Detecting Dependency Parse Errors with Minimal Resources

Markus Dickinson and **Amber Smith**
Indiana University
Bloomington, IN USA
{md7,smithamj}@indiana.edu

Abstract

To detect errors in automatically-obtained dependency parses, we take a grammar-based approach. In particular, we develop methods that incorporate n-grams of different lengths and use information about possible parse revisions. Using our methods allows annotators to focus on problematic parses, with the potential to find over half the parse errors by examining only 20% of the data, as we demonstrate. A key result is that methods using a small gold grammar outperform methods using much larger grammars containing noise. To perform annotation error detection on newly-parsed data, one only needs a small grammar.

1 Introduction and Motivation

There is a need for high-quality dependency annotation for the training and evaluation of dependency parsers (Buchholz and Marsi, 2006), ideally large amounts of annotated data. This is a lofty goal for any language, especially languages with few, if any, annotated resources. Citing Abeillé (2003), Hwa et al. (2005) say: "it appears that acquiring 20,000-40,000 sentences — including the work of building style guides, redundant manual annotation for quality checking, and so forth – can take from four to seven years." As pointed out by Dickinson (2010), a major bottleneck in obtaining annotation involves the need for human correction, leading to the following process: 1) automatically parse corpora (van Noord and Bouma, 2009), which will contain errors, and 2) identify problematic parses for human post-processing. We develop this second step of detecting errors.

In particular, there is the problem of having little annotated data to work with, as in the cases of: lesser-resourced languages (e.g., Ambati et al., 2010; Simpson et al., 2009), new annotation schemes, and new domains with limited in-domain annotated data (e.g., Plank and van Noord, 2010). In these situations, there is a large cost to annotate data, and parsing results are worse than in cases with more annotated training data (Nivre, 2010).

We develop error detection methods based on a coarse grammar, comparing parsed rules to rules in a grammar in order to identify anomalies, as outlined in section 2. This is akin to theoretically-driven work in treebanking, where a grammar is used to guide treebank annotation (e.g., Oepen et al., 2004; Rosén et al., 2005; Bond et al., 2004), but it shares an insight with work incorporating grammatical information to improve parsing, namely that even simple grammatical information can inform parse output (e.g., Plank and van Noord, 2010; Ambati, 2010; Seeker et al., 2010).

Our methods are simple and efficient, requiring no additional parsing technology. This is especially beneficial for lesser-resourced languages, and, as we describe in section 3, makes the methods applicable to any treebanking scenario. Also, we want to know "which linguistic constructs are hard to analyze" (Goldberg and Elhadad, 2010a). Framing parse failures in terms of grammatical anomalies makes them easily interpretable, leading to quicker annotation decisions, as systematic problems can be seen at a glance (cf. Wallis, 2003; Hara et al., 2009) and perhaps also helping unearth latent theoretical decisions (cf., e.g., Leech, 2004).

We improve upon previous methods in two ways, as described in section 4. First, we streamline the different sources of information by adding the counts for all n-grams within a rule: this balances concerns over sparse data for longer n-grams with the fact that longer n-grams are more informative. Secondly, taking the scores initially assigned by our methods, we compare them with scores for possible parse revisions. This checks whether the parser could have made a better decision and more directly connects to parse revision work (e.g., Attardi and Ciaramita, 2007). As we

Proceedings of the 12th International Conference on Parsing Technologies, pages 241–252,
October 5-7, 2011, Dublin City University.

show in section 5, these methods have the potential to help annotation quality go from, e.g., 65% accuracy to 85% by presenting annotators with cases highly likely to be erroneous. Based on the results using a noisy grammar in section 6, we also conclude that a small gold grammar is more effective than a large noisy one, a useful result since such resources can quickly be developed manually.

2 Error Detection

We build from methods for detecting *ad hoc*, or anomalous, rules in dependency parses (Dickinson, 2010). Dependency *rules* represent a head with its arguments and adjuncts, and ad hoc rules are "used for specific constructions and unlikely to be used again," indicating annotation errors and rules for ungrammaticalities (Dickinson, 2011).

To understand a dependency rule, consider figure 1 from the Talbanken05 corpus (Nilsson and Hall, 2005), for the Swedish sentence in (1).[1]

(1) Det går bara inte ihop .
it goes just not together
'It just doesn't add up.'

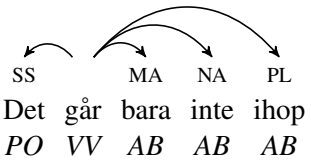

Figure 1: Dependency graph example

A grammar *rule* is comprised of a dependency relation rewriting as a head with its sequence of POS-dependent pairs, as in figure 2.

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)

The error detection methods work by comparing an individual parsed rule to rules in a (training) grammar. Based on comparisons to similar rules, a score is assigned to each individual element of a rule, and elements with the lowest scores are flagged. The intuition is that there should be regularities in dependency structure; non-conformity to regularities indicates a potential problem.

Dickinson (2010) compares pairs of dependency relations and POS tags (instead of using only, e.g., dependencies), and we follow suit. Relatedly, although scores can be obtained for each unit in its role as a dependent or as a head, we score elements based on how they function as dependents (see also section 2.1). In figure 2, for instance, the PO position is scored with respect to its role in rule #2, where it is a dependent, and not rule #3, where it is a head.

As Dickinson (2010) says, "We do not want to compare a rule to all grammar rules, only to those which *should* have the same valents." For a given parse rule, we can compare it to rules with the same head or rules which have the same "mother" (left-hand side (LHS)) dependency relation. We follow Dickinson (2010) in comparing to rules *either* with the same LHS or with the same head, taking the greater value of scores; this gives a rule more chances to prove its validity. The formula is given in (2), where e_i refers to the i^{th} element in a rule ($r = e_1 ... e_m$) and the score is based on having the same head (h) or the same LHS (lhs).

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

Most importantly, there is the method of comparison itself. Dickinson (2010) explores the *bigram* and *whole rule* methods. To see how these work, consider the bigram method for the rule in (3), with implicit START and END tags. To score AT:AJ, the bigram method counts up how often the bigrams DT:PO AT:AJ and AT:AJ VN appear in all OO (LHS) or VN (head) rules in the grammar. The whole rule method works similarly, but comparing full subsequences. We develop more general methods in section 4, with the same rough idea: rules with low scores are likely errors.

(3) OO → DT:PO **AT:AJ** VN

The formula for the bigram method is given in (4), where c = comparable item (head or LHS), and $C(x, c)$ refers to the count of x, c in the grammar. Which grammar we use is discussed in section 3.

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

2.1 Unary Rules

Consider again the "unary" rules in the grammar, as in SS → PO in figure 2. Dickinson (2010) included these rules, as this captures the fact that,

[1] Category definitions are in appendix A.

e.g., PO has no dependents. However, for the way rules are scored, we can remove these unary rules.

Because unary rules only contain heads, only items which are the heads in rules are actually affected by such rules. For example, with SS → PO, only PO-headed rules will be affected. But, as mentioned above, we score elements by how they function *as dependents*. We can thus ignore such rules. Removing them from a grammar also gives a better indication of the size of a rule set.

3 Scenarios

As there are different scenarios for building annotated corpora—corpora for new languages, corpora of a much larger nature than previously created, corpora in a new domain, etc.—the assumptions of one's resources can be quite different. We sketch different possibilities here.

3.1 Types of Grammars

Our methods are based on having a grammar to compare parse rules to. Automatically-parsed data can be obtained in different ways, affecting grammar resources. First, parsed data can be obtained from a parser trained on gold data. In this case, a GOLD GRAMMAR can be extracted from the gold data. For error detection, we can then compare the parsed data to this fixed grammar (section 5).

One subtype of this is the situation where the gold grammar is small (SMALL GRAMMAR). To be in this situation, one could develop a small treebank and extract a grammar from it, or one could manually write (coarse) grammar rules. We work with corpus-extracted grammars, but one advantage of the type of approach we take—unlike more theoretically grammar-driven ways of treebanking (e.g., Oepen et al., 2004; Rosén et al., 2005; Bond et al., 2004)—is that grammar-writing is simpler, using only very coarse categories.

The second type of situation (NOISY GRAMMAR) arises when not all the rules in the grammar are valid, as when rules are extracted straight from automatically-parsed data. If the data has been parsed, some parser exists, but it may be the case that: a) data is exchanged, but the technology is not; or b) the data is partially hand-corrected. In either case, a NOISY GRAMMAR can be extracted from the parsed data (section 6).

This leads to the possibility of hybrid grammars, where some rules have been hand-checked and others have not—i.e., a concatenation of a gold and a noisy grammar. Since gold and noisy grammars are more primary, we focus on them.

Finally, there may be a hand-crafted or specialized parser, tuned to a particular annotation scheme. This can arise from continual development in a large project (e.g., the Alpino project (Plank and van Noord, 2010)) or when one uses a parser without having access to the corpus it was trained on. If the parser has an accessible grammar, there is a GOLD GRAMMAR; otherwise, a NOISY GRAMMAR can be extracted.

3.2 Grammar-Based Error Detection

The reason we phrase error detection in terms of grammars is that some grammar is always available, whereas we cannot always assume a modifiable parser. Error detection based on coarse grammars is applicable to any of these scenarios, as opposed to, e.g., methods which rely on details of how a parser is likely to fail (e.g., Attardi and Ciaramita, 2007; Goldberg and Elhadad, 2010b).[2] Additionally, because we can always obtain a grammar from parsed data, we will have access to the frequency of occurrence of each rule.[3]

4 Methods of Comparison

In this section, we develop the best methods of rule comparison (section 4.1) and introduce a new, orthogonal way of flagging possible errors (section 4.2). In order to compare directly to the previous work, we use the Swedish Talbanken corpus (Nilsson and Hall, 2005) with the same data split as in the CoNLL-X Shared Task (Buchholz and Marsi, 2006); in section 5 and beyond, we switch the training and testing data. In all experiments, we use gold standard POS tags.

To keep the data sets clear across different training regiments, we refer to them as the *large* and *small* Talbanken data sets. The large Talbanken data has 11,042 sentences, 191,467 words, 96,517 (non-unary) rule tokens and 26,292 rule types. The small data set has 389 sentences, 5,656 words, 3,107 rule tokens and 1,284 rule types. As we show in section 5, even such a small grammar can be highly effective in detecting parse errors.

[2]Those methods are of course well-suited to the issue of improving parsing technology.

[3]Even for hand-written grammars, the methods we develop can be altered to be type-based instead of token-based.

4.1 Method Improvement

As discussed in section 2, the main previous methods of flagging errors look for anomalous bigrams or anomalous "whole rules." Each method has its limitations, looking only at local context (bigram) or only at the whole context (whole rule). Yet there is a straightforward, principled way to combine the methods: add together all n-gram parts of a rule (cf. Zhao et al., 2010; Bergsma et al., 2009).

To do this, during training we break a rule down into its component n-grams (cf. steps 1 and 2 below) and store the frequency of the rule for each n-gram. For rule scoring, we then:

1. Add START and END context tags to the list of elements in a rule.
2. Calculate all (contiguous) bigrams, trigrams, etc., up to the length of the whole rule.
3. Calculate the frequency of all n-grams (based on a training set) containing a given element.

This is encapsulated in the formula in (5).

$$(5)\quad s(e_i, c) = \sum_{ngram : e_i \in ngram \wedge n \geq 2} C(ngram, c)$$

For example, focusing on AT:AJ back in rule (3) with the head VN as the comparable item (c), we count up the grammar frequencies of n-grams, as in (6). A benefit of this method is that, by using local and global contexts, equally-frequent bigrams, for example, are sorted out by whether they continue to occur in longer forms.

(6) s(AT:AJ, VN) = C(DT:PO **AT:AJ**, VN)
+ C(**AT:AJ** VN, VN)
+ C(START DT:PO **AT:AJ**, VN)
+ C(DT:PO **AT:AJ** VN, VN)
+ C(**AT:AJ** VN END, VN)
+ C(START DT:PO **AT:AJ** VN, VN)
+ C(DT:PO **AT:AJ** VN END, VN)
+ C(START DT:PO **AT:AJ** VN END, VN)

We refer to the new method as the *all-gram* (*All.*) method, and results are reported in table 1 using MaltParser (Nivre et al., 2007). Our goal is to improve annotation post-editing, and so we report precision (P) and recall (R) of error detection, for positions below the threshold. Either an attachment or labeling error counts as an error. Precision is thus the complement of a labeled attachment score (LAS); for example, by using no score, the parser has an LAS of 82.0% and error detection precision of 18.0%. Two F-scores are provided—F_1 and $F_{0.5}$, which weights precision twice as much—and the best value of each is reported. We favor precision, so as to prevent annotators from sorting through false positives.

Score	Thr.	pos.	P	R	F_1	$F_{0.5}$
None	n/a	5,656	18.0%	100%	30.5%	21.5%
All.	0	56	92.9%	5.1%	9.7%	21.0%
	60	479	61.8%	29.1%	39.6%	50.5%
	390	1,234	42.5%	51.7%	46.7%	44.1%
Tri.	0	215	77.2%	16.3%	27.0%	44.2%
	5	478	66.3%	31.2%	42.4%	54.1%
	49	1,202	44.1%	52.2%	47.8%	45.5%
High.	0	215	77.2%	16.3%	27.0%	44.2%
	5	424	69.6%	29.0%	41.0%	**54.4%**
	90	1,373	42.2%	57.0%	**48.5%**	44.5%

Table 1: Talbanken error detection results for different scores: parser = MaltParser trained on large data; grammar = large gold; evaluation data = small data (*pos.* = positions below threshold (*Thr.*))

Comparing the results here on the same data in Dickinson (2010), we have a slightly higher F_1 score than the previous best method (46.7% vs. 46.4% [whole rule]) and a higher $F_{0.5}$ than the best method (50.5% vs. 49.9% [bigram]).

4.1.1 Excluding Bigrams

The methods are still very close, so we attempt to improve further. Consider the tree in figure 3, for the example in (7), where there is no verb, and a preposition (PR) is the sentence's root. The resulting rule, which occurs 79 times in the training data, is TOP → *root* ROOT:PR.

(7) Ur giftermålsbalken 5 kap < 1 och < 2
From marriage act 5 ch. < 1 and < 2
'From the marriage act, ch. 5, para. (?) 1 and 2'

When bigram information is included in calculating scores, both *root* ROOT:PR and ROOT:PR END get counted 79 times, leading to high scores for both PRs in (8), when only the first is correct. Bigrams do not distinguish the correct first PR from the incorrect second PR attachment in (8): both have 79 pieces of supporting evidence.

(8) TOP → *root* ROOT:PR ROOT:PR

We need both left and right contexts in rule scoring. We thus experiment with using both trigram information as a model of its own (*Tri.*) and a model which uses all trigrams and above (i.e., the

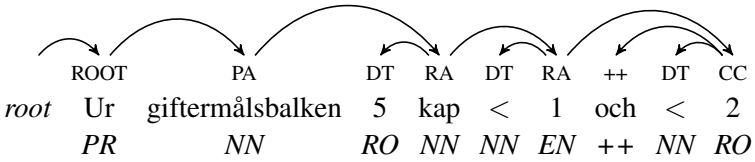

Figure 3: Preposition-rooted sentence (from large Talbanken data set)

same as (5), but with $n \geq 3$), which we refer to as the *high-gram* method (*High.*). These are included in table 1, where we can see improvements in precision and recall. For example, with a threshold of 5, we find 29% of all errors by examining 424 instances (i.e., 7.5% of the parsed corpus).

Trigrams are the minimum length to obtain contextual information on each side, and longer n-grams (for the high-gram method) give more precise indication that the rule context is valid. Even though long n-grams are rare, they are useful, leading to a positive, albeit small, improvement.

4.1.2 Notes on Evaluation

The intended use of the scores is for annotators to start with the lowest-scoring positions (i.e., tokens) and work upwards. The F-measures give some sense of this, but we need a way to evaluate across different corpora. To account for this, we do two things: first, we report the values for the lowest threshold, so that one can get a sense of the precision for the highest-priority cases. Secondly, we use Talbanken for development, finding the best $F_{0.5}$ score and calculating the percentage of the test corpus that the threshold identifies. In the case of evaluating on the small test corpus, $\frac{424}{5{,}656} = 7.5\%$, so other experiments with a similar large/small split would be evaluated with a threshold identifying as close to 7.5% of the evaluation corpus as possible. We reset thresholds for experiments where the large corpus is the test corpus, as these behave differently, due to differing parser quality; in section 5.1, we will set this at 23%. The bottom two lines of table 7, for instance, present results for the lowest threhold (0) and the one which identifies as close to 23% of the tokens as possible (25). Also, as we emphasize precision, in future tables we report $F_{0.5}$ and ignore F_1.

4.2 Revision Checking

The n-gram models discussed so far are not very sophisticated. Low scores conflate two issues: 1) the element in question is anomalous, or 2) there was no better attachment or labeling for this element, i.e., the parser could not have made a better decision. Identifying the latter cases could reduce false positives, i.e., correct low-scoring positions.

For example, for (9), the parser correctly attaches *hållas* ('hold') to *måste* ('must') in a VG relation, as in figure 4. The high-gram scoring method assigns a score of 0, because UK:UK MV VG:VV and MV VG:VV END were never observed in the grammar, but there is no better attachment in this instance. By determining that attaching to *som* or *hemligt* is no better, we can overcome some limitations of the original scoring. We refer to the process of checking for a better-scoring attachment or labeling as a *revision check*.

(9) Du kommer under din utbildning att få
You come under your education to get
se och höra sådant som måste hållas
see and hear such as must be held
hemligt .
secret .
'You will during your education see and hear things that must be kept secret.'

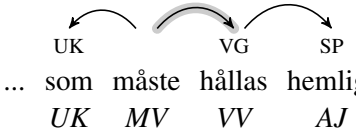

Figure 4: Correct low-scoring dependency structure

What a better-scoring revision means depends to a great extent on parser quality. Trained on a small gold corpus, the parser has not seen much data. If other attachments or labelings result in better scores, the parser may not have had enough information to select the revision and perhaps should have given it more weight. For a parser with high accuracy (e.g., trained on large gold data), on the other hand, the parser should have considered the revision and rejected it on the basis of good evidence. The elements which do not have

a better revision are ones which may represent new types of constructions. Thus, the parser may do worse if there are no reasonable revisions—just the opposite of the expectation for a small corpus.

4.2.1 Revision Checking Algorithm

For an element e_i in a rule r ($lhs \rightarrow e_1 e_2, ..., e_n$), where e_i is a (pos_i, dep_i) pair:

1. Score e_i, e_{i-1} (left context), and e_{i+1} (right context) within r, using, e.g., the high-gram method.
2. Check a different labeling in the original rule.
   (a) Replace (pos_i, dep_i) with all relevant (pos_i, dep_j) pairs (where $dep_i \neq dep_j$)
      - **Relevant** dependency labels occurred in training with pos_i; other labels will result in a score of 0.
   (b) Score the modified rule.
   (c) Flag the rule if a different labeling results in an **improvement**, namely:
      i. The modified element's score increases, and left/right context scores improve or stay the same.
      ii. The modified element's score stays the same, and context scores increase.
3. Check different re-attachments (and re-labelings) by placing the element in question (e_i) into other rules (x), if appropriate.
   (a) Remove e_i from the original rule:
      i. Remove e_i from r to obtain r'.
      ii. Score r' and extract scores for positions to the left/right of where e_i used to be.
      iii. If e_{i-1} or e_{i+1} is worse than originally scored, skip next step and do not flag.
   (b) Modify and score other rules (x):
      i. Insert element into the appropriate position within x, following linear ordering.
      ii. Determine whether this is an appropriate **candidate attachment site**:
         A. Inserting e_i does not introduce a cyclic structure.
         B. Inserting e_i does not create a non-projective structure.
      iii. Try all **relevant** dependency labelings for this element to obtain different x'.
      iv. Score each modified rule x', in particular e', e'_{-1}, and e'_{+1}.
      v. **Flag** the original rule if a different atttachment+labeling results in an **improvement** (as defined above).

While the idea is straightforward, there are several points to note. First, the output is that we flag an element if *any* revision shows improvement, but do not keep track of which revision is better. Secondly, in step #3bii we use a strict notion of where an element may be inserted—ensuring no cyclicity or non-projectivity—but either of these conditions may be relaxed. Further, we do not check rootedness, as we do not enforce a globally optimal tree. Finally, the algorithm can check any element in any order; if one were to perform rule revision, the ordering of elements would matter, as scores would change based on previous revisions.

4.2.2 Using the Flagged Elements

Once rules are flagged as having a better-scoring revision or not, we can use: a) the original (high-gram) score, and b) the revision check flag (yes = a better revision exists). These can be used independently or in conjunction. We will evaluate by reporting error detection precision and recall for the lowest-scoring flagged and unflagged positions, as well as for all flagged and unflagged positions. We can also view the process as: correct all the flagged positions and then move to the unflagged ones, and so we report combinations of all flagged positions plus some amount of unflagged positions.

In table 2, we report results using the large gold grammar on the small parsed data. We note first that the highest precision is recorded for the lowest unflagged position (82.8% vs. 77.2% without the flagged/unflagged distinction). As mentioned, when the parser is high quality, the cases where we cannot find a better attachment or labeling may be the most difficult. This gives us some promise of being able to use these techniques for active learning (e.g., Sassano and Kurohashi, 2010). Secondly, the F-scores are lower than the best ones obtained on this data ignoring the revision flagging. Thus, when the parser is of a high quality, we want to prioritize the initial high-gram scores over a revision check. As we will see in section 5.2, smaller grammars present a different picture.[4]

5 Small Gold Grammars

Turning to a small gold grammar, we perform two sets of experiments: in the first, we use MaltParser[5] on the Swedish Talbanken data (sections 5.1 and 5.2). Then, in section 5.3, we use MSTParser (McDonald et al., 2006)[6] on the

[4] Determining at what point one switches from "low quality" to "high quality" is a question for future research.

[5] http://maltparser.org/

[6] http://sourceforge.net/projects/mstparser/

Score	Thr.	pos.	P	R	$F_{0.5}$
L	0	215	77.2%	16.3%	44.2%
LF	0	70	65.7%	4.5%	17.7%
AF	1,269	670	43.1%	28.4%	39.1%
LU	0	145	**82.8%**	11.8%	37.6%
AU	31,987	4,986	14.6%	71.6%	17.3%
AF+U0	n/a	815	50.2%	40.3%	47.8%
AF+U5	n/a	931	51.5%	47.1%	**50.5%**

Table 2: Revision checking: same set-up as in table 1, using high-gram method for checking. (L=lowest, A=all, F=flagged, U=unflagged, Ux=unflagged up to threshold x)

dependency-converted English Wall Street Journal (WSJ) data (Marcus et al., 1993),[7] and we vary the size and appropriateness of the gold grammar. With these permutations, we demonstrate the wide applicability of our methods. With little annotated training data, parser results are worse than before, placing more priority on error detection.

5.1 Working with a Small Gold Grammar

Swapping the Talbanken data, the training data is now approximately 10% of the corpus (small data) and the test data 90% (large data). The results are given in table 3, where the first thing to note is the overall worse parser performance, reflected in a much higher baseline precision (*None*) of 35.4% (=64.6% LAS), significantly higher then with the normal split (P=18.0%, LAS=82.0%, table 1).

Score	Thr.	pos.	P	R	$F_{0.5}$
None	n/a	191,467	35.4%	100%	40.7%
Freq.	0	116,847	48.1%	82.7%	52.5%
Bi.	0	21,110	85.2%	26.5%	59.0%
	6	48,890	70.0%	50.4%	65.0%
Tri.	0	44,297	72.7%	47.5%	**65.7%**
High.	0	44,297	72.7%	47.5%	**65.7%**
All.	0	21,110	85.2%	26.5%	59.0%
	9	48,690	70.3%	50.4%	65.2%

Table 3: Talbanken error detection results for different scores: parser = MaltParser trained on small data; grammar = small gold; evaluation data = large data

With low parser accuracy, baseline F measures are higher: by going over every instance, an annotator will, in principle, find every error (100% recall), with 35% precision. However, that means going through over 190,000 words by hand. To improve the corpus with less effort, we want to identify errors with high precision. Indeed, in the results in table 3, we find precision around 70% with recall around 50%.[8]

Consider the high-gram method, which has the highest $F_{0.5}$, 65.7%, at a score of 0—i.e., no rules in the grammar with any trigrams or longer n-grams supporting a given rule.[9] First, this is much higher than the 54.4% for the standard data split (table 1), partly due to the fact that the parser accuracy is lower. Secondly, 44,297 positions are flagged—23% of the corpus—with 32,209 erroneous; annotators could correct nearly 3 out of every 4 flagged dependencies, fixing 47% of the errors. We will use 23% of the corpus for other experiments with large data (cf. section 4.1.2). Finally, if 32,209 corrections are added to the original 123,595 correct positions, the resulting corpus will have correct dependencies 81.4% of the time (vs. 64.6% LAS), making the corpus more suitable for, e.g., training parsers (cf. Nivre, 2010).

Note also that at the lowest threshold, the bigram method is the most precise (85.2%). What we observed before (cf. example (8) in section 4.1.1) is true—positive evidence of bigrams may be misleading—but *negative* evidence from bigrams may be useful (cf. Dickinson, 2011). If a position has no bigram support, this is worse than no trigrams. One can thus consider splitting the zero high-gram elements into two classes: those which never occurred as bigrams (more likely to be erroneous) and those which did.

To gauge an upper bound on error detection, we use the large gold grammar for an oracle experiment. This helps sort out the effect of the grammar (and its size) from the effect of the comparison methods. We can see the results in table 4.

In comparing to table 3, we see that the results with the much larger grammar are a few points better. The best $F_{0.5}$ measure goes from 65.7% to 69.6%, despite the grammar sizes being 1,284 and 26,292 rule types, respectively (section 4). Even a small gold grammar is extremely useful for error detection. Furthermore, the small grammar here is based on data which is disjoint from the evaluation data, whereas the large grammar comes from (gold) annotation for the evaluation data.

[7] We used the LTH Constituent-to-Dependency Conversion Tool (Johansson and Nugues, 2007), selecting options for CoNLL 2007 format and no NP rebracketing.

[8] Since errors are often inter-related (Hara et al., 2009), it is likely an annotator would actually have a higher recall, but we do not explore inter-relatedness here.

[9] The trigram method performs identically here because no trigrams means no higher n-grams.

Score	Thr.	pos.	P	R	$F_{0.5}$
None	n/a	191,467	35.4%	100%	40.7%
Freq.	0	84,067	57.5%	71.2%	59.8%
Bi.	84	33,197	80.5%	39.2%	66.6%
Tri.	18	39,707	78.8%	46.1%	69.0%
High.	23	39,278	79.8%	46.2%	**69.6%**
All.	203	39,846	77.0%	45.2%	67.5%

Table 4: Talbanken oracle error detection results: same set-up as table 3, but grammar = large gold

5.2 Small Gold Grammar Revision Checking

Turning to flagging positions by whether they have better-scoring possible revisions, we report the results for Talbanken in table 5. With a previous best $F_{0.5}$ of 65.7% for all zero-scoring element, we observe improvement here (68.2%), obtained by going over all (21,670) flagged positions and then including the (34,343) lowest unflagged positions.

Score	Thr.	pos.	P	R	$F_{0.5}$
L	0	44,297	72.7%	47.5%	65.7%
LF	0	9,954	**73.8%**	10.8%	34.1%
AF	72	21,670	69.1%	22.0%	48.4%
LU	0	34,343	72.4%	36.6%	60.5%
AU	1233	169,797	31.2%	77.9%	35.4%
AF+U0	n/a	56,013	71.1%	58.7%	**68.2%**

Table 5: Revision checking: same set-up as in table 3, using high-gram method for checking

This matches the intuition that, when the quality of the parse is low, such a sanity check can improve parse error detection. Note in this case that 69.1% of all flagged (AF) cases need revision, not much lower than the overall (lowest) precision of 72.7%. No matter the rule score, if it is flagged, then it is quite likely to be in need of revision. To evaluate in other situations with large evaluation corpora, we will take all flagged positions and then the lowest-scoring unflagged positions.

Consider the practical effect: 56,013 positions are identified, with 39,825 (71.1%) erroneous. Fixing these cases would correct 58.7% of the errors, resulting in 85.4% corpus accuracy ($\frac{163,420}{191,467}$).

False Negatives We investigate some of the false negatives—i.e., cases which the revision check does not flag but which are low-scoring—in order to discover the limitations of the method. In general, the underflagging often results from the conservative nature of candidate selection. Because we do not allow cycles to be introduced, for example, it is nearly impossible to revise a ROOT element. The most frequent false negatives are those with multiple ROOTs, e.g., the erroneous IP position in TOP → root ROOT:NN ROOT:IP (not flagged 82 times). Extremely long rules (e.g., over 10 dependents) are also underflagged. Because we do not permit non-projectivity, if an element is between two sister elements, those are often the *only* positions which can serve as alternate heads; otherwise, branches have to cross. This is schematized in figure 5, where revisions for D could only include C and E as heads.

Figure 5: Schematic for sister relations

5.3 Grammar Size and Quality

As with the Talbanken data, we want to test these methods on English data using a small amount of training data and a large amount of test data. To do this, we train different parser models. We use the default training options for MSTParser and train one model on section 00 of the WSJ data, one on section 24, and one on both. All three models are then used to parse sections 02-21, with results in table 6. We use the default settings, as this is a good fit for using the methods in the real world.[10]

Par.	Tokens	Size	UAS	LAS
00	46,451	7,250	81.7%	73.4%
24	32,853	5,797	80.5%	71.8%
0024	79,304	11,095	83.5%	76.0%

Table 6: Parser accuracies for WSJ models with different training data (*Par.*), including number of training *Tokens* and *Size* of grammar (non-unary rule types)

In addition to varying the parser, we also vary the grammar, running tests for each model using gold standard grammars derived from different sections. This set-up allows us to see the effect of varying the size of the parser training data, the size of the grammar, and the different grammars across data sets—i.e., error detection using a grammar from a different section than the one the parser is trained on. The results are in table 7.

[10] In development, we also tested parser models with the optimized settings used for McDonald et al. (2005), but there was little impact on accuracy: 73.7%, 72.1%, and 76.2% LAS for the respective training data situations in table 6.

Par.	Gram.	Thr.	pos.	P	R	$F_{0.5}$
00	None	n/a	950,022	26.6%	100%	31.2%
	00	0	143,365	79.5%	45.1%	**68.9%**
		9	217,804	70.8%	61.0%	68.6%
	24	0	152,281	78.0%	47.0%	**68.9%**
		6	219,712	69.4%	60.2%	67.3%
	0024	0	130,724	**81.0%**	41.9%	68.3%
		16	217,628	70.7%	60.9%	68.5%
24	None	n/a	950,022	28.2%	100%	32.9%
	00	0	155,889	79.7%	46.4%	69.7%
		6	218,227	73.4%	59.8%	**70.2%**
	24	0	157,733	79.1%	46.6%	69.4%
		5	217,751	72.6%	59.0%	69.4%
	0024	0	139,604	**81.3%**	42.3%	68.6%
		12	218,207	73.3%	59.7%	70.1%
0024	None	n/a	950,022	24.0%	100%	28.3%
	00	0	127,667	77.7%	43.6%	**67.2%**
		14	219,362	64.6%	62.3%	64.1%
	24	0	133,596	76.3%	44.8%	66.9%
		9	216,826	64.0%	60.9%	63.3%
	0024	0	114,486	**79.7%**	40.1%	66.5%
		25	218,731	64.4%	61.9%	63.9%

Table 7: WSJ: high-gram scores for lowest and 23% thresholds: parser = default MST trained on different data (*Par.*); grammar = gold from section listed (*Gram.*); evaluation data = sections 02-21

Looking at the results for the parser trained on section 00, there are 252,700 errors (26.6%) in over 950,000 words. By using the high-gram method and a grammar extracted from the gold data for section 00, we obtain a precision of 70.8% and recall of 61.0% at the 23% threshold: thus, one could correct 61% of the errors by looking at only 23% of the corpus (217,804 positions, with about 154,000 of them erroneous). Alternatively, using a threshold score of zero on the same data results in higher precision (79.5%) and lower recall (45.1%). This pattern of higher precision at the lowest threshold and higher recall for the 23% threshold is consistent across all testing scenarios, showing the effectiveness of correcting by working up from the lowest-scoring items.

Within the results for each parser, the grammar based on section 00 is more effective at sorting out the errors (i.e., has a higher $F_{0.5}$ score) than the other grammars—although the differences are small. This is true even for the parser trained on section 24 (70.2% vs. 69.4%), and the reverse situation (parser=00, grammar=24) even performs on a par with the other grammars. This indicates that a grammar from a different (albeit, related) corpus can be effective in error detection. Future work can explore applying this work across domains.

As for grammar size, the differences in $F_{0.5}$ between the smallest (section 24) and the largest (both sections) for all experiments is $<$1%. Even with a small gold grammar, we again conclude that this method can effectively sort out a majority of the errors with high precision.

5.3.1 Revision Checking

To gauge the results of revision checking, we report results for the WSJ parser trained only on section 00, as shown in table 8. The results for the other two parsers are not shown for space reasons, but they follow exactly the same trends, namely a consistent improvement, verifying the Talbanken results (section 5.2). For example, comparing to the 00 parser results without revision checking in table 7, we observe a greater $F_{0.5}$ value for all grammars by taking all flagged positions and the lowest-scoring unflagged positions (AF+U0). With the section 00 grammar, for instance, we see an improvement in $F_{0.5}$ from 68.9% to 71.6%.

Gr.	Score	Thr.	pos.	P	R	$F_{0.5}$
00	LF	0	78,805	**92.2%**	28.8%	66.4%
	AF	1,325	141,914	79.7%	44.8%	68.9%
	AF+U0	n/a	206,474	74.8%	61.1%	**71.6%**
24	LF	0	79,755	**91.9%**	29.0%	64.1%
	AF	1,167	138,912	79.4%	43.6%	68.2%
	AF+U0	n/a	211,438	73.7%	61.6%	**70.9%**
0024	LF	0	76,570	**92.0%**	28.0%	63.3%
	AF	3,200	144,041	80.3%	45.8%	69.8%
	AF+U0	n/a	198,195	76.1%	59.7%	**72.1%**

Table 8: WSJ revision checking results, using the high-gram method, for the parser trained on section 00; same set-up as in table 7

Perhaps the most notable feature of these results is the precision of the lowest-scoring flagged positions, around 92% for all grammars. This means that for this type of data, annotators could go over 80,000 positions, with very few false positives, providing much potential for efficient correction.

6 Noisy Grammar

In this section, we switch to using a noisy grammar, i.e., one extracted from the parsed data itself. Using the same parser for Swedish as in section 5.1 and parsing the large corpus, we extract a grammar from the parsed data and obtain the results for the high-gram method in table 9. As

the parsed rules are from the grammar, this can be seen as an internal consistency check.

Score	Thr.	pos.	P	R	$F_{0.5}$
None	n/a	191,467	35.4%	100%	40.7%
No-Rev.	3	267	88.4%	0.3%	1.7%
	76	44,274	68.4%	44.6%	61.8%
AF	2,784	12,769	67.9%	12.8%	36.4%
AF+U55	n/a	44,155	68.8%	44.7%	62.1%

Table 9: Talbanken error detection results for the high-gram method: parser = MaltParser trained on small data; grammar = large noisy; evaluation data = small data (*No-Rev.* = No revision checking)

These results are noticably lower than when using a small gold grammar (tables 3 and 5). Without revision checking, the high-gram $F_{0.5}$ score goes from 65.7% to 61.8%, in spite of the fact that the noisy grammar has 95,900 rule tokens and 25,904 types, compared to 3,107/1,284 in the gold grammar. That is, a small set of high-quality rules outperforms a large set of more questionable rules, possibly because of parser bias in the grammar.

As for the impact of revision checking, the precision for all 12,769 flagged positions is 67.9%—on a par with the precision without revision checking. Indeed, for all flagged positions combined with many lowest-scoring unflagged positions (up to the 55 threshold, or 23% of the corpus), the $F_{0.5}$ score is slightly improved, though still well below the small gold grammar case.

Comparing the noisy grammar in table 10 to the small gold grammars in table 7, the trend in English is more pronounced. For example, for the parser trained on section 00 (and the 00 grammar), the $F_{0.5}$ goes from 68.9% to 56.9%. Each noisy WSJ grammar has over 85,000 rules, yet the noise greatly pulls down the accuracy, thereby confirming our preference for (small) gold grammars.[11]

7 Summary and Outlook

Taking into account different ways in which automatic dependency parses are obtained, we have advocated for a grammar-based method of detecting parse errors and have illustrated the gains that can be made by using such methods, including the incorporation of revision checking. Methods using a small gold grammar outperform the methods using much larger grammars with noise in them. Thus, to employ such methods on new data, one needs a small grammar, perhaps from a small hand-annotated corpus. Using our methods can improve the resulting annotation of large amounts of parsed data, allowing annotators to focus on problematic parses and not correct ones.

[11] We also tried concatenating the small gold and large noisy grammars to make a hybrid grammar (section 3), but the results were hardly better than in tables 9 and 10.

Par.	Thr.	pos.	P	R	$F_{0.5}$
00	3	315	90.5%	0.1%	0.6%
	460	218,497	58.7%	50.7%	56.9%
24	3	400	90.5%	0.1%	0.7%
	438	218,612	61.1%	49.8%	58.4%
0024	3	377	91.8%	0.2%	0.8%
	479	218,512	55.7%	53.5%	55.3%

Table 10: WSJ: high-gram scores: parser = default MST trained on different data (*Par.*); grammar = extracted from appropriate parsed 02-21; evaluation data = 02-21

In the future, one can test these methods on real-world corpus-building efforts, integrating them into a particular annotation workflow. Because the method of scoring parses is very general, one can also explore using the scores in different contexts, such as scoring the validity of parse structures in a parser combination model (e.g., Surdeanu and Manning, 2010; Sagae and Tsujii, 2007; Sagae and Lavie, 2006); sorting sentences for active learning (cf. Sassano and Kurohashi, 2010); or selecting parse structures for parsing (Chen et al., 2009).

Acknowledgments

We would like to thank Jennifer Foster, the IU CL discussion group, and the three anonymous reviewers for useful feedback.

A Some Talbanken05 categories

POS tags	
++	coord. conj.
AB	adverb
AJ	adjective
EN	indef. article
FV	*få* (get)
NN	noun
PO	pronoun
PR	preposition
RO	numeral
VN	verbal noun
VV	verb

Dependencies	
++	coord. conj.
AT	nominal pre-modifier
CC	sister of first conjunct (binary branching coordination)
DT	determiner
FS	dummy subject
MA	attitude adverbial
NA	negation adverbial
PA	preposition comp.
PL	verb particle
RA	place adverbial
SS	subject
VG	verb group

References

Abeillé, Anne (ed.) (2003). *Treebanks: Building and using syntactically annoted corpora*. Dordrecht: Kluwer Academic Publishers.

Ambati, Bharat Ram (2010). Importance of Linguistic Constraints in Statistical Dependency Parsing. In *Proceedings of the ACL 2010 Student Research Workshop*. Uppsala, Sweden, pp. 103–108.

Ambati, Bharat Ram, Mridul Gupta, Samar Husain and Dipti Misra Sharma (2010). A high recall error identification tool for Hindi treebank validation. In *Proceedings of The 7th International Conference on Language Resources and Evaluation (LREC)*. Valletta, Malta.

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

Bergsma, Shane, Dekang Lin and Randy Goebel (2009). Web-Scale N-gram Models for Lexical Disambiguation. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. Pasadena, California, pp. 1507–1512.

Bond, Francis, Sanae Fujita et al. (2004). The Hinoki Treebank: Toward Text Understanding. In *Proceedings of the 5th International Workshop on Linguistically Interpreted Corpora (LINC-04)*. Geneva, pp. 7–10.

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.

Chen, Wenliang, Jun'ichi Kazama, Kiyotaka Uchimoto and Kentaro Torisawa (2009). Improving Dependency Parsing with Subtrees from Auto-Parsed Data. In *Proceedings of EMNLP-09*. Singapore, pp. 570–579.

Dickinson, Markus (2010). Detecting Errors in Automatically-Parsed Dependency Relations. In *The 48th Annual Meeting of the Association for Computational Linguistics (ACL-10)*. Uppsala, Sweden.

Dickinson, Markus (2011). Detecting Ad Hoc Rules for Treebank Development. *Linguistic Issues in Language Technology* 4(3).

Goldberg, Yoav and Michael Elhadad (2010a). An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Los Angeles, California, pp. 742–750.

Goldberg, Yoav and Michael Elhadad (2010b). Inspecting the Structural Biases of Dependency Parsing Algorithms. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*. Uppsala, Sweden, pp. 234–242.

Hara, Tadayoshi, Yusuke Miyao and Jun'ichi Tsujii (2009). Effective Analysis of Causes and Inter-dependencies of Parsing Errors. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*. Paris, France: Association for Computational Linguistics, pp. 180–191.

Hwa, Rebecca, Philip Resnik, Amy Weinberg, Clara Cabezas and Okan Kolak (2005). Bootstrapping parsers via syntactic projection across parallel texts. *Natural Language Engineering* 11, 311–325.

Johansson, Richard and Pierre Nugues (2007). Extended Constituent-to-dependency Conversion for English. In *Proceedings of NODALIDA 2007*. Tartu, Estonia.

Leech, Geoffrey (2004). Adding Linguistic Annotation. In Martin Wynne (ed.), *Developing Linguistic Corpora: a Guide to Good Practice*, Oxford: Oxbow Books, pp. 17–29.

Marcus, M., Beatrice Santorini and M. A. Marcinkiewicz (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics* 19(2), 313–330.

McDonald, Ryan, Koby Crammer and Fernando Pereira (2005). Online Large-Margin Training of Dependency Parsers. In *Proceedings of the 43rd Annual Meeting of the ACL*. AnnArbor, pp. 91–98.

McDonald, Ryan, Kevin Lerman and Fernando Pereira (2006). Multilingual Dependency Analysis with a Two-Stage Discriminative Parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*. New York City, pp. 216–220.

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 (2010). Harvest Time - Explorations of the Swedish Treebank. *The Ninth Treebanks and Linguistic Theories Workshop*, Invited Talk. Tartu, Estonia. December 4, 2010.

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.

Oepen, Stephan, Dan Flickinger and Francis Bond (2004). Towards holistic grammar engineering and testing—grafting treebank maintenance into the grammar revision cycle. In *Beyond Shallow Analyses—Formalisms and Statistical Modelling for Deep Analysis (Workshop at The First International Joint Conference on Natural Language Processing (IJCNLP-04))*. Hainan, China.

Plank, Barbara and Gertjan van Noord (2010). Grammar-Driven versus Data-Driven: Which Parsing System Is More Affected by Domain Shifts? In *Proceedings of the 2010 Workshop on NLP and Linguistics: Finding the Common Ground*. Uppsala, Sweden, pp. 25–33.

Rosén, Victoria, Koenraad de Smedt, Helge Dyvik and Paul Meurer (2005). TREPIL: Developing Methods and Tools for Multilevel Treebank Construction. In *Proceedings of the Fourth Workshop on Treebanks and Linguistic Theories (TLT 2005)*. Barcelona, Spain, pp. 161–172.

Sagae, Kenji and Alon Lavie (2006). Parser Combination by Reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*. New York City, USA, pp. 129–132.

Sagae, Kenji and Jun'ichi Tsujii (2007). Dependency Parsing and Domain Adaptation with LR Models and Parser Ensembles. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*. Prague, Czech Republic, pp. 1044–1050.

Sassano, Manabu and Sadao Kurohashi (2010). Using Smaller Constituents Rather Than Sentences in Active Learning for Japanese Dependency Parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Uppsala, Sweden, pp. 356–365.

Seeker, Wolfgang, Ines Rehbein, Jonas Kuhn and Josef Van Genabith (2010). Hard Constraints for Grammatical Function Labelling. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Uppsala, Sweden, pp. 1087–1097.

Simpson, Heather, Kazuaki Maeda and Christopher Cieri (2009). Basic Language Resources for Diverse Asian Languages: A Streamlined Approach for Resource Creation. In *Proceedings of the 7th Workshop on Asian Language Resources*. Suntec, Singapore, pp. 55–62.

Surdeanu, Mihai and Christopher D. Manning (2010). Ensemble Models for Dependency Parsing: Cheap and Good? In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Los Angeles, pp. 649–652.

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 Abeillé (2003), pp. 61–71.

Zhao, Yanyan, Bing Qin, Shen Hu and Ting Liu (2010). Generalizing Syntactic Structures for Product Attribute Candidate Extraction. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Los Angeles, California, pp. 377–380.