Detecting Errors in Automatically-Parsed Dependency Relations

Markus Dickinson
Indiana University
md7@indiana.edu

Abstract

We outline different methods to detect errors in automatically-parsed dependency corpora, by comparing so-called dependency rules to their representation in the training data and flagging anomalous ones. By comparing each new rule to every relevant rule from training, we can identify parts of parse trees which are likely erroneous. Even the relatively simple methods of comparison we propose show promise for speeding up the annotation process.

1 Introduction and Motivation

Given the need for high-quality dependency parses in applications such as statistical machine translation (Xu et al., 2009), natural language generation (Wan et al., 2009), and text summarization evaluation (Owczarzak, 2009), there is a corresponding need for high-quality dependency annotation, for the training and evaluation of dependency parsers (Buchholz and Marsi, 2006). Furthermore, parsing accuracy degrades unless sufficient amounts of labeled training data from the same domain are available (e.g., Gildea, 2001; Sekine, 1997), and thus we need larger and more varied annotated treebanks, covering a wide range of domains. However, there is a bottleneck in obtaining annotation, due to the need for manual intervention in annotating a treebank. One approach is to develop automatically-parsed corpora (van Noord and Bouma, 2009), but a natural disadvantage with such data is that it contains parsing errors. Identifying the most problematic parses for human post-processing could combine the benefits of automatic and manual annotation, by allowing a human annotator to efficiently correct automatic errors. We thus set out in this paper to detect errors in automatically-parsed data.

If annotated corpora are to grow in scale and retain a high quality, annotation errors which arise from automatic processing must be minimized, as errors have a negative impact on training and evaluation of NLP technology (see discussion and references in Boyd et al., 2008, sec. 1). There is work on detecting errors in dependency corpus annotation (Boyd et al., 2008), but this is based on finding inconsistencies in annotation for identical recurring strings. This emphasis on identical strings can result in high precision, but many strings do not recur, negatively impacting the recall of error detection. Furthermore, since the same strings often receive the same automatic parse, the types of inconsistencies detected are likely to have resulted from manual annotation. While we can build from the insight that simple methods can provide reliable annotation checks, we need an approach which relies on more general properties of the dependency structures, in order to develop techniques which work for automatically-parsed corpora.

Developing techniques to detect errors in parses in a way which is independent of corpus and parser has fairly broad implications. By using only the information available in a training corpus, the methods we explore are applicable to annotation error detection for either hand-annotated or automatically-parsed corpora and can also provide insights for parse reranking (e.g., Hall and Novák, 2005) or parse revision (Attardi and Ciaramita, 2007). Although we focus only on detecting errors in automatically-parsed data, similar techniques have been applied for hand-annotated data (Dickinson, 2008; Dickinson and Foster, 2009).

Our general approach is based on extracting a grammar from an annotated corpus and comparing *dependency rules* in a new (automatically-annotated) corpus to the grammar. Roughly speaking, if a dependency rule—which represents all the dependents of a head together (see section 3.1)—does not fit well with the grammar, it is flagged as potentially erroneous. The methods do not have to be retrained for a given parser's output (e.g., Campbell and Johnson, 2002), but work by comparing any tree to what is in the training grammar (cf. also approaches stacking hand-written rules on top of other parsers (Bick, 2007)).

We propose to flag erroneous parse rules, using information which reflects different grammatical properties: POS lookup, bigram information, and full rule comparisons. We build on a method to detect so-called ad hoc rules, as described in section 2, and then turn to the main approaches in section 3. After a discussion of a simple way to flag POS anomalies in section 4, we evaluate the different methods in section 5, using the outputs from two different parsers. The methodology proposed in this paper is easy to implement and independent of corpus, language, or parser.

2 Approach

We take as a starting point two methods for detecting *ad hoc* rules in constituency annotation (Dickinson, 2008). Ad hoc rules are CFG productions extracted from a treebank which are "used for specific constructions and unlikely to be used again," indicating annotation errors and rules for ungrammaticalities (see also Dickinson and Foster, 2009).

Each method compares a given CFG rule to all the rules in a treebank grammar. Based on the number of similar rules, a score is assigned, and rules with the lowest scores are flagged as potentially ad hoc. This procedure is applicable whether the rules in question are from a new data set—as in this paper, where parses are compared to a training data grammar—or drawn from the treebank grammar itself (i.e., an internal consistency check).

The two methods differ in how the comparisons are done. First, the *bigram method* abstracts a rule to its bigrams. Thus, a rule such as NP → JJ NN provides support for NP → DT JJ JJ NN, in that it shares the JJ NN sequence. By contrast, in the other method, which we call the *whole rule method*,[1] a rule is compared in its totality to the grammar rules, using Levenshtein distance. There is no abstraction, meaning all elements are present—e.g., NP → DT JJ JJ NN is very similar to NP → DT JJ NN because the sequences differ by only one category.

While previously used for constituencies, what is at issue is simply the *valency* of a rule, where by valency we refer to a head and its entire set of arguments and adjuncts (cf. Przepiórkowski, 2006)—that is, a head and all its dependents. The methods work because we expect there to be regularities in valency structure in a treebank grammar; non-conformity to such regularities indicates a potential problem.

[1]This is referred to *whole daughters* in Dickinson (2008), but the meaning of "daughters" is less clear for dependencies.

3 Ad hoc rule detection

3.1 An appropriate representation

To capture valency, consider the dependency tree from the Talbanken05 corpus (Nilsson and Hall, 2005) in figure 1, for the Swedish sentence in (1), which has four dependency pairs.[2]

(1)

Det	går	bara	inte	ihop	.
it	goes	just	not	together	

'It just doesn't add up.'

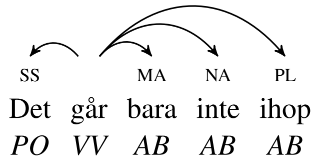


Figure 1: Dependency graph example

On a par with constituency rules, we define a grammar *rule* as a dependency relation rewriting as a head with its sequence of POS/dependent pairs (cf. Kuhlmann and Satta, 2009), as in figure 2. This representation supports the detection of idiosyncracies in valency.[3]

1. TOP → root ROOT:VV
2. ROOT → SS:PO VV MA:AB NA:AB PL:AB
3. SS → PO
4. MA → AB
5. NA → AB
6. PL → AB

Figure 2: Rule representation for (1)

For example, for the ROOT category, the head is a verb (VV), and it has 4 dependents. The extent to which this rule is odd depends upon whether comparable rules—i.e., other ROOT rules or other VV rules (see section 3.2)—have a similar set of dependents. While many of the other rules seem rather spare, they provide useful information, showing categories which have no dependents. With a TOP rule, we have a rule for every

[2]Category definitions are in appendix A.

[3]Valency is difficult to define for coordination and is specific to an annotation scheme. We leave this for the future.

head, including the virtual root. Thus, we can find anomalous rules such as TOP → root ROOT:AV ROOT:NN, where multiple categories have been parsed as ROOT.

3.2 Making appropriate comparisons

In comparing rules, we are trying to find evidence that a particular (parsed) rule is valid by examining the evidence from the (training) grammar.

Units of comparison To determine similarity, one can compare dependency relations, POS tags, or both. Valency refers to both properties, e.g., verbs which allow verbal (POS) subjects (dependency). Thus, we use the pairs of dependency relations and POS tags as the units of comparison.

Flagging individual elements Previous work scored only entire rules, but some dependencies are problematic and others are not. Thus, our methods score individual elements of a rule.

Comparable rules We do not want to compare a rule to all grammar rules, only to those which *should* have the same valents. Comparability could be defined in terms of a rule's dependency relation (LHS) or in terms of its head. Consider the four different object (OO) rules in (2). These vary a great deal, and much of the variability comes from the fact that they are headed by different POS categories, which tend to have different selectional properties. The head POS thus seems to be predictive of a rule's valency.

(2) a. OO → **PO**
 b. OO → DT:EN AT:AJ **NN** ET:VV
 c. OO → SS:PO **QV** VG:VV
 d. OO → DT:PO AT:AJ **VN**

But we might lose information by ignoring rules with the same left-hand side (LHS). Our approach is thus to take the greater value of scores when comparing to rules *either* with the same dependency relation or with the same head. A rule has multiple chances to prove its value, and low scores will only be for rules without any type of support.

Taking these points together, for a given rule of interest r, we assign a score (S) to each element e_i in r, where $r = e_1...e_m$ by taking the maximum of scores for rules with the same head (h) or same LHS (lhs), as in (3). For the first element in (2b), for example, S(DT:EN) $=$ $\max\{s$(DT:EN, NN), s(DT:EN, OO)$\}$. The question is now how we define $s(e_i, c)$ for the comparable element c.

(3) $S(e_i) = \max\{s(e_i, h), s(e_i, lhs)\}$

3.3 Whole rule anomalies

3.3.1 Motivation

The whole rule method compares a list of a rule's dependents to rules in a database, and then flags rule elements without much support. By using all dependents as a basis for comparison, this method detects improper dependencies (e.g., an adverb modifying a noun), dependencies in the wrong overall location of a rule (e.g., an adverb before an object), and rules with unnecessarily long argument structures. For example, in (4), we have an improper relation between *skall* ('shall') and *sambeskattas* ('be taxed together'), as in figure 3. It is parsed as an adverb (AA), whereas it should be a verb group (VG). The rule for this part of the tree is +F → ++:++ SV AA:VV, and the AA:VV position will be low-scoring because the ++:++ SV context does not support it.

(4)

Makars	övriga	inkomster	är	B-inkomster
spouses'	other	incomes	are	B-incomes

och	*skall*	som	tidigare	*sambeskattas*	.
and	shall	as	previously	be taxed togeher	.

'The other incomes of spouses are B-incomes and shall, as previously, be taxed together.'

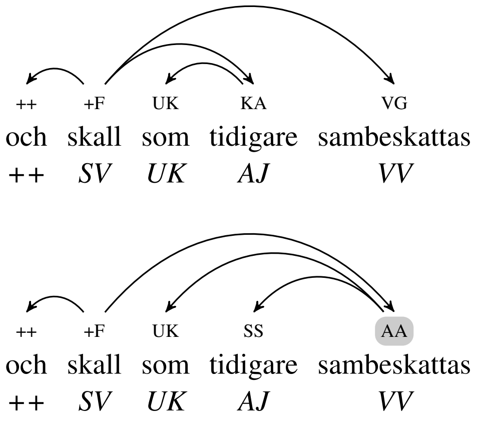


Figure 3: Wrong label (top=gold, bottom=parsed)

3.3.2 Implementation

The method we use to determine similarity arises from considering what a rule is like without a problematic element. Consider +F → ++:++ SV AA:VV from figure 3, where AA should be a different category (VG). The rule without this error, +F → ++:++ SV, starts several rules in the training data, including some with VG:VV as the next item. The subrule ++:++ SV seems to be reliable, whereas the subrules containing AA:VV (++:++ AA:VV and SV AA:VV) are less reliable. We thus determine reliability by seeing how often each subsequence occurs in the training rule set.

Throughout this paper, we use the term *subrule* to refer to a rule subsequence which is exactly one element shorter than the rule it is a component of. We examine subrules, counting their frequency *as subrules*, not as complete rules. For example, TOP rules with more than one dependent are problematic, e.g., TOP $\rightarrow$ root ROOT:AV ROOT:NN. Correspondingly, there are no rules with *three* elements containing the subrule root ROOT:AV.

We formalize this by setting the score $s(e_i, c)$ equal to the summation of the frequencies of all comparable subrules containing e_i from the training data, as in (5), where B is the set of subrules of r with length one less.

(5) $s(e_i, c) = \sum_{sub \in B : e_i \in sub} C(sub, c)$

For example, with c = +F, the frequency of +F $\rightarrow$ ++:++ SV as a subrule is added to the scores for ++:++ and SV. In this case, +F $\rightarrow$ ++:++ SV **VG:BV**, +F $\rightarrow$ ++:++ SV **VG:AV**, and +F $\rightarrow$ ++:++ SV **VG:VV** all add support for +F $\rightarrow$ ++:++ SV being a legitimate subrule. Thus, ++:++ and SV are less likely to be the sources of any problems. Since +F $\rightarrow$ SV AA:VV and +F $\rightarrow$ ++:++ AA:VV have very little support in the training data, AA:VV receives a low score.

Note that the subrule count $C(sub, c)$ is different than counting the number of rules containing a subrule, as can be seen with identical elements. For example, for SS $\rightarrow$ VN ET:PR ET:PR, C(VN ET:PR, SS) $= 2$, in keeping with the fact that there are 2 pieces of evidence for its legitimacy.

3.4 Bigram anomalies

3.4.1 Motivation

The bigram method examines relationships between adjacent sisters, complementing the whole rule method by focusing on local properties. For (6), for example, we find the gold and parsed trees in figure 4. For the long parsed rule TA $\rightarrow$ PR HD:ID HD:ID **IR:IR AN:RO JR:IR**, all elements get low whole rule scores, i.e., are flagged as potentially erroneous. But only the final elements have anomalous bigrams: HD:ID IR:IR, IR:IR AN:RO, and AN:RO JR:IR all never occur.

(6)

När	det	gäller	inkomståret	1971	(
when	it	concerns	the income year	1971	(

taxeringsåret	1972	)	skall	barnet	...
assessment year	1972	)	shall	the child	...

'Concerning the income year of 1971 (assessment year 1972), the child . . . '

3.4.2 Implementation

To obtain a bigram score for an element, we simply add together the bigrams which contain the element in question, as in (7).

(7) $s(e_i, c) = C(e_{i-1}\mathbf{e_i}, c) + C(\mathbf{e_i}e_{i+1}, c)$

Consider the rule from figure 4. With $c = TA$, the bigram HD:ID IR:IR never occurs, so both HD:ID and IR:IR get 0 added to their score. HD:ID **HD:ID**, however, is a frequent bigram, so it adds weight to HD:ID, i.e., positive evidence comes from the bigram on the left. If we look at IR:IR, on the other hand, **IR:IR** AN:RO occurs 0 times, and so IR:IR gets a total score of 0.

Both scoring methods treat each element independently. Every single element could be given a low score, even though once one is corrected, another would have a higher score. Future work can examine factoring in all elements at once.

4 Additional information

The methods presented so far have limited definitions of comparability. As using complementary information has been useful in, e.g., POS error detection (Loftsson, 2009), we explore other simple comparable properties of a dependency grammar. Namely, we include: a) frequency information of an overall dependency rule and b) information on how likely each dependent is to be in a relation with its head, described next.

4.1 Including POS information

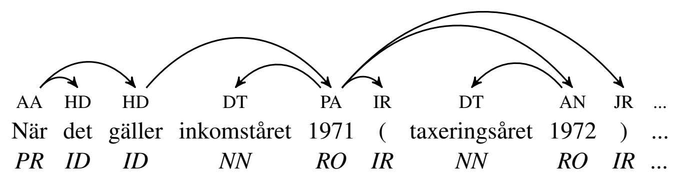
Consider PA $\rightarrow$ SS:NN XX:XX HV OO:VN, as illustrated in figure 5 for the sentence in (8). This rule is entirely correct, yet the XX:XX position has low whole rule and bigram scores.

(8)

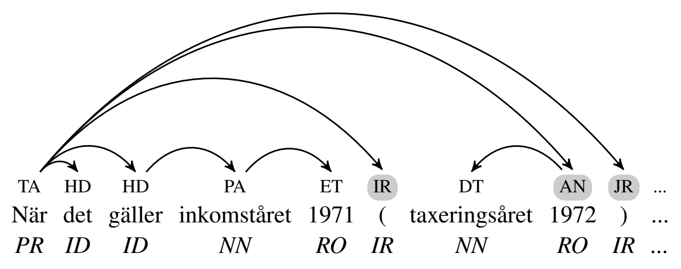
Uppgift	om	vilka	orter	som
information	of	which	neighborhood	who

har	utkörning	finner	Ni	också	i	...
has	delivery	find	you	also	in	...

'You can also find information about which neighborhoods have delivery services in . . . '



AA HD HD DT PA IR DT AN JR ...
När det gäller inkomståret 1971 (taxeringsåret 1972) ...
PR ID ID NN RO IR NN RO IR ...



TA HD HD PA ET IR DT AN JR ...
När det gäller inkomståret 1971 (taxeringsåret 1972) ...
PR ID ID NN RO IR NN RO IR ...

Figure 4: A rule with extra dependents (top=gold, bottom=parsed)

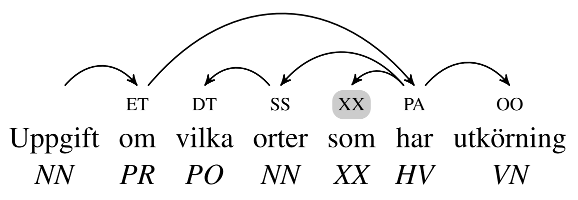


Figure 5: Overflagging (gold=parsed)

One method which does not have this problem of overflagging uses a "lexicon" of POS tag pairs, examining relations between POS, irrespective of position. We extract POS pairs, note their dependency relation, and add a L/R to the label to indicate which is the head (Boyd et al., 2008). Additionally, we note how often two POS categories occur as a non-depenency, using the label NIL, to help determine whether there should be any attachment. We generate NILs by enumerating all POS pairs in a sentence. For example, from figure 5, the parsed POS pairs include NN PR $\mapsto$ ET-L, NN PO $\mapsto$ NIL, etc.

We convert the frequencies to probabilities. For example, of 4 total occurrences of XX HV in the training data, 2 are XX-R (cf. figure 5). A probability of 0.5 is quite high, given that NILs are often the most frequent label for POS pairs.

5 Evaluation

In evaluating the methods, our main question is: how accurate are the dependencies, in terms of both attachment and labeling? We therefore currently examine the scores for elements functioning as dependents in a rule. In figure 5, for example, for *har* ('has'), we look at its score within ET $\rightarrow$ PR **PA:HV** and not when it functions as a head, as in PA $\rightarrow$ SS:NN XX:XX **HV** OO:VN.

Relatedly, for each method, we are interested in whether elements with scores below a threshold have worse attachment accuracy than scores above, as we predict they do. We can measure this by scoring each testing data position below the threshold as a 1 if it has the correct head and dependency relation and a 0 otherwise. These are simply labeled attachment scores (LAS). Scoring separately for positions above and below a threshold views the task as one of sorting parser output into two bins, those more or less likely to be correctly parsed. For development, we also report unlabeled attachement scores (UAS).

Since the goal is to speed up the post-editing of corpus data by flagging erroneous rules, we also report the precision and recall for error detection. We count either attachment or labeling errors as an error, and precision and recall are measured with respect to how many errors are found below the threshold. For development, we use two F-scores to provide a measure of the settings to examine across language, corpus, and parser conditions: the balanced F_1 measure and the $F_{0.5}$ measure, weighing precision twice as much. Precision is likely more important in this context, so as to prevent annotators from sorting through too many false positives. In practice, one way to use these methods is to start with the lowest thresholds and work upwards until there are too many non-errors.

To establish a basis for comparison, we compare method performance to a parser on its own.[4] By examining the parser output without any automatic assistance, how often does a correction need to be made?

5.1 The data

All our data comes from the CoNLL-X Shared Task (Buchholz and Marsi, 2006), specifically the 4 data sets freely available online. We use the Swedish Talbanken data (Nilsson and Hall, 2005) and the transition-based dependency parser MaltParser (Nivre et al., 2007), with the default settings, for developing the method. To test across languages and corpora, we use MaltParser on the other 3 corpora: the Danish DDT (Kromann, 2003), Dutch Alpino (van der Beek et al., 2002), and Portuguese Bosque data (Afonso et al., 2002). Then, we present results using the graph-based parser MSTParser (McDonald and Pereira, 2006), again with default settings, to test the methods across parsers. We use the gold standard POS tags for all experiments.

5.2 Development data

In the first line of table 1, we report the baseline MaltParser accuracies on the Swedish test data, including baseline error detection precision (=1-LAS_b), recall, and (the best) F-scores. In the rest of table 1, we report the best-performing results for each of the methods,[5] providing the number of rules below and above a particular threshold, along with corresponding UAS and LAS values. To get the raw number of identified rules, multiply the number of corpus position below a threshold (b) times the error detection precision (P). For example, the bigram method with a threshold of 39 leads to finding 283 errors (455 $\times$ .622).

Dependency elements with frequency below the lowest threshold have lower attachment scores (66.6% vs. 90.1% LAS), showing that simply using a complete rule helps sort dependencies. However, frequency thresholds have fairly low precision, i.e., 33.4% at their best. The whole rule and bigram methods reveal greater precision in identifying problematic dependencies, isolating elements with lower UAS and LAS scores than with frequency, along with corresponding greater precision and F-scores. The bigram method is more fine-grained, identifying small numbers of rule elements at each threshold, resulting in high error detection precision. With a threshold of 39, for example, we find over a quarter of the parser errors with 62% precision, from this one piece of information. For POS information, we flag 23.6% of the cases with over 60% precision (at 81.6).

Taking all these results together, we can begin to sort more reliable from less reliable dependency tree elements, using very simple information. Additionally, these methods naturally group cases together by linguistic properties (e.g., adverbial-verb dependencies within a particualr context), allowing a human to uncover the principle behind parse failure and ajudicate similar cases at the same time (cf. Wallis, 2003).

5.3 Discussion

Examining some of the output from the Talbanken test data by hand, we find that a prominent cause of false positives, i.e., correctly-parsed cases with low scores, stems from low-frequency dependency-POS label pairs. If the dependency rarely occurs in the training data with the particular POS, then it receives a low score, regardless of its context. For example, the parsed rule TA $\rightarrow$ **IG:IG** RO has a correct dependency relation (IG) between the POS tags IG and its head RO, yet is assigned a whole rule score of 2 and a bigram score of 20. It turns out that IG:IG only occurs 144 times in the training data, and in 11 of those cases (7.6%) it appears immediately before RO. One might consider normalizing the scores based on overall frequency or adjusting the scores to account for other dependency rules in the sentence: in this case, there may be no better attachment.

Other false positives are correctly-parsed elements that are a part of erroneous rules. For instance, in AA $\rightarrow$ UK:UK SS:PO TA:AJ AV SP:AJ OA:PR **+F:HV** +F:HV, the first +F:HV is correct, yet given a low score (0 whole rule, 1 bigram). The following and erroneous +F:HV is similarly given a low score. As above, such cases might be handled by looking for attachments in other rules (cf. Attardi and Ciaramita, 2007), but these cases should be relatively unproblematic for hand-correction, given the neighboring error.

We also examined false negatives, i.e., errors with high scores. There are many examples of PR PA:NN rules, for instance, with the NN improp-

[4]One may also use parser confidence or parser revision methods as a basis of comparison, but we are aware of no systematic evaluation of these approaches for detecting errors.

[5]Freq=rule frequency, WR=whole rule, Bi=bigram, POS=POS-based (POS scores multiplied by 10,000)

Score	Thr.	b	a	UAS$_b$	LAS$_b$	UAS$_a$	LAS$_a$	P	R	F$_1$	F$_{0.5}$
None	n/a	5656	0	87.4%	82.0%	0%	0%	18.0%	100%	30.5%	21.5%
Freq	0	1951	3705	76.6%	66.6%	93.1%	90.1%	33.4%	64.1%	**43.9%**	**36.9%**
WR	0	894	4762	64.7%	54.0%	91.7%	87.3%	46.0%	40.5%	43.0%	**44.8%**
	6	1478	4178	71.1%	60.9%	93.2%	89.5%	39.1%	56.9%	**46.4%**	41.7%
Bi	0	56	5600	10.7%	7.1%	88.2%	82.8%	92.9%	5.1%	9.7%	21.0%
	39	455	5201	51.6%	37.8%	90.6%	85.9%	62.2%	27.9%	38.5%	**49.9%**
	431	1685	3971	74.1%	63.7%	93.1%	89.8%	36.3%	60.1%	**45.2%**	39.4%
POS	0	54	5602	27.8%	22.2%	87.4%	82.6%	77.8%	4.1%	7.9%	17.0%
	81.6	388	5268	48.5%	38.4%	90.3%	85.3%	61.6%	23.5%	34.0%	**46.5%**
	763	1863	3793	75.4%	65.8%	93.3%	90.0%	34.2%	62.8%	**44.3%**	37.7%

Table 1: MaltParser results for Talbanken, for select values (b = below, a = above threshold (Thr.))

erly attached, but there are also many correct instances of PR PA:NN. To sort out the errors, one needs to look at lexical knowledge and/or other dependencies in the tree. With so little context, frequent rules with only one dependent are not prime candidates for our methods of error detection.

5.4 Other corpora

We now turn to the parsed data from three other corpora. The Alpino and Bosque corpora are approximately the same size as Talbanken, so we use the same thresholds for them. The DDT data is approximately half the size; to adjust, we simply halve the scores. In tables 2, 3, and 4, we present the results, using the best F$_{0.5}$ and F$_1$ settings from development. At a glance, we observe that the best method differs for each corpus and depending on an emphasis of precision or recall, with the bigram method generally having high precision.

Score	Thr.	b	LAS$_b$	LAS$_a$	P	R
None	n/a	5585	73.8%	0%	26.2%	100%
Freq	0	1174	43.2%	81.9%	56.8%	45.6%
WR	0	483	32.5%	77.7%	67.5%	22.3%
	6	787	39.4%	79.4%	60.6%	32.6%
Bi	39	253	33.6%	75.7%	66.4%	11.5%
	431	845	45.6%	78.8%	54.4%	31.4%
POS	81.6	317	51.7%	75.1%	48.3%	10.5%
	763	1767	53.5%	83.2%	46.5%	56.1%

Table 2: MaltParser results for Alpino

For Alpino, error detection is better with frequency than, for example, bigram scores. This is likely due to the fact that Alpino has the smallest label set of any of the corpora, with only 24 dependency labels and 12 POS tags (cf. 64 and 41 in Talbanken, respectively). With a smaller label set, there are less possible bigrams that could be anomalous, but more reliable statistics about a whole rule. Likewise, with fewer possible POS tag pairs, Alpino has lower precision for the low-threshold POS scores than the other corpora.

Score	Thr.	b	LAS$_b$	LAS$_a$	P	R
None	n/a	5867	82.2%	0%	17.8%	100%
Freq	0	1561	61.2%	89.9%	38.8%	58.1%
WR	0	693	48.1%	86.8%	51.9%	34.5%
	6	1074	54.4%	88.5%	45.6%	47.0%
Bi	39	227	15.4%	84.9%	84.6%	18.4%
	431	776	51.0%	87.0%	49.0%	36.5%
POS	81.6	369	33.3%	85.5%	66.7%	23.6%
	763	1681	60.1%	91.1%	39.9%	64.3%

Table 3: MaltParser results for Bosque

Score	Thr.	b	LAS$_b$	LAS$_a$	P	R
None	n/a	5852	81.0%	0%	19.0%	100%
Freq	0	1835	65.9%	88.0%	34.1%	56.4%
WR	0	739	53.9%	85.0%	46.1%	30.7%
	3	1109	60.1%	85.9%	39.9%	39.9%
Bi	19.5	185	25.4%	82.9%	74.6%	12.4%
	215.5	884	56.8%	85.4%	43.2%	34.4%
POS	40.8	179	30.2%	82.7%	69.8%	11.3%
	381.5	1214	62.5%	85.9%	37.5%	41.0%

Table 4: MaltParser results for DDT

For the whole rule scores, the DDT data is worse (compare its 46.1% precision with Bosque's 45.6%, with vastly different recall values), which could be due to the smaller training data. One might also consider the qualitative differences in the dependency inventory of DDT compared to the others—e.g., appositions, distinctions in names, and more types of modifiers.

5.5 MSTParser

Turning to the results of running the methods on the output of MSTParser, we find similar but slightly worse values for the whole rule and bigram methods, as shown in tables 5-8. What is most striking are the differences in the POS-based method for Bosque and DDT (tables 7 and 8), where a large percentage of the test corpus is underneath the threshold. MSTParser is apparently positing fewer distinct head-dependent pairs, as most of them fall under the given thresholds. With the exception of the POS-based method for DDT (where LAS_b is actually higher than LAS_a) the different methods seem to be accurate enough to be used as part of corpus post-editing.

Score	Thr.	b	LAS_b	LAS_a	P	R
None	n/a	5656	81.1%	0%	18.9%	100%
Freq	0	3659	65.2%	89.7%	34.8%	64.9%
WR	0	4740	55.7%	86.0%	44.3%	37.9%
	6	4217	59.9%	88.3%	40.1%	53.9%
Bi	39	5183	38.9%	84.9%	61.1%	27.0%
	431	3997	63.2%	88.5%	36.8%	57.1%
POS	81.6	327	42.8%	83.4%	57.2%	17.5%
	763	1764	68.0%	87.0%	32.0%	52.7%

Table 5: MSTParser results for Talbanken

Score	Thr.	b	LAS_b	LAS_a	P	R
None	n/a	5585	75.4%	0%	24.6%	100%
Freq	0	1371	49.5%	83.9%	50.5%	50.5%
WR	0	453	40.0%	78.5%	60.0%	19.8%
	6	685	45.4%	79.6%	54.6%	27.2%
Bi	39	226	39.8%	76.9%	60.2%	9.9%
	431	745	48.2%	79.6%	51.8%	28.1%
POS	81.6	570	60.4%	77.1%	39.6%	16.5%
	763	1860	61.9%	82.1%	38.1%	51.6%

Table 6: MSTParser results for Alpino

Score	Thr.	b	LAS_b	LAS_a	P	R
None	n/a	5867	82.5%	0%	17.5%	100%
Freq	0	1562	63.9%	89.3%	36.1%	55.0%
WR	0	540	50.6%	85.8%	49.4%	26.0%
	6	985	58.0%	87.5%	42.0%	40.4%
Bi	39	117	34.2%	83.5%	65.8%	7.5%
	431	736	56.4%	86.3%	43.6%	31.3%
POS	81.6	2978	75.8%	89.4%	24.2%	70.3%
	763	3618	74.3%	95.8%	25.7%	90.7%

Table 7: MSTParser results for Bosque

Score	Thr.	b	LAS_b	LAS_a	P	R
None	n/a	5852	82.9%	0%	17.1%	100%
Freq	0	1864	70.3%	88.8%	29.7%	55.3%
WR	0	624	60.6%	85.6%	39.4%	24.6%
	3	1019	65.4%	86.6%	34.6%	35.3%
Bi	19.5	168	28.6%	84.5%	71.4%	12.0%
	215.5	839	61.6%	86.5%	38.4%	32.2%
POS	40.8	5714	83.0%	79.0%	17.0%	97.1%
	381.5	5757	82.9%	80.0%	17.1%	98.1%

Table 8: MSTParser results for DDT

6 Summary and Outlook

We have proposed different methods for flagging the errors in automatically-parsed corpora, by treating the problem as one of looking for anomalous rules with respect to a treebank grammar. The different methods incorporate differing types and amounts of information, notably comparisons among dependency rules and bigrams within such rules. Using these methods, we demonstrated success in sorting well-formed output from erroneous output across language, corpora, and parsers.

Given that the rule representations and comparison methods use both POS and dependency information, a next step in evaluating and improving the methods is to examine automatically POS-tagged data. Our methods should be able to find POS errors in addition to dependency errors. Furthermore, although we have indicated that differences in accuracy can be linked to differences in the granularity and particular distinctions of the annotation scheme, it is still an open question as to which methods work best for which schemes and for which constructions (e.g., coordination).

Acknowledgments

Thanks to Sandra Kübler and Amber Smith for comments on an earlier draft; Yvonne Samuelsson for help with the Swedish translations; the IU Computational Linguistics discussion group for feedback; and Julia Hockenmaier, Chris Brew, and Rebecca Hwa for discussion on the general topic.

A Some Talbanken05 categories

POS tags	
++	coord. conj.
AB	adverb
AJ	adjective
AV	*vara* (be)
EN	indef. article
HV	*ha(va)* (have)
ID	part of idiom
IG	punctuation
IR	parenthesis
NN	noun
PO	pronoun
PR	preposition
RO	numeral
QV	*kunna* (can)
SV	*skola* (will)
UK	sub. conj.
VN	verbal noun
VV	verb
XX	unclassifiable

Dependencies	
++	coord. conj.
+F	main clause coord.
AA	adverbial
AN	apposition
AT	nomainl pre-modifier
DT	determiner
ET	nominal post-modifier
HD	head
IG	punctuation
IR	parenthesis
JR	second parenthesis
KA	comparative adverbial
MA	attitude adverbial
NA	negation adverbial
OO	object
PA	preposition comp.
PL	verb particle
SS	subject
TA	time adverbial
UK	sub. conj.
VG	verb group
XX	unclassifiable

References

Afonso, Susana, Eckhard Bick, Renato Haber and Diana Santos (2002). Floresta Sintá(c)tica: a treebank for Portuguese. In *Proceedings of LREC 2002*. Las Palmas, pp. 1698–1703.

Attardi, Giuseppe and Massimiliano Ciaramita (2007). Tree Revision Learning for Dependency Parsing. In *Proceedings of NAACL-HLT-07*. Rochester, NY, pp. 388–395.

Bick, Eckhard (2007). Hybrid Ways to Improve Domain Independence in an ML Dependency Parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*. Prague, Czech Republic, pp. 1119–1123.

Boyd, Adriane, Markus Dickinson and Detmar Meurers (2008). On Detecting Errors in Dependency Treebanks. *Research on Language and Computation* 6(2), 113–137.

Buchholz, Sabine and Erwin Marsi (2006). CoNLL-X Shared Task on Multilingual Dependency Parsing. In *Proceedings of CoNLL-X*. New York City, pp. 149–164.

Campbell, David and Stephen Johnson (2002). A transformational-based learner for dependency grammars in discharge summaries. In *Proceedings of the ACL-02 Workshop on Natural Language Processing in the Biomedical Domain*. Phildadelphia, pp. 37–44.

Dickinson, Markus (2008). Ad Hoc Treebank Structures. In *Proceedings of ACL-08*. Columbus, OH.

Dickinson, Markus and Jennifer Foster (2009). Similarity Rules! Exploring Methods for Ad-Hoc Rule Detection. In *Proceedings of TLT-7*. Groningen, The Netherlands.

Gildea, Daniel (2001). Corpus Variation and Parser Performance. In *Proceedings of EMNLP-01*. Pittsburgh, PA.

Hall, Keith and Václav Novák (2005). Corrective Modeling for Non-Projective Dependency Parsing. In *Proceedings of IWPT-05*. Vancouver, pp. 42–52.

Kromann, Matthias Trautner (2003). The Danish Dependency Treebank and the underlying linguistic theory. In *Proceedings of TLT-03*.

Kuhlmann, Marco and Giorgio Satta (2009). Treebank Grammar Techniques for Non-Projective Dependency Parsing. In *Proceedings of EACL-09*. Athens, Greece, pp. 478–486.

Loftsson, Hrafn (2009). Correcting a POS-Tagged Corpus Using Three Complementary Methods. In *Proceedings of EACL-09*. Athens, Greece, pp. 523–531.

McDonald, Ryan and Fernando Pereira (2006). Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL-06*. Trento.

Nilsson, Jens and Johan Hall (2005). *Reconstruction of the Swedish Treebank Talbanken*. MSI report 05067, Växjö University: School of Mathematics and Systems Engineering.

Nivre, Joakim, Johan Hall, Jens Nilsson, Atanas Chanev, Gulsen Eryigit, Sandra Kübler, Svetoslav Marinov and Erwin Marsi (2007). MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering* 13(2), 95–135.

Owczarzak, Karolina (2009). DEPEVAL(summ): Dependency-based Evaluation for Automatic Summaries. In *Proceedings of ACL-AFNLP-09*. Suntec, Singapore, pp. 190–198.

Przepiórkowski, Adam (2006). What to acquire from corpora in automatic valence acquisition. In Violetta Koseska-Toszewa and Roman Roszko (eds.), *Semantyka a konfrontacja jezykowa, tom 3*, Warsaw: Slawistyczny Ośrodek Wydawniczy PAN, pp. 25–41.

Sekine, Satoshi (1997). The Domain Dependence of Parsing. In *Proceedings of ANLP-96*. Washington, DC.

van der Beek, Leonoor, Gosse Bouma, Robert Malouf and Gertjan van Noord (2002). The Alpino Dependency Treebank. In *Proceedings of CLIN 2001*. Rodopi.

van Noord, Gertjan and Gosse Bouma (2009). Parsed Corpora for Linguistics. In *Proceedings of the EACL 2009 Workshop on the Interaction between Linguistics and Computational Linguistics: Virtuous, Vicious or Vacuous?*. Athens, pp. 33–39.

Wallis, Sean (2003). Completing Parsed Corpora. In Anne Abeillé (ed.), *Treebanks: Building and using syntactically annoted corpora*, Dordrecht: Kluwer Academic Publishers, pp. 61–71.

Wan, Stephen, Mark Dras, Robert Dale and Cécile Paris (2009). Improving Grammaticality in Statistical Sentence Generation: Introducing a Dependency Spanning Tree Algorithm with an Argument Satisfaction Model. In *Proceedings of EACL-09*. Athens, Greece, pp. 852–860.

Xu, Peng, Jaeho Kang, Michael Ringgaard and Franz Och (2009). Using a Dependency Parser to Improve SMT for Subject-Object-Verb Languages. In *Proceedings of NAACL-HLT-09*. Boulder, Colorado, pp. 245–253.