Parser Showdown at the Wall Street Corral: An Empirical Investigation of Error Types in Parser Output

Jonathan K. Kummerfeld[†] **David Hall**[†] **James R. Curran**[‡] **Dan Klein**[†]

[†]Computer Science Division
University of California, Berkeley
Berkeley, CA 94720, USA
{jkk,dlwh,klein}@cs.berkeley.edu

[‡]ə-lab, School of IT
University of Sydney
Sydney, NSW 2006, Australia
james@it.usyd.edu.au

Abstract

Constituency parser performance is primarily interpreted through a single metric, F-score on WSJ section 23, that conveys no linguistic information regarding the remaining errors. We classify errors within a set of linguistically meaningful types using tree transformations that repair groups of errors together. We use this analysis to answer a range of questions about parser behaviour, including what linguistic constructions are difficult for state-of-the-art parsers, what types of errors are being resolved by rerankers, and what types are introduced when parsing out-of-domain text.

1 Introduction

Parsing has been a major area of research within computational linguistics for decades, and constituent parser F-scores on WSJ section 23 have exceeded 90% (Petrov and Klein, 2007), and 92% when using self-training and reranking (McClosky et al., 2006; Charniak and Johnson, 2005). While these results give a useful measure of overall performance, they provide no information about the nature, or relative importance, of the remaining errors.

Broad investigations of parser errors beyond the PARSEVAL metric (Abney et al., 1991) have either focused on specific parsers, e.g. Collins (2003), or have involved conversion to dependencies (Carroll et al., 1998; King et al., 2003). In all of these cases, the analysis has not taken into consideration how a set of errors can have a common cause, e.g. a single mis-attachment can create multiple node errors.

We propose a new method of error classification using tree transformations. Errors in the parse tree are repaired using subtree movement, node creation, and node deletion. Each step in the process is then associated with a linguistically meaningful error type, based on factors such as the node that is moved, its siblings, and parents.

Using our method we analyse the output of thirteen constituency parsers on newswire. Some of the frequent error types that we identify are widely recognised as challenging, such as prepositional phrase (PP) attachment. However, other significant types have not received as much attention, such as clause attachment and modifier attachment.

Our method also enables us to investigate where reranking and self-training improve parsing. Previously, these developments were analysed only in terms of their impact on F-score. Similarly, the challenge of out-of-domain parsing has only been expressed in terms of this single objective. We are able to decompose the drop in performance and show that a disproportionate number of the extra errors are due to coordination and clause attachment.

This work presents a comprehensive investigation of parser behaviour in terms of linguistically meaningful errors. By applying our method to multiple parsers and domains we are able to answer questions about parser behaviour that were previously only approachable through approximate measures, such as counts of node errors. We show which errors have been reduced over the past fifteen years of parsing research; where rerankers are making their gains and where they are not exploiting the full potential of k-best lists; and what types of errors arise when moving out-of-domain. We have released our system[1] to enable future work to apply our methodology.

[1]http://code.google.com/p/berkeley-parser-analyser/

2 Background

Most attempts to understand the behaviour of constituency parsers have focused on overall evaluation metrics. The three main methods are intrinsic evaluation with PARSEVAL, evaluation on dependencies extracted from the constituency parse, and evaluation on downstream tasks that rely on parsing.

Intrinsic evaluation with PARSEVAL, which calculates precision and recall over labeled tree nodes, is a useful indicator of overall performance, but does not pinpoint which structures the parser has most difficulty with. Even when the breakdown for particular node types is presented (e.g. Collins, 2003), the interaction between node errors is not taken into account. For example, a VP node could be missing because of incorrect PP attachment, a coordination error, or a unary production mistake. There has been some work that addresses these issues by analysing the output of constituency parsers on linguistically motivated error types, but only by hand on sets of around 100 sentences (Hara et al., 2007; Yu et al., 2011). By automatically classifying parse errors we are able to consider the output of multiple parsers on thousands of sentences.

The second major parser evaluation method involves extraction of grammatical relations (King et al., 2003; Briscoe and Carroll, 2006) or dependencies (Lin, 1998; Briscoe et al., 2002). These metrics have been argued to be more informative and generally applicable (Carroll et al., 1998), and have the advantage that the breakdown over dependency types is more informative than over node types. There have been comparisons of multiple parsers (Foster and van Genabith, 2008; Nivre et al., 2010; Cer et al., 2010), as well as work on finding relations between errors (Hara et al., 2009), and breaking down errors by a range of factors (McDonald and Nivre, 2007). However, one challenge is that results for constituency parsers are strongly influenced by the dependency scheme being used and how easy it is to extract the dependencies from a given parser's output (Clark and Hockenmaier, 2002). Our approach does not have this disadvantage, as we analyse parser output directly.

The third major approach involves extrinsic evaluation, where the parser's output is used in a downstream task, such as machine translation (Quirk and Corston-Oliver, 2006), information extraction (Miyao et al., 2008), textual entailment (Yuret et al., 2010), or semantic dependencies (Dridan and Oepen, 2011). While some of these approaches give a better sense of the impact of parse errors, they require integration into a larger system, making it less clear where a given error originates.

The work we present here differs from existing approaches by directly and automatically classifying errors into meaningful types. This enables the first very broad, yet detailed, study of parser behaviour, evaluating the output of thirteen parsers over thousands of sentences.

3 Parsers

Our evaluation is over a wide range of PTB constituency parsers and their variants from the past fifteen years. For all parsers we used the publicly available version, with the standard parameter settings.

Berkeley (Petrov et al., 2006; Petrov and Klein, 2007). An unlexicalised parser with a grammar constructed with automatic state splitting.

Bikel (2004) implementation of Collins (1997).

BUBS (Dunlop et al., 2011; Bodenstab et al., 2011). A 'grammar-agnostic constituent parser,' which uses a Berkeley Parser grammar, but parses with various pruning techniques to improve speed, at the cost of accuracy.

Charniak (2000). A generative parser with a maximum entropy-inspired model. We also use the reranker (Charniak and Johnson, 2005), and the self-trained model (McClosky et al., 2006).

Collins (1997). A generative lexicalised parser, with three models, a base model, a model that uses subcategorisation frames for head words, and a model that takes into account traces.

SSN (Henderson, 2003; Henderson, 2004). A statistical left-corner parser, with probabilities estimated by a neural network.

Stanford (Klein and Manning, 2003a; Klein and Manning, 2003b). We consider both the unlexicalised PCFG parser (-U) and the factored parser (-F), which combines the PCFG parser with a lexicalised dependency parser.

System	F	P	R	Exact	Speed
ENHANCED TRAINING / SYSTEMS					
Charniak-SR	92.07	92.44	91.70	44.87	1.8
Charniak-R	91.41	91.78	91.04	44.04	1.8
Charniak-S	91.02	91.16	90.89	40.77	1.8
STANDARD PARSERS					
Berkeley	90.06	90.30	89.81	36.59	4.2
Charniak	89.71	89.88	89.55	37.25	1.8
SSN	89.42	89.96	88.89	32.74	1.8
BUBS	88.50	88.57	88.43	31.62	27.6
Bikel	88.16	88.23	88.10	32.33	0.8
Collins-3	87.66	87.82	87.50	32.22	2.0
Collins-2	87.62	87.77	87.48	32.51	2.2
Collins-1	87.09	87.29	86.90	30.35	3.3
Stanford-L	86.42	86.35	86.49	27.65	0.7
Stanford-U	85.78	86.48	85.09	28.35	2.7

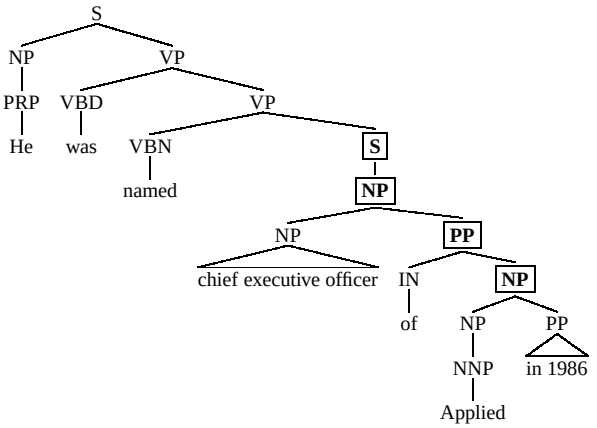
Table 1: PARSEVAL results on WSJ section 23 for the parsers we consider. The columns are F-score, precision, recall, exact sentence match, and speed (sents/sec). Coverage was left out as it was above 99.8% for all parsers. In the ENHANCED TRAINING / SYSTEMS section we include the Charniak parser with reranking (R), with a self-trained model (S), and both (SR).

Table 1 shows the standard performance metrics, measured on section 23 of the WSJ, using all sentences. Speeds were measured using a Quad-Core Xeon CPU (2.33GHz 4MB L2 cache) with 16GB of RAM. These results clearly show the variation in parsing performance, but they do not show which constructions are the source of those variations.

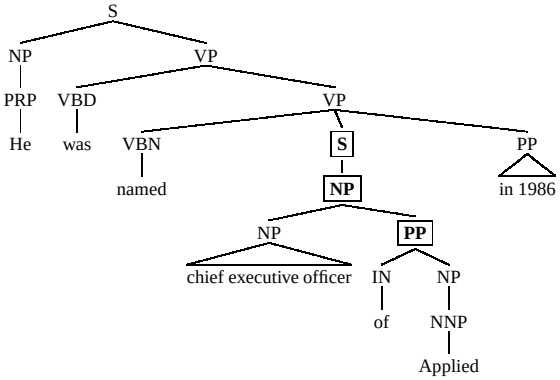
4 Error Classification

While the statistics in Table 1 give a sense of overall parser performance they do not provide linguistically meaningful intuition for the source of remaining errors. Breaking down the remaining errors by node type is not particularly informative, as a single attachment error can cause multiple node errors, many of which are for unrelated node types. For example, in Figure 1 there is a PP attachment error that causes seven bracket errors (extra S, NP, PP, and NP, missing S, NP, and PP). Determining that these correspond to a PP attachment error from just the labels of the missing and extra nodes is difficult. In contrast, the approach we describe below takes into consideration the relations between errors, grouping them into linguistically meaningful sets.

We classify node errors in two phases. First, we find a set of tree transformations that convert the output tree into the gold tree. Second, the transformation are classified into error types such as PP attachment and coordination. Pseudocode for our method is shown in Algorithm 1. The tree transformation stage corresponds to the main loop, while the second stage corresponds to the final loop.



(a) Parser output



(b) Gold tree

Figure 1: Grouping errors by node type is of limited usefulness. In this figure and those that follow the top tree is the incorrect parse and the bottom tree is the correct parse. Bold, boxed nodes are either extra (marked in the incorrect tree) or missing (marked in the correct tree). This is an example of **PP Attachment** (*in 1986* is too low), but that is not at all clear from the set of incorrect nodes (extra S, NP, PP, and NP, missing S, NP, and PP).

4.1 Tree Transformation

The core of our transformation process is a set of operations that move subtrees, create nodes, and delete nodes. Searching for the shortest path to transform one tree into another is prohibitively slow.[2] We find

[2]We implemented various search procedures and found similar results on the sentences that could be processed in a reason-

Algorithm 1 Tree transformation error classification

```
U = initial set of node errors
Sort U by the depth of the error in the tree, deepest first
G = ∅
repeat
    for all errors e ∈ U do
        if e fits an environment template t then
            g = new error group
            Correct e as specified by t
            for all errors f that t corrects do
                Remove f from U
                Insert f into g
            end for
            Add g to G
        end if
    end for
until unable to correct any further errors
for all remaining errors e ∈ U do
    Insert a group into G containing e
end for
for all groups g ∈ G do
    Classify g based on properties of the group
end for
```

a path by applying a greedy bottom–up approach, iterating through the errors in order of tree depth.

We match each error with a template based on nearby tree structure and errors. For example, in Figure 1 there are four extra nodes that all cover spans ending at *Applied in 1986*: S, NP, PP, NP. There are also three missing nodes with spans ending between *Applied* and *in*: PP, NP, and S. Figure 2 depicts these errors as spans, showing that this case fits three criteria: (1) there are a set of extra spans all ending at the same point, (2) there are a set of missing spans all ending at the same point, and (3) the extra spans cross the missing spans, extending beyond their end-point. This indicates that the node starting after *Applied* is attaching too low and should be moved up, outside all of the extra nodes. Together, the criteria and transformation form a template.

Once a suitable template is identified we correct the error by moving subtrees, adding nodes and removing nodes. In the example this is done by moving the node spanning *in 1986* up in the tree until it is outside of all the extra spans. Since moving the PP leaves a unary production from an NP to an NP, we also collapse that level. In total this corrects seven

able amount of time.

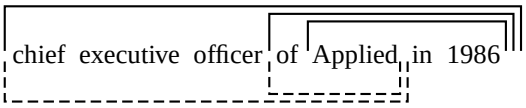


Figure 2: Templates are defined in terms of extra and missing spans, shown here with unbroken lines above and dashed lines below, respectively. This is an example of a set of extra spans that cross a set of missing spans (which in both cases all end at the same position). If the last two words are moved, two of the extra spans will match the two missing spans. The other extra span is deleted during the move as it creates an NP→NP unary production.

errors, as there are three cases in which an extra node is present that matches a missing node once the PP is moved. All of these errors are placed in a single group and information about the nearby tree structure before and after the transformation is recorded.

We continue to make passes through the list until no errors are corrected on a pass. For each remaining node error an individual error group is created.

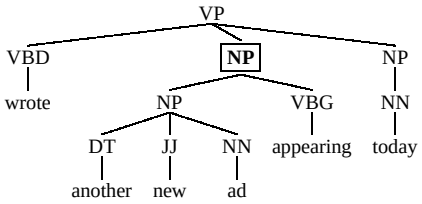
The templates were constructed by hand based on manual analysis of parser output. They cover a range of combinations of extra and missing spans, with further variation for whether crossing is occurring and if so whether the crossing bracket starts or ends in the middle of the correct bracket. Errors that do not match any of our templates are left uncorrected.

4.2 Transformation Classification

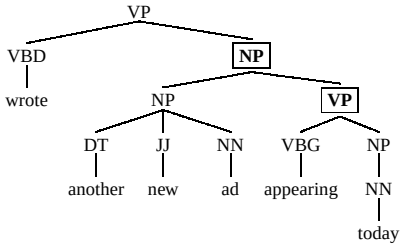
We began with a large set of node errors, in the first stage they were placed into groups, one group per tree transformation used to get from the test tree to the gold tree. Next we classify each group as one of the error types below.

PP Attachment Any case in which the transformation involved moving a Prepositional Phrase, or the incorrect bracket is over a PP, e.g.
He was (VP *named chief executive officer of* (NP *Applied* (PP *in 1986*)))
where (PP *in 1986*) should modify the entire VP, rather than just *Applied*.

NP Attachment Several cases in which NPs had to be moved, particularly for mistakes in appositive constructions and incorrect attachments within a verb phrase, e.g.
The bonds (VP *go* (PP *on sale* (NP *Oct. 19*)))
where *Oct. 19* should be an argument of *go*.

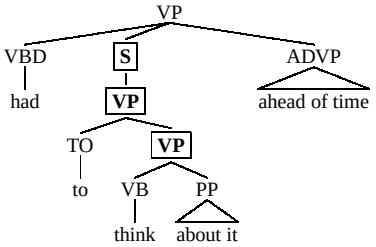


(a) Parser output

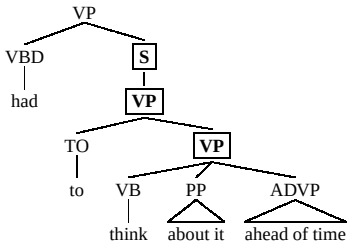


(b) Gold tree

Figure 3: **NP Attachment**: *today* is too high, it should be the argument of *appearing*, rather than *wrote*. This causes three node errors (extra NP, missing NP and VP).



(a) Parser output

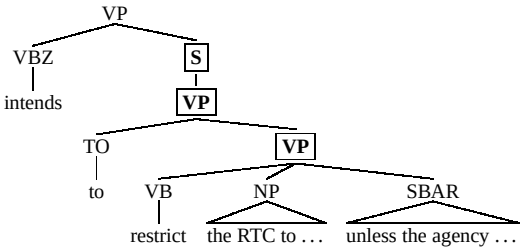


(b) Gold tree

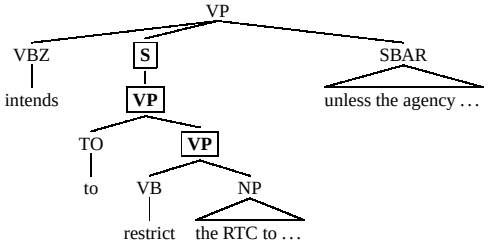
Figure 4: **Modifier Attachment**: *ahead of time* is too high, it should modify *think*, not *had*. This causes six node errors (extra S, VP, and VP, missing S, VP, and VP).

Modifier Attachment Cases involving incorrectly placed adjectives and adverbs, including errors corrected by subtree movement and errors requiring only creation of a node, e.g.
(NP (ADVP *even more*) *severe setbacks*)
where there should be an extra ADVP node over *even more severe*.

Clause Attachment Any group that involves movement of some form of S node.

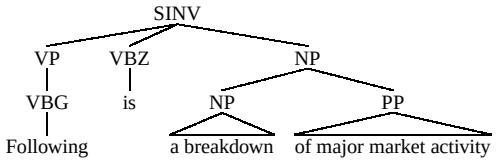


(a) Parser output

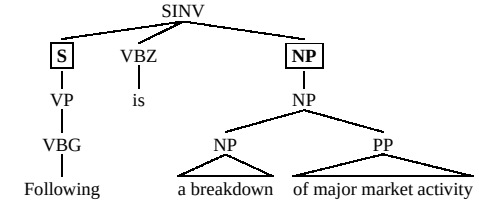


(b) Gold tree

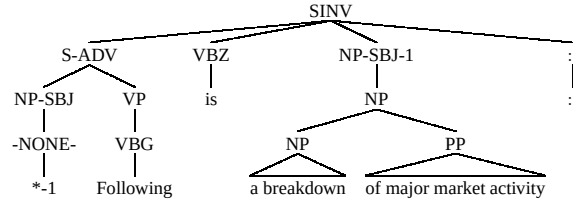
Figure 5: **Clause Attachment**: *unless the agency receives specific congressional authorization* is attaching too low. This causes six node errors (extra S, VP, and VP, missing S, VP and VP).



(a) Parser output

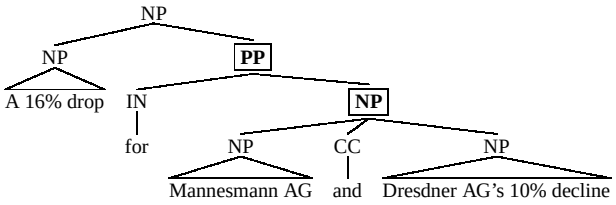


(b) Gold tree

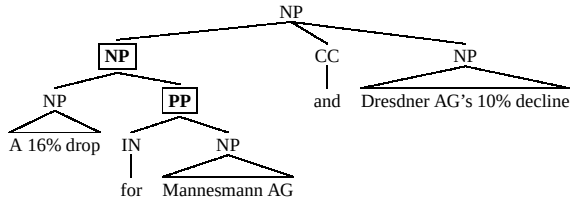


(c) Gold tree with traces and function tags

Figure 6: Two **Unary** errors, a missing S and a missing NP. The third tree is the PTB tree before traces and function tags are removed. Note that the missing NP is over another NP, a production that does occur widely in the treebank, particularly over the word *it*.

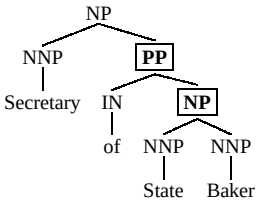


(a) Parser output

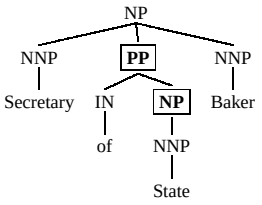


(b) Gold tree

Figure 7: **Coordination**: *and Dresdner AG's 10% decline* is too low. This causes four node errors (extra PP and NP, missing NP and PP).



(a) Parser output



(b) Gold tree

Figure 8: **NP Internal Structure**: *Baker* is too low, causing four errors (extra PP and NP, missing PP and NP).

Unary Mistakes involving unary productions that are not linked to a nearby error such as a matching extra or missing node. We do not include a breakdown by unary type, though we did find that clause labeling (S, SINV, etc) accounted for a large proportion of the errors.

Coordination Cases in which a conjunction is an immediate sibling of the nodes being moved, or is the leftmost or rightmost node being moved.

NP Internal Structure While most NP structure is not annotated in the PTB, there is some use of ADJP, NX, NAC and QP nodes. We form a single group for each NP that has one or more errors involving these types of nodes.

Different label In many cases a node is present in the tree that spans the correct set of words, but has the wrong label, in which case we group the two node errors, (one extra, one missing), as a single error.

Single word phrase A range of node errors that span a single word, with checks to ensure this is not linked to another error (e.g. one part of a set of internal noun phrase errors).

Other There is a long tail of other errors. Some could be placed within the categories above, but would require far more specific rules.

For many of these error types it would be difficult to extract a meaningful understanding from only the list of node errors involved. Even for error types that can be measured by counting node errors or rule production errors, our approach has the advantage that we identify groups of errors with a single cause. For example, a missing unary production may correspond to an extra bracket that contains a subtree that attached incorrectly.

4.3 Methodology

We used sections 00 and 24 as development data while constructing the tree transformation and error group classification methods. All of our examples in text come from these sections as well, but for all tables of results we ran our system on section 23. We chose to run our analysis on section 23 as it is the only section we are sure was not used in the development of any of the parsers, either for tuning or feature development. Our evaluation is entirely focused on the errors of the parsers, so unless there is a particular construction that is unusually prevalent in section 23, we are not revealing any information about the test set that could bias future work.

5 Results

Our system enables us to answer questions about parser behaviour that could previously only be probed indirectly. We demonstrate its usefulness by applying it to a range of parsers (here), to reranked K-best lists of various lengths, and to output for out-of-domain parsing (following sections).

In Table 2 we consider the breakdown of parser

Parser	F-score	PP Attach	Clause Attach	Diff Label	Mod Attach	NP Attach	Co-ord	1-Word Span	Unary	NP Int.	Other
Best		0.60	0.38	0.31	0.25	0.25	0.23	0.20	0.14	0.14	0.50
Charniak-RS	92.07										
Charniak-R	91.41										
Charniak-S	91.02										
Berkeley	90.06										
Charniak	89.71										
SSN	89.42										
BUBS	88.63										
Bikel	88.16										
Collins-3	87.66										
Collins-2	87.62										
Collins-1	87.09										
Stanford-F	86.42										
Stanford-U	85.78										
Worst		1.12	0.61	0.51	0.39	0.45	0.40	0.42	0.27	0.27	1.13

Table 2: Average number of bracket errors per sentence due to the top ten error types. For instance, Stanford-U produces output that has, on average, 1.12 bracket errors per sentence that are due to PP attachment. The scale for each column is indicated by the Best and Worst values.

Error Type	Occurrences	Nodes Involved	Ratio
PP Attachment	846	1455	1.7
Single word phrase	490	490	1.0
Clause Attachment	385	913	2.4
Modifier Attachment	383	599	1.6
Different Label	377	754	2.0
Unary	347	349	1.0
NP Attachment	321	597	1.9
NP Internal Structure	299	352	1.2
Coordination	209	557	2.7
Unary Clause Label	185	200	1.1
VP Attachment	64	159	2.5
Parenthetical Attachment	31	74	2.4
Missing Parenthetical	12	17	1.4
Unclassified	655	734	1.1

Table 3: Breakdown of errors on section 23 for the Charniak parser with self-trained model and reranker. Errors are sorted by the number of times they occur. Ratio is the average number of node errors caused by each error we identify (i.e. Nodes Involved / Occurrences).

errors on WSJ section 23. The shaded area of each bar indicates the frequency of parse errors (i.e. empty means fewest errors). The area filled in is determined by the expected number of node errors per sentence that are attributed to that type of error. The average number of node errors per sentence for a completely full bar is indicated by the Worst row, and the value for a completely empty bar is indicated by the Best row. Exact error counts are available at http://code.google.com/p/berkeley-parser-analyser/.

We use counts of node errors to make the contributions of each type of error more interpretable. As Table 3 shows, some errors typically cause only a single node error, where as others, such as coordination, generally cause several. This means that considering counts of error groups would overemphasise some error types, e.g. single word phrase errors are second most important by number of groups (in Table 3), but seventh by total number of node errors (in Table 2).

As expected, PP attachment is the largest contributor to errors, across all parsers. Interestingly, coordination is sixth on the list, though that is partly due to the fact that there are fewer coordination decisions to be made in the treebank.[3]

By looking at the performance of the Collins parser we can see the development over the past fifteen years. There has been improvement across the board, but in some cases, e.g. clause attachment errors and different label errors, the change has been more limited (24% and 29% reductions respectively). We investigated the breakdown of the different label errors by label, but no particular cases of la-

[3]This is indicated by the frequency of CCs and PPs in sections 02–21 of the treebank, 16,844 and 95,581 respectively. These counts are only an indicator of the number of decisions as the nodes can be used in ways that do not involve a decision, such as sentences that start with a conjunction.

System	K	F-score	PP Attach	Clause Attach	Diff Label	Mod Attach	NP Attach	Co-ord	1-Word Span	Unary	NP Int.	Other
Best			0.08	0.04	0.08	0.05	0.06	0.04	0.08	0.04	0.04	0.11
Oracle	1000	98.30										
	100	97.54										
	50	97.18										
	20	96.40										
	10	95.66										
	5	94.61										
	2	92.59										
Charniak	1000	92.07										
	100	92.08										
	50	92.07										
	20	92.05										
	10	92.16										
	5	91.94										
	2	91.56										
	1	91.02										
Worst			0.66	0.43	0.33	0.26	0.28	0.26	0.23	0.16	0.19	0.60

Table 4: Average number of bracket errors per sentence for a range of K-best list lengths using the Charniak parser with reranking and the self-trained model. The oracle results are determined by taking the parse in each K-best list with the highest F-score.

bel confusion stand out, and we found that the most common cases remained the same between Collins and the top results.

It is also interesting to compare pairs of parsers that share aspects of their architecture. One such pair is the Stanford parser, where the factored parser combines the unlexicalised parser with a lexicalised dependency parser. The main sources of the 0.64 gain in F-score are PP attachment and coordination.

Another interesting pair is the Berkeley parser and the BUBS parser, which uses a Berkeley grammar, but improves speed by pruning. The pruning methods used in BUBS are particularly damaging for PP attachment errors and unary errors.

Various comparisons can be made between Charniak parser variants. We discuss the reranker below. For the self-trained model McClosky et al. (2006) performed some error analysis, considering variations in F-score depending on the frequency of tags such as PP, IN and CC in sentences. Here we see gains on all error types, though particularly for clause attachment, modifier attachment and coordination, which fits with their observations.

5.1 Reranking

The standard dynamic programming approach to parsing limits the range of features that can be employed. One way to deal with this issue is to modify the parser to produce the top K parses, rather than just the 1-best, then use a model with more sophisticated features to choose the best parse from this list (Collins, 2000). While re-ranking has led to gains in performance (Charniak and Johnson, 2005), there has been limited analysis of how effectively rerankers are using the set of available options. Recent work has explored this question in more depth, but focusing on how variation in the parameters impacts performance on standard metrics (Huang, 2008; Ng et al., 2010; Auli and Lopez, 2011; Ng and Curran, 2012).

In Table 4 we present a breakdown over error types for the Charniak parser, using the self-trained model and reranker. The oracle results use the parse in each K-best list with the highest F-score. While this may not give the true oracle result, as F-score does not factor over sentences, it gives a close approximation. The table has the same columns as Table 2, but the ranges on the bars now reflect the min and max for these sets.

While there is improvement on all errors when using the reranker, there is very little additional gain beyond the first 5-10 parses. Even for the oracle results, most of the improvement occurs within the first 5-10 parses. The limited utility of extra parses

Corpus	F-score	PP Attach	Clause Attach	Diff Label	Mod Attach	NP Attach	Co-ord	1-Word Span	Unary	NP Int.	Other
Best		0.022	0.016	0.013	0.011	0.011	0.010	0.009	0.006	0.005	0.021
WSJ 23	92.07										
Brown-F	85.91										
Brown-G	84.56										
Brown-K	84.09										
Brown-L	83.95										
Brown-M	84.65										
Brown-N	85.20										
Brown-P	84.09										
Brown-R	83.60										
G-Web Blogs	84.15										
G-Web Email	81.18										
Worst		0.040	0.035	0.053	0.020	0.034	0.023	0.046	0.009	0.029	0.073

Table 5: Average number of node errors per word for a range of domains using the Charniak parser with reranking and the self-trained model. We use per word error rates here rather than per sentence as there is great variation in average sentence length across the domains, skewing the per sentence results.

for the reranker may be due to the importance of the base parser output probability feature (which, by definition, decreases within the K-best list).

Interestingly, the oracle performance improves across all error types, even at the 2-best level. This indicates that the base parser model is not particularly biased against a single error. Focusing on the rows for $K = 2$ we can also see two interesting outliers. The PP attachment improvement of the oracle is considerably higher than that of the reranker, particularly compared to the differences for other errors, suggesting that the reranker lacks the features necessary to make the decision better than the parser. The other interesting outlier is NP internal structure, which continues to make improvements for longer lists, unlike the other error types.

5.2 Out-of-Domain

Parsing performance drops considerably when shifting outside of the domain a parser was trained on (Gildea, 2001). Clegg and Shepherd (2005) evaluated parsers qualitatively on node types and rule productions. Bender et al. (2011) designed a Wikipedia test set to evaluate parsers on dependencies representing ten specific linguistic phenomena.

To provide a deeper understanding of the errors arising when parsing outside of the newswire domain, we analyse performance of the Charniak parser with reranker and self-trained model on the eight parts of the Brown corpus (Marcus et al., 1993), and two parts of the Google Web corpus (Petrov and McDonald, 2012). Table 6 shows statistics for the corpora. The variation in average sentence lengths skew the results for errors per sentence. To handle this we divide by the number of words to determine the results in Table 5, rather than by the number of sentences, as in previous figures.

Corpus	Description	Sentences	Av. Length
WSJ 23	Newswire	2416	23.5
Brown F	Popular	3164	23.4
Brown G	Biographies	3279	25.5
Brown K	General	3881	17.2
Brown L	Mystery	3714	15.7
Brown M	Science	881	16.6
Brown N	Adventure	4415	16.0
Brown P	Romance	3942	17.4
Brown R	Humour	967	22.7
G-Web Blogs	Blogs	1016	23.6
G-Web Email	E-mail	2450	11.9

Table 6: Variation in size and contents of the domains we consider. The variation in average sentence lengths skews the results for errors per sentences, and so in Table 5 we consider errors per word.

There are several interesting features in the table. First, on the Brown datasets, while the general trend is towards worse performance on all errors, NP internal structure is a notable exception and in some cases PP attachment and unaries are as well.

In the other errors we see similar patterns across the corpora, except humour (Brown R), on which the parser is particularly bad at coordination and clause

attachment. This makes sense, as the colloquial nature of the text includes more unusual uses of conjunctions, for example:

She was a living doll and no mistake – the ...

Comparing the Brown corpora and the Google Web corpora, there are much larger divergences. We see a particularly large decrease in NP internal structure. Looking at some of the instances of this error, it appears to be largely caused by incorrect handling of structures such as URLs and phone numbers, which do not appear in the PTB. There are also some more difficult cases, for example:

... going up for sale in the next month or do .

where *or do* is a QP. This typographical error is extremely difficult to handle for a parser trained only on well-formed text.

For e-mail there is a substantial drop on single word phrases. Breaking the errors down by label we found that the majority of the new errors are missing or extra NPs over single words. Here the main problem appears to be temporal expressions, though there also appear to be a substantial number of errors that are also at the POS level, such as when NNP is assigned to *ta* in this case:

... let you know that I 'm out ta here !

Some of these issues, such as URL handling, could be resolved with suitable training data. Other issues, such as ungrammatical language and unconventional use of words, pose a greater challenge.

6 Conclusion

The single F-score objective over brackets or dependencies obscures important differences between statistical parsers. For instance, a single attachment error can lead to one or many mismatched brackets.

We have created a novel tree-transformation methodology for evaluating parsers that categorises errors into linguistically meaningful types. Using this approach, we presented the first detailed examination of the errors produced by a wide range of constituency parsers for English. We found that PP attachment and clause attachment are the most challenging constructions, while coordination turns out to be less problematic than previously thought. We also noted interesting variations in error types for parsers variants.

We investigated the errors resolved in reranking, and introduced by changing domains. We found that the Charniak rerankers improved most error types, but made little headway on improving PP attachment. Changing domain has an impact on all error types, except NP internal structure.

We have released our system so that future constituent parsers can be evaluated using our methodology. Our analysis provides new insight into the development of parsers over the past fifteen years, and the challenges that remain.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful suggestions. This research was partially supported by a General Sir John Monash Fellowship to the first author, the Office of Naval Research under MURI Grant No. N000140911081, an NSF Fellowship to the second author, ARC Discovery grant DP1097291, the Capital Markets CRC, and the NSF under grant 0643742.

References

S. Abney, S. Flickenger, C. Gdaniec, C. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. Procedure for quantitatively comparing the syntactic coverage of english grammars. In *Proceedings of the workshop on Speech and Natural Language*, pages 306–311, Pacific Grove, California, USA, February.

Michael Auli and Adam Lopez. 2011. A comparison of loopy belief propagation and dual decomposition for integrated ccg supertagging and parsing. In *Proceedings of ACL*, pages 470–480, Portland, Oregon, USA, June.

Emily M. Bender, Dan Flickinger, Stephan Oepen, and Yi Zhang. 2011. Parser evaluation over local and nonlocal deep dependencies in a large corpus. In *Proceedings of EMNLP*, pages 397–408, Edinburgh, United Kingdom, July.

Daniel M. Bikel. 2004. Intricacies of collins' parsing model. *Computational Linguistics*, 30(4):479–511.

Nathan Bodenstab, Aaron Dunlop, Keith Hall, and Brian Roark. 2011. Beam-width prediction for efficient context-free parsing. In *Proceedings of ACL*, pages 440–449, Portland, Oregon, USA, June.

Ted Briscoe and John Carroll. 2006. Evaluating the accuracy of an unlexicalized statistical parser on the PARC DepBank. In *Proceedings of ACL*, pages 41–48, Sydney, Australia, July.

Ted Briscoe, John Carroll, Jonathan Graham, and Ann Copestake, 2002. *Relational Evaluation Schemes*, pages 4–8. Las Palmas, Canary Islands, Spain, May.

John Carroll, Ted Briscoe, and Antonio Sanfilippo. 1998. Parser evaluation: a survey and a new proposal. In *Proceedings of LREC*, pages 447–454, Granada, Spain, May.

Daniel Cer, Marie-Catherine de Marneffe, Daniel Jurafsky, and Christopher D. Manning. 2010. Parsing to stanford dependencies: Trade-offs between speed and accuracy. In *Proceedings of LREC*, Valletta, Malta, May.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of ACL*, pages 173–180, Ann Arbor, Michigan, USA, June.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of NAACL*, pages 132–139, Seattle, Washington, USA, April.

Stephen Clark and Julia Hockenmaier. 2002. Evaluating a wide-coverage ccg parser. In *Proceedings of the LREC Beyond Parseval Workshop*, Las Palmas, Canary Islands, Spain, May.

Andrew B. Clegg and Adrian J. Shepherd. 2005. Evaluating and integrating treebank parsers on a biomedical corpus. In *Proceedings of the ACL Workshop on Software*, pages 14–33, Ann Arbor, Michigan, USA, June.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of ACL*, pages 16–23, Madrid, Spain, July.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of ICML*, pages 175–182, Palo Alto, California, USA, June.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4):589–637.

Rebecca Dridan and Stephan Oepen. 2011. Parser evaluation using elementary dependency matching. In *Proceedings of IWPT*, pages 225–230, Dublin, Ireland, October.

Aaron Dunlop, Nathan Bodenstab, and Brian Roark. 2011. Efficient matrix-encoded grammars and low latency parallelization strategies for cyk. In *Proceedings of IWPT*, pages 163–174, Dublin, Ireland, October.

Jennifer Foster and Josef van Genabith. 2008. Parser evaluation and the bnc: Evaluating 4 constituency parsers with 3 metrics. In *Proceedings of LREC*, Marrakech, Morocco, May.

Daniel Gildea. 2001. Corpus variation and parser performance. In *Proceedings of EMNLP*, pages 167–202, Pittsburgh, Pennsylvania, USA, June.

Tadayoshi Hara, Yusuke Miyao, and Jun'ichi Tsujii. 2007. Evaluating impact of re-training a lexical disambiguation model on domain adaptation of an hpsg parser. In *Proceedings of IWPT*, pages 11–22, Prague, Czech Republic, June.

Tadayoshi Hara, Yusuke Miyao, and Jun'ichi Tsujii. 2009. Descriptive and empirical approaches to capturing underlying dependencies among parsing errors. In *Proceedings of EMNLP*, pages 1162–1171, Singapore, August.

James Henderson. 2003. Inducing history representations for broad coverage statistical parsing. In *Proceedings of NAACL*, pages 24–31, Edmonton, Canada, May.

James Henderson. 2004. Discriminative training of a neural network statistical parser. In *Proceedings of ACL*, pages 95–102, Barcelona, Spain, July.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL*, pages 586–594, Columbus, Ohio, USA, June.

Tracy H. King, Richard Crouch, Stefan Riezler, Mary Dalrymple, and Ronald M. Kaplan. 2003. The PARC 700 dependency bank. In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora at EACL*, Budapest, Hungary, April.

Dan Klein and Christopher D. Manning. 2003a. Accurate unlexicalized parsing. In *Proceedings of ACL*, pages 423–430, Sapporo, Japan, July.

Dan Klein and Christopher D. Manning. 2003b. Fast exact inference with a factored model for natural language parsing. In *Proceedings of NIPS*, pages 3–10, Vancouver, British Columbia, Canada, December.

Dekang Lin. 1998. A dependency-based method for evaluating broad-coverage parsers. *Natural Language Engineering*, 4(2):97–114.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: the penn treebank. *Computational Linguistics*, 19(2):313–330.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *Proceedings of NAACL*, pages 152–159, New York, New York, USA, June.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of EMNLP*, pages 122–131, Prague, Czech Republic, June.

Yusuke Miyao, Rune Sætre, Kenji Sagae, Takuya Matsuzaki, and Jun'ichi Tsujii. 2008. Task-oriented evaluation of syntactic parsers and their representations. In

Proceedings of ACL, pages 46–54, Columbus, Ohio, USA, June.

Dominick Ng and James R. Curran. 2012. N-best CCG parsing and reranking. In *Proceedings of ACL*, Jeju, South Korea, July.

Dominick Ng, Matthew Honnibal, and James R. Curran. 2010. Reranking a wide-coverage ccg parser. In *Proceedings of ALTA*, pages 90–98, Melbourne, Australia, December.

Joakim Nivre, Laura Rimell, Ryan McDonald, and Carlos Gómez-Rodríguez. 2010. Evaluation of dependency parsers on unbounded dependencies. In *Proceedings of Coling*, pages 833–841, Beijing, China, August.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proceedings of NAACL*, pages 404–411, Rochester, New York, USA, April.

Slav Petrov and Ryan McDonald. 2012. SANCL Shared Task. LDC2012E43. Linguistic Data Consortium. Philadelphia, Philadelphia, USA.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of ACL*, pages 433–440, Sydney, Australia, July.

Chris Quirk and Simon Corston-Oliver. 2006. The impact of parse quality on syntactically-informed statistical machine translation. In *Proceedings of EMNLP*, pages 62–69, Sydney, Australia, July.

Kun Yu, Yusuke Miyao, Takuya Matsuzaki, Xiangli Wang, and Junichi Tsujii. 2011. Analysis of the difficulties in chinese deep parsing. In *Proceedings of IWPT*, pages 48–57, Dublin, Ireland, October.

Deniz Yuret, Aydin Han, and Zehra Turgut. 2010. Semeval-2010 task 12: Parser evaluation using textual entailments. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 51–56, Uppsala, Sweden, July.