.94	71.9 (71 - 297 = -226)

(a) LFGDEP

C&J	82.58	78.40	83.62	79.22
C&J post-editor-QDev	82.58	78.72 (41 - 12 = 29)	83.62	79.47(41 - 16 = 25)
C&J FT	82.58	78.41	83.62	79.26
C&J FT post-editor-QDev	82.58	78.73 (41 - 11 = 30)	83.62	79.5(41 - 16 = 25)
Malt	72.59	67.39	74.10	69
Malt post-editor-QDev	72.59	67.65 (56 - 26 = 30)	74.10	69.45(62 - 17 = 45)
MST	74.75	68.9	76.42	70.59
MST post-editor-QDev	74.75	69.62 (99 - 18 = 81)	76.42	71.17(86 - 25 = 61)

(b) Stanford Dependencies

C&J	90.66	68.47	90.99	69.27
C&J post-editor-QDev	90.66	81.34 (1212 - 5 = 1207)	90.99	81.51(1152 - 3 = 1149)
C&J FT	90.78	84.08	91.21	86.9
C&J FT post-editor-QDev	90.78	86.33 (227 - 22 = 205)	91.21	84.81(223 - 30 = 193)
Malt	85.39	66.96	87.08	68.54
Malt post-editor-QDev	85.39	79.37 (1219 - 88 = 1131)	87.08	80.68(1209 - 89 = 1120)
MST	85.29	68.09	87.03	69.64
MST post-editor-QDev	85.29	79.23 (1133 - 113 = 1020)	87.03	67.63(790 - 1043 = -253)

(c) LTH Conversion

Table 4: Parser accuracy scores for QuestionDev and QuestionTest

correct the mislabelled dependency between the expletive *there* and the main verb in questions such as *How many James Bond novels are there?* from `advmod` to `expl`. An inspection of the LFGDEP rules show that many correct relabellings are from `subj` to `xcomp` and vice versa in questions of the form *What are/is X?*. We have tracked these parser errors back to the question annotation strategy in the Penn Treebank. According to the Penn Treebank bracketing guidelines (Bies et al., 1995), copular verbs are annotated differently to other main verbs in questions in that they do not introduce a VP node (see Figure 4). Judge et al. (2006) comment that this distinction is difficult for parsers to learn. The fact that the relabelling occurs for the dependency parsers (where the conversion is applied to the gold constituency trees before parser training) as well as the constituency parser (where the conversion is applied to the parser output) suggests that this is not a parser-specific problem but rather that the gold standard PTB questions contain some noise.[6]

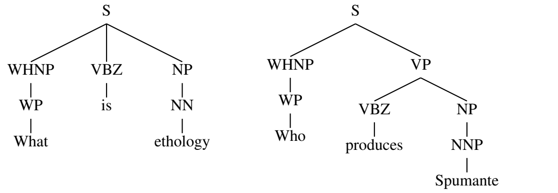

Figure 4: Question Annotation According to PTB Bracketing Guidelines

6 Conclusion

We have presented a technique for modifying the labels in a dependency tree and shown how it

[6] An example is the following tree in *WSJ02*: ((SBARQ (“ “) (WHNP-305 (WP What)) (SQ (NP-SBJ (-NONE- *T*-305)) (**VP** (VBZ is) (NP-PRD (NP (DT the) (NN way)) (ADVP (RB forward))))) (. ?)))

can be used to improve labelled attachment accuracy. We have also demonstrated how the technique can be used to pinpoint problems in automatic constituency-to-dependency converters. The latter use of the technique is important given the absence of a truly gold dependency test set for English.

We have tested our label correction method on three parsers and shown that it has considerably more success on the Charniak and Johnson reranking parser (for which it brought about statistically significant improvements in accuracy) than on MaltParser and MSTParser. Since the Charniak and Johnson parser is a two-stage parser in which attachment mistakes made during the first-stage are corrected during the second, this suggests that the optimal application of our method is after attachment post-editing. We intend to explore this in the future. We also intend to explore the extent to which the method can be improved by taking into account label hierarchies and by imposing global constraints.

References

Giuseppe Attardi and Massimiliano Ciaramita. 2007. Tree revision learning for dependency parsing. In *Proceedings of Human Language Technologies and the Annual Conference of the North American Chapter of the Association for Computational Linguistics(NAACL-HLT)*.

Ann Bies, Mark Ferguson, Karen Katz, and Robert MacIntyre. 1995. Bracketing guidelines for treebank ii style, penn treebank project. Technical Report Tech Report MS-CIS-95-06, University of Pennsylvania, Philadelphia, PA.

Anton Bryl, Josef van Genabith, and Yvette Graham. 2009. Guessing the grammatical function of a non-root f-structure in lfg. In *Proceedings of the Interational Workshop on Parsing Technologies*, Paris, France, October.

Aoife Cahill, Michael Burke, Ruth O'Donovan, Stefan Riezler, Josef van Genabith, and Andy Way. 2008. Wide-coverage deep statistical parsing using automatic dependency structure annotation. *Computational Linguistics*, 34(1):81–124.

Ozlem Cetinoglu, Jennifer Foster, Joakim Nivre, Deirdre Hogan, Aoife Cahill, and Josef van Genabith. 2010. Lfg without c-structures. In *Proceedings of the Ninth International Workshop on Treebanks and Linguistic Theories*, Tartu.

Eugene Charniak and Mark Johnson. 2005. Course-to-fine n-best-parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the ACL*, pages 173–180, Ann Arbor, June.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the Annual Meeting of the North American Association for Computational Linguistics (NAACL-00)*, pages 132–139, Seattle, Washington.

Grzegorz Chrupała, Nicolas Stroppa, Josef van Genabith, and Georgiana Dinu. 2007. Better training for function labeling. In *International Conference on Recent Advances in Natural Language Processing (RANLP 2007)*, pages 133–138, Bulgaria, September.

Stephen Clark, Mark Steedman, and James R. Curran. 2004. Object extraction and question parsing using ccg. In *Proceedings of EMNLP*, pages 111–118, Barcelona.

Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The stanford typed dependencies representation. In *Proceedings of the COLING Workshop on Cross-Framework and Cross-Domain Parser Evaluation*, Manchester, August.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, Genoa, Italy.

Markus Dickinson. 2008. Ad hoc treebank structures. In *Proceedings of the 46th Annual Meeting of the ACL*, Columbus, Ohio, June.

Markus Dickinson. 2010. Detecting errors in automatically-parsed dependency relations. In *Proceedings of the 48th Annual Meeting of the ACL*, Uppsala, Sweden, June.

Jesus Gimenez and Llus Marquez. 2004. Svmtool: A general pos tagger generator based on support vector machines. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04)*, pages 43–46, Lisbon, Portugal.

Keith Hall and Václav Novák. 2005. Corrective modeling for non-projective dependency parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 42–52, Vancouver, British Columbia, October. Association for Computational Linguistics.

Keith Hall and Vaclav Novak, 2011. *Trends in Parsing Technology*, volume 43 of *Text, Speech and Language Technology*, chapter Corrective Dependency Parsing, pages 151–167. Springer.

Richard Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for english. In Joakim Nivre, Heiki-Jaan Kaalep, Kadri Muischnek, and Mare Koit, editors, *Proceedings of NODALIDA 2007*, pages 105–112, Tartu, Estonia.

John Judge, Aoife Cahill, and Josef van Genabith. 2006. Questionbank: Creating a corpus of parse-annotated questions. In *Proceedings of the 21st COLING/44th ACL*.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The penn treebank: Annotating predicate argument structure. In *Proceedings of the 1994 ARPA Speech and Natural Language Workshop*, pages 114–119, Princeton, New Jersey.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL*, pages 81–88.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the ACL*, pages 91–98, Ann Arbor, June.

Ryan McDonald. 2006. *Discriminative Learning and Spanning Tree Algorithms for Dependency Parsing*. Ph.D. thesis, University of Pennsylvania.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2006. Maltparser: A data-driven parser-generator for dependency parsing. In *In Proceedings of the fifth international conference on Language Resources and Evaluation (LREC2006)*, pages 2216–2219.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan Mac Donald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The conll 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932, Prague, Czech Republic, June.

Joakim Nivre, Marco Kuhlmann, and Johan Hall. 2009. An improved oracle for dependency parsing with online reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 73–76, Paris, France, October. Association for Computational Linguistics.

Michel Simard, Cyril Goutte, and Pierre Isabelle. 2007. Statistical phrase-based post-editing. In *Proceedings of HLT-NAACL*, Rochester, NY, April.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT-03)*, pages 195–206, Nancy, France.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 562–571, Honolulu.