Evaluation of Dependency Parsers on Unbounded Dependencies

Joakim Nivre
Uppsala University
joakim.nivre@lingfil.uu.se

Laura Rimell
Univ. of Cambridge
laura.rimell@cl.cam.ac.uk

Ryan McDonald
Google Inc.
ryanmcd@google.com

Carlos Gómez-Rodríguez
Universidade da Coruña
cgomezr@udc.es

Abstract

We evaluate two dependency parsers, MSTParser and MaltParser, with respect to their capacity to recover unbounded dependencies in English, a type of evaluation that has been applied to grammar-based parsers and statistical phrase structure parsers but not to dependency parsers. The evaluation shows that when combined with simple post-processing heuristics, the parsers correctly recall unbounded dependencies roughly 50% of the time, which is only slightly worse than two grammar-based parsers specifically designed to cope with such dependencies.

1 Introduction

Though syntactic parsers for English are reported to have accuracies over 90% on the Wall Street Journal (WSJ) section of the Penn Treebank (PTB) (McDonald et al., 2005; Sagae and Lavie, 2006; Huang, 2008; Carreras et al., 2008), broad-coverage parsing is still far from being a solved problem. In particular, metrics like attachment score for dependency parsers (Buchholz and Marsi, 2006) and Parseval for constituency parsers (Black et al., 1991) suffer from being an average over a highly skewed distribution of different grammatical constructions. As a result, infrequent yet semantically important construction types could be parsed with accuracies far below what one might expect.

This shortcoming of aggregate parsing metrics was highlighted in a recent study by Rimell et al. (2009), introducing a new parser evaluation corpus containing around 700 sentences annotated with unbounded dependencies in seven different grammatical constructions. This corpus was used to evaluate five state-of-the-art parsers for English, focusing on grammar-based and statistical phrase structure parsers. For example, in the sentence *By Monday, they hope to have a sheaf of* **documents** *both sides can* **trust**., parsers should recognize that there is a dependency between **trust** and **documents**, an instance of object extraction out of a (reduced) relative clause. In the evaluation, the recall of state-of-the-art parsers on this kind of dependency varies from a high of 65% to a low of 1%. When averaging over the seven constructions in the corpus, none of the parsers had an accuracy higher than 61%.

In this paper, we extend the evaluation of Rimell et al. (2009) to two dependency parsers, MSTParser (McDonald, 2006) and MaltParser (Nivre et al., 2006a), trained on data from the PTB, converted to Stanford typed dependencies (de Marneffe et al., 2006), and combined with a simple post-processor to extract unbounded dependencies from the basic dependency tree. Extending the evaluation to dependency parsers is of interest because it sheds light on whether highly tuned grammars or computationally expensive parsing formalisms are necessary for extracting complex linguistic phenomena in practice. Unlike the best performing grammar-based parsers studied in Rimell et al. (2009), neither MSTParser nor MaltParser was developed specifically as a parser for English, and neither has any special mechanism for dealing with unbounded dependencies. Dependency parsers are also often asymptotically faster than grammar-based or constituent parsers, e.g., MaltParser parses sentences in linear time.

Our evaluation ultimately shows that the recall of MSTParser and MaltParser on unbounded dependencies is much lower than the average (un)labeled attachment score for each system. Nevertheless, the two dependency parsers are found to perform only slightly worse than the best grammar-based parsers evaluated in Rimell et al.

Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010), pages 833–841,
Beijing, August 2010

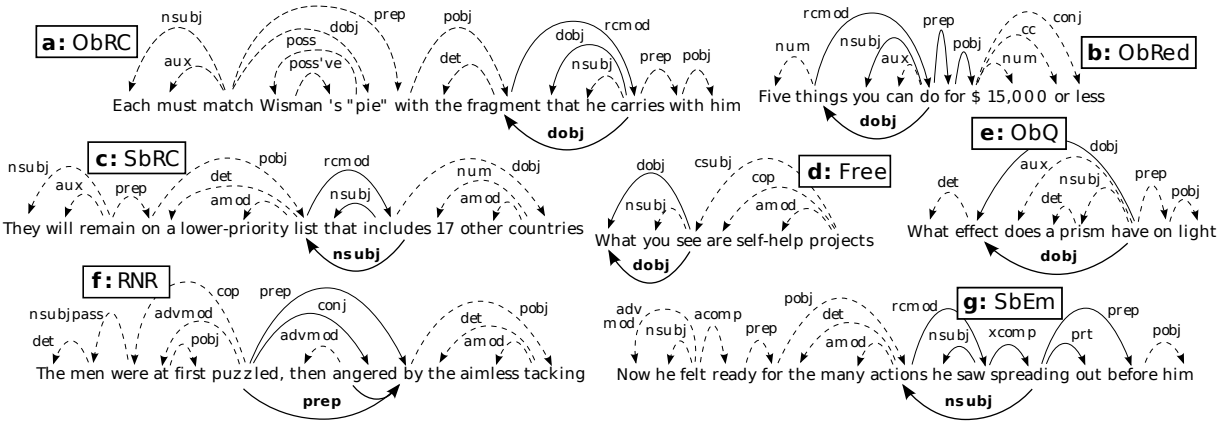

Figure 1: Examples of seven unbounded dependency constructions (**a**–**g**). Arcs drawn *below* each sentence represent the dependencies scored in the evaluation, while the tree *above* each sentence is the Stanford basic dependency representation, with solid arcs indicating crucial dependencies (cf. Section 4). All examples are from the development sets.

(2009) and considerably better than the other statistical parsers in that evaluation. Interestingly, though the two systems have similar accuracies overall, there is a clear distinction between the kinds of errors each system makes, which we argue is consistent with observations by McDonald and Nivre (2007).

2 Unbounded Dependency Evaluation

An unbounded dependency involves a word or phrase interpreted at a distance from its surface position, where an unlimited number of clause boundaries may in principle intervene. The unbounded dependency corpus of Rimell et al. (2009) includes seven grammatical constructions: object extraction from a relative clause (ObRC), object extraction from a reduced relative clause (ObRed), subject extraction from a relative clause (SbRC), free relatives (Free), object questions (ObQ), right node raising (RNR), and subject extraction from an embedded clause (SbEm), all chosen for being relatively frequent and easy to identify in PTB trees. Examples of the constructions can be seen in Figure 1. The evaluation set contains 80 sentences per construction (which may translate into more than 80 dependencies, since sentences containing coordinations may have more than one gold-standard dependency), while the development set contains between 13 and 37 sentences per construction. The data for ObQ sentences was obtained from various years of TREC, and for the rest of the constructions from the WSJ (0-1 and 22-24) and Brown sections of the PTB.

Each sentence is annotated with one or more gold-standard dependency relations representing the relevant unbounded dependency. The gold-standard dependencies are shown as arcs below the sentences in Figure 1. The format of the dependencies in the corpus is loosely based on the Stanford typed dependency scheme, although the evaluation procedure permits alternative representations and does not require that the parser output match the gold-standard exactly, as long as the "spirit" of the construction is correct.

The ability to recover unbounded dependencies is important because they frequently form part of the basic predicate-argument structure of a sentence. Subject and object dependencies in particular are crucial for a number of tasks, including information extraction and question answering. Moreover, Rimell et al. (2009) show that, although individual types of unbounded dependencies may be rare, the unbounded dependency types in the corpus, considered as a class, occur in as many as 10% of sentences in the PTB.

In Rimell et al. (2009), five state-of-the-art parsers were evaluated for their recall on the gold-standard dependencies. Three of the parsers were based on grammars automatically extracted from the PTB: the C&C CCG parser (Clark and Curran, 2007), the Enju HPSG parser (Miyao and Tsujii, 2005), and the Stanford parser (Klein and Manning, 2003). The two remaining systems were the

RASP parser (Briscoe et al., 2006), using a manually constructed grammar and a statistical parse selection component, and the DCU post-processor of PTB parsers (Cahill et al., 2004) using the output of the Charniak and Johnson reranking parser (Charniak and Johnson, 2005). Because of the wide variation in parser output representations, a mostly manual evaluation was performed to ensure that each parser got credit for the constructions it recovered correctly. The parsers were run essentially "out of the box", meaning that the development set was used to confirm input and output formats, but no real tuning was performed. In addition, since a separate question model is available for C&C, this was also evaluated on ObQ sentences. The best overall performers were C&C and Enju, which is unsurprising since they are deep parsers based on grammar formalisms designed to recover just such dependencies. The DCU post-processor performed somewhat worse than expected, often identifying the existence of an unbounded dependency but failing to identify the grammatical class (subject, object, etc.). RASP and Stanford, although not designed to recover such dependencies, nevertheless recovered a subset of them. Performance of the parsers also varied widely across the different constructions.

3 Dependency Parsers

In this paper we repeat the study of Rimell et al. (2009) for two dependency parsers, with the goal of evaluating how parsers based on dependency grammars perform on unbounded dependencies.

MSTParser[1] is a freely available implementation of the parsing models described in McDonald (2006). According to the categorization of parsers in Kübler et al. (2008) it is a *graph-based* parsing system in that core parsing algorithms can be equated to finding directed maximum spanning trees (either projective or non-projective) from a dense graph representation of the sentence. Graph-based parsers typically rely on global training and inference algorithms, where the goal is to learn models in which the weight/probability of correct trees is higher than that of incorrect trees. At inference time a global search is run to find the highest weighted dependency tree. Unfortunately, global inference and learning for graph-based dependency parsing is typically NP-hard (McDonald and Satta, 2007). As a result, graph-based parsers (including MSTParser) often limit the scope of their features to a small number of adjacent arcs (usually two) and/or resort to approximate inference (McDonald and Pereira, 2006).

MaltParser[2] is a freely available implementation of the parsing models described in Nivre et al. (2006a) and Nivre et al. (2006b). MaltParser is categorized as a *transition-based* parsing system, characterized by parsing algorithms that produce dependency trees by transitioning through abstract state machines (Kübler et al., 2008). Transition-based parsers learn models that predict the next state given the current state of the system as well as features over the history of parsing decisions and the input sentence. At inference time, the parser starts in an initial state, then greedily moves to subsequent states – based on the predictions of the model – until a termination state is reached. Transition-based parsing is highly efficient, with run-times often linear in sentence length. Furthermore, transition-based parsers can easily incorporate arbitrary non-local features, since the current parse structure is fixed by the state. However, the greedy nature of these systems can lead to error propagation if early predictions place the parser in incorrect states.

McDonald and Nivre (2007) compared the accuracy of MSTParser and MaltParser along a number of structural and linguistic dimensions. They observed that, though the two parsers exhibit indistinguishable accuracies overall, MSTParser tends to outperform MaltParser on longer dependencies as well as those dependencies closer to the root of the tree (e.g., verb, conjunction and preposition dependencies), whereas MaltParser performs better on short dependencies and those further from the root (e.g., pronouns and noun dependencies). Since long dependencies and those near to the root are typically the last constructed in transition-based parsing systems, it was concluded that MaltParser does suffer from some form of error propagation. On the other hand, the

[1] http://mstparser.sourceforge.net

[2] http://www.maltparser.org

richer feature representations of MaltParser led to improved performance in cases where error propagation has not occurred. However, that study did not investigate unbounded dependencies.

4 Methodology

In this section, we describe the methodological setup for the evaluation, including parser training, post-processing, and evaluation.[3]

4.1 Parser Training

One important difference between MSTParser and MaltParser, on the one hand, and the best performing parsers evaluated in Rimell et al. (2009), on the other, is that the former were never developed specifically as parsers for English. Instead, they are best understood as data-driven parser generators, that is, tools for generating a parser given a training set of sentences annotated with dependency structures. Over the years, both systems have been applied to a wide range of languages (see, e.g., McDonald et al. (2006), McDonald (2006), Nivre et al. (2006b), Hall et al. (2007), Nivre et al. (2007)), but they come with no language-specific enhancements and are not equipped specifically to deal with unbounded dependencies.

Since the dependency representation used in the evaluation corpus is based on the Stanford typed dependency scheme (de Marneffe et al., 2006), we opted for using the WSJ section of the PTB, converted to Stanford dependencies, as our primary source of training data. Thus, both parsers were trained on section 2–21 of the WSJ data, which we converted to Stanford dependencies using the Stanford parser (Klein and Manning, 2003). The Stanford scheme comes in several varieties, but because both parsers require the dependency structure for each sentence to be a tree, we had to use the so-called *basic* variety (de Marneffe et al., 2006).

It is well known that questions are very rare in the WSJ data, and Rimell et al. (2009) found that parsers trained only on WSJ data generally performed badly on the questions included in the evaluation corpus, while the C&C parser equipped with a model trained on a combination of WSJ and question data had much better performance. To investigate whether the performance of MSTParser and MaltParser on questions could also be improved by adding more questions to the training data, we trained one variant of each parser using data that was extended with 3924 questions taken from QuestionBank (QB) (Judge et al., 2006).[4] Since the QB sentences are annotated in PTB style, it was possible to use the same conversion procedure as for the WSJ data. However, it is clear that the conversion did not always produce adequate dependency structures for the questions, an observation that we will return to in the error analysis below.

In comparison to the five parsers evaluated in Rimell et al. (2009), it is worth noting that MSTParser and MaltParser were trained on the same basic data as four of the five, but with a different kind of syntactic representation – dependency trees instead of phrase structure trees or theory-specific representations from CCG and HPSG. It is especially interesting to compare MSTParser and MaltParser to the Stanford parser, which essentially produces the same kind of dependency structures as output but uses the original phrase structure trees from the PTB as input to training.

For our experiments we used MSTParser with the same parsing algorithms and features as reported in McDonald et al. (2006). However, unlike that work we used an atomic maximum entropy model as the second stage arc predictor as opposed to the more time consuming sequence labeler. McDonald et al. (2006) showed that there is negligible accuracy loss when using atomic rather than structured labeling. For MaltParser we used the projective Stack algorithm (Nivre, 2009) with default settings and a slightly enriched feature model. All parsing was projective because the Stanford dependency trees are strictly projective.

[3] To ensure replicability, we provide all experimental settings, post-processing scripts and additional information about the evaluation at http://stp.ling.uu.se/∼nivre/exp/.

[4] QB contains 4000 questions, but we removed all questions that also occurred in the test or development set of Rimell et al. (2009), who sampled their questions from the same TREC QA test sets.

4.2 Post-Processing

All the development and test sets in the corpus of Rimell et al. (2009) were parsed using MSTParser and MaltParser after part-of-speech tagging the input using SVMTool (Giménez and Màrquez, 2004) trained on section 2–21 of the WSJ data in Stanford basic dependency format. The Stanford parser has an internal module that converts the *basic* dependency representation to the *collapsed* representation, which explicitly represents additional dependencies, including unbounded dependencies, that can be inferred from the basic representation (de Marneffe et al., 2006). We performed a similar conversion using our own tool.

Broadly speaking, there are three ways in which unbounded dependencies can be inferred from the Stanford basic dependency trees, which we will refer to as *simple*, *complex*, and *indirect*. In the simple case, the dependency coincides with a single, direct dependency relation in the tree. This is the case, for example, in Figure 1**d**–**e**, where all that is required is that the parser identifies the dependency relation from a governor to an argument (`dobj(see, What)`, `dobj(have, effect)`), which we call the Arg relation; no post-processing is needed.

In the complex case, the dependency is represented by a *path* of direct dependencies in the tree, as exemplified in Figure 1**a**. In this case, it is not enough that the parser correctly identifies the Arg relation `dobj(carries, that)`; it must also find the dependency `rcmod(fragment, carries)`. We call this the Link relation, because it links the argument role inside the relative clause to an element outside the clause. Other examples of the complex case are found in Figure 1**c** and in Figure 1**f**.

In the indirect case, finally, the dependency cannot be defined by a path of labeled dependencies, whether simple or complex, but must be inferred from a larger context of the tree using heuristics. Consider Figure 1**b**, where there is a Link relation (`rcmod(things, do)`), but no corresponding Arg relation inside the relative clause (because there is no overt relative pronoun). However, given the other dependencies, we can infer with high probability that the implicit relation is `dobj`. Another example of the indirect case is in Figure 1**g**. Our post-processing tool performs more heuristic inference for the indirect case than the Stanford parser does (cf. Section 4.3).

In order to handle the complex and indirect cases, our post-processor is triggered by the occurrence of a Link relation (`rcmod` or `conj`) and first tries to add dependencies that are directly implied by a single Arg relation (relations involving relative pronouns for `rcmod`, shared heads and dependents for `conj`). If there is no overt relative pronoun, or the function of the relative pronoun is underspecified, the post-processor relies on the obliqueness hierarchy `subj` $<$ `dobj` $<$ `pobj` and simply picks the first "missing function", unless it finds a clausal complement (indicated by the labels `ccomp` and `xcomp`), in which case it descends to the lower clause and restarts the search there.

4.3 Parser Evaluation

The evaluation was performed using the same criteria as in Rimell et al. (2009). A dependency was considered correctly recovered if the gold-standard head and dependent were correct and the label was an "acceptable match" to the gold-standard label, indicating the grammatical function of the extracted element at least to the level of subject, passive subject, object, or adjunct.

The evaluation in Rimell et al. (2009) took into account a wide variety of parser output formats, some of which differed significantly from the gold-standard. Since MSTParser and MaltParser produced Stanford dependencies for this experiment, evaluation required less manual examination than for some of the other parsers, as was also the case for the output of the Stanford parser in the original evaluation. However, a manual evaluation was still performed in order to resolve questionable cases.

5 Results

The results are shown in Table 1, where the accuracy for each construction is the percentage of gold-standard dependencies recovered correctly. The *Avg* column represents a macroaverage, i.e. the average of the individual scores on the seven constructions, while the *WAvg* column represents

Parser	ObRC	ObRed	SbRC	Free	ObQ	RNR	SbEm	Avg	WAvg
MST	34.1	47.3	78.9	65.5	13.8	**45.4**	**37.6**	46.1	63.4
Malt	**40.7**	**50.5**	**84.2**	**70.2**	16.2	39.7	23.5	46.4	**66.9**
MST-Q					**41.2**			**50.0**	
Malt-Q					31.2			48.5	

Table 1: Parser accuracy on the unbounded dependency corpus.

Parser	ObRC	ObRed	SbRC	Free	ObQ	RNR	SbEm	Avg	WAvg
C&C	**59.3**	62.6	80.0	72.6	**81.2**	**49.4**	22.4	**61.1**	69.9
Enju	47.3	**65.9**	82.1	**76.2**	32.5	47.1	32.9	54.9	**70.9**
MST	34.1	47.3	78.9	65.5	41.2	45.4	**37.6**	50.0	63.4
Malt	40.7	50.5	**84.2**	70.2	31.2	39.7	23.5	48.5	66.9
DCU	23.1	41.8	56.8	46.4	27.5	40.8	5.9	34.6	47.0
RASP	16.5	1.1	53.7	17.9	27.5	34.5	15.3	23.8	34.1
Stanford	22.0	1.1	74.7	64.3	41.2	45.4	10.6	37.0	50.3

Table 2: Parser accuracy on the unbounded dependency corpus. The ObQ score for C&C, MSTParser, and MaltParser is for a model trained with additional questions (without this C&C scored 27.5; MSTParser and MaltParser as in Table 1).

a weighted macroaverage, where the constructions are weighted proportionally to their relative frequency in the PTB. WAvg excludes ObQ sentences, since frequency statistics were not available for this construction in Rimell et al. (2009).

Our first observation is that the accuracies for both systems are considerably below the ∼90% unlabeled and ∼88% labeled attachment scores for English that have been reported previously (McDonald and Pereira, 2006; Hall et al., 2006). Comparing the two parsers, we see that MaltParser is more accurate on dependencies in relative clause constructions (ObRC, ObRed, SbRC, and Free), where argument relations tend to be relatively local, while MSTParser is more accurate on dependencies in RNR and SbEm, which involve more distant relations. Without the additional QB training data, the average scores for the two parsers are indistinguishable, but MSTParser appears to have been better able to take advantage of the question training, since MST-Q performs better than Malt-Q on ObQ sentences. On the weighted average MaltParser scores 3.5 points higher, because the constructions on which it outperforms MSTParser are more frequent in the PTB, and because WAvg excludes ObQ, where MSTParser is more accurate.

Table 2 shows the results for MSTParser and MaltParser in the context of the other parsers evaluated in Rimell et al. (2009).[5] For the parsers which have a model trained on questions, namely C&C, MSTParser, and MaltParser, the figure shown for ObQ sentences is that of the question model. It can be seen that MSTParser and MaltParser perform below C&C and Enju, but above the other parsers, and that MSTParser achieves the highest score on SbEm sentences and MaltParser on SbRC sentences. It should be noted, however, that Table 2 does not represent a direct comparison across all parsers, since most of the other parsers would have benefited from heuristic post-processing of the kind implemented here for MSTParser and MaltParser. This is especially true for RASP, where the grammar explicitly leaves some types of attachment decisions for post-processing. For DCU, improved labeling heuristics would significantly improve performance. It is instructive to compare the dependency parsers to the Stanford parser, which uses the same output representation and has been used to prepare the training data for our experiments. Stanford has very low recall on ObRed and SbEm, the categories where heuristic inference plays the largest role, but mirrors MSTParser for most other categories.

6 Error Analysis

We now proceed to a more detailed error analysis, based on the development sets, and classify

[5] The average scores reported differ slightly from those in Rimell et al. (2009), where a microaverage (i.e., average over all dependencies in the corpus, regardless of construction) was reported.

the errors made by the parsers into three categories: A *global* error is one where the parser completely fails to build the relevant clausal structure – the relative clause in ObRC, ObRed, SbRC, Free, SbEmb; the interrogative clause in ObQ; and the clause headed by the higher conjunct in RNR – often as a result of surrounding parsing errors. When a global error occurs, it is usually meaningless to further classify the error, which means that this category excludes the other two. An Arg error is one where the parser has constructed the relevant clausal structure but fails to find the Arg relation – in the simple and complex cases – or the set of surrounding Arg relations needed to infer an implicit Arg relation – in the indirect case (cf. Section 4.2). A Link error is one where the parser fails to find the crucial Link relation – `rcmod` in ObRC, ObRed, SbRC, SbEmb; `conj` in RNR (cf. Section 4.2). Link errors are not relevant for Free and ObQ, where all the crucial relations are clause-internal.

Table 3 shows the frequency of different error types for MSTParser (first) and MaltParser (second) in the seven development sets. First of all, we can see that the overall error distribution is very similar for the two parsers, which is probably due to the fact that they have been trained on exactly the same data with exactly the same annotation (unlike the five parsers previously evaluated). However, there is a tendency for MSTParser to make fewer Link errors, especially in the relative clause categories ObRC, ObRed and SbRC, which is compatible with the observation from the test results that MSTParser does better on more global dependencies, while MaltParser has an advantage on more local dependencies, although this is not evident from the statistics from the relatively small development set.

Comparing the different grammatical constructions, we see that Link errors dominate for the relative clause categories ObRC, ObRed and SbRC, where the parsers make very few errors with respect to the internal structure of the relative clauses (in fact, no errors at all for MaltParser on SbRC). This is different for SbEm, where the analysis of the argument structure is more complex, both because there are (at least) two clauses involved and because the unbounded dependency

Type	Global	Arg	Link	A+L	Errors	# Deps
ObRC	0/1	1/1	7/11	5/3	13/16	20
ObRed	0/1	0/1	6/7	3/4	9/13	23
SbRC	2/1	1/0	7/13	0/0	10/14	43
Free	2/1	3/5	–	–	5/6	22
ObQ	4/7	13/13	–	–	17/20	25
RNR	6/4	4/6	0/0	4/5	14/15	28
SbEm	3/4	3/2	0/0	3/3	9/9	13

Table 3: Distribution of error types in the development sets; frequencies for MSTParser listed first and MaltParser second. The columns Arg and Link give frequencies for Arg/Link errors occurring without the other error type, while A+L give frequencies for joint Arg and Link errors.

can only be inferred indirectly from the basic dependency representation (cf. Section 4.2). Another category where Arg errors are frequent is RNR, where all such errors consist in attaching the relevant dependent to the second conjunct instead of to the first.[6] Thus, in the example in Figure 1**f**, both parsers found the `conj` relation between **puzzled** and **angered** but attached **by** to the second verb.

Global errors are most frequent for RNR, probably indicating that coordinate structures are difficult to parse in general, and for ObQ (especially for MaltParser), probably indicating that questions are not well represented in the training set even after the addition of QB data.[7] As noted in Section 4.1, this may be partly due to the fact that conversion to Stanford dependencies did not seem to work as well for QB as for the WSJ data. Another problem is that the part-of-speech tagger used was trained on WSJ data only and did not perform as well on the ObQ data. Uses of *What* as a determiner were consistently mistagged as pronouns, which led to errors in parsing. Thus, for the example in Figure 1**e**, both parsers produced the correct analysis except that, because of the tagging error, they treated **What** rather than **effect** as the head of the *wh*-phrase, which counts as an error in the evaluation.

In order to get a closer look specifically at the Arg errors, Table 4 gives the confusion matrix

[6] In the Stanford scheme, an argument or adjunct must be attached to the first conjunct in a coordination to indicate that it belongs to both conjuncts.

[7] Parsers trained without QB had twice as many global errors.

	Sb	Ob	POb	EmSb	EmOb	Other	Total
Sb	–	0/0	0/0	0/0	0/0	2/1	2/1
Ob	2/3	–	0/0	0/1	0/0	4/2	6/6
POb	2/0	7/5	–	0/0	0/0	5/8	14/13
EmSb	1/1	4/2	0/0	–	0/0	1/2	6/5
EmOb	0/0	3/1	0/0	0/0	–	1/6	4/7
Total	5/4	14/8	0/0	0/1	0/0	13/19	32/32

Table 4: Confusion matrix for Arg errors (excluding RNR and using parsers trained on QB for ObQ); frequencies for MSTParser listed first and MaltParser second. The column Other covers errors where the function is left unspecified or the argument is attached to the wrong head.

for such errors, showing which grammatical functions are mistaken for each other, with an extra category Other for cases where the function is left unspecified by the parser or the error is an attachment error rather than a labeling error (and excluding the RNR category because of the special nature of the Arg errors in this category). The results again confirm that the two parsers make very few errors on subjects and objects clause-internally. The few cases where an object is mistaken as a subject occur in ObQ, where both parsers perform rather poorly in general. By contrast, there are many more errors on prepositional objects and on embedded subjects and objects. We believe an important part of the explanation for this pattern is to be found in the Stanford dependency representation, where subjects and objects are marked as such but all other functions realized by *wh* elements are left unspecified (using the generic *rel* dependency), which means that the recovery of these functions currently has to rely on heuristic rules as described in Section 4.2. Finally, we think it is possible to observe the tendency for MaltParser to be more accurate at local labeling decisions – reflected in fewer cross-label confusions – and for MSTParser to perform better on more distant attachment decisions – reflected in fewer errors in the Other category (and in fewer Link errors).

7 Conclusion

In conclusion, the capacity of MSTParser and MaltParser to recover unbounded dependencies is very similar on the macro and weighted macro level, but there is a clear distinction in their strengths – constructions involving more distant dependencies such as ObQ, RNR and SbEm for MSTParser and constructions with more locally defined configurations such as ObRC, ObRed, SbRC and Free for MaltParser. This is a pattern that has been observed in previous evaluations of the parsers and can be explained by the global learning and inference strategy of MSTParser and the richer feature space of MaltParser (McDonald and Nivre, 2007).

Perhaps more interestingly, the accuracies of MSTParser and MaltParser are only slightly below the best performing systems in Rimell et al. (2009) – C&C and Enju. This is true even though MSTParser and MaltParser have not been engineered specifically for English and lack special mechanisms for handling unbounded dependencies, beyond the simple post-processing heuristic used to extract them from the output trees. Thus, it is reasonable to speculate that the addition of such mechanisms could lead to computationally lightweight parsers with the ability to extract unbounded dependencies with high accuracy.

Acknowledgments

We thank Marie-Catherine de Marneffe for great help with the Stanford parser and dependency scheme, Lluís Màrquez and Jesús Giménez for great support with SVMTool, Josef van Genabith for sharing the QuestionBank data, and Stephen Clark and Mark Steedman for helpful comments on the evaluation process and the paper. Laura Rimell was supported by EPSRC grant EP/E035698/1 and Carlos Gómez-Rodríguez by MEC/FEDER (HUM2007-66607-C04) and Xunta de Galicia (PGIDIT07SIN005206PR, Redes Galegas de PL e RI e de Ling. de Corpus, Bolsas Estadas INCITE/FSE cofinanced).

References

Black, E., S. Abney, D. Flickinger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of 4th DARPA Workshop*, 306–311.

Briscoe, T., J. Carroll, and R. Watson. 2006. The second release of the RASP system. In *Proceedings*

of the COLING/ACL 2006 Interactive Presentation Sessions, 77–80.

Buchholz, S. and E. Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of CoNLL*, 149–164.

Cahill, A., M. Burke, R. O'Donovan, J. Van Genabith, and A. Way. 2004. Long-distance dependency resolution in automatically acquired wide-coverage PCFG-based LFG approximations. In *Proceedings of ACL*, 320–327.

Carreras, X., M. Collins, and T. Koo. 2008. TAG, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proceedings of CoNLL*, 9–16.

Charniak, E. and M. Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of ACL*, 173–180.

Clark, S. and J. R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33:493–552.

de Marneffe, M.-C., B. MacCartney, and C. D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*.

Giménez, J. and L. Màrquez. 2004. SVMTool: A general POS tagger generator based on support vector machines. In *Proceedings of LREC*.

Hall, J., J. Nivre, and J. Nilsson. 2006. Discriminative classifiers for deterministic dependency parsing. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, 316–323.

Hall, J., J. Nilsson, J. Nivre, G. Eryiğit, B. Megyesi, M. Nilsson, and M. Saers. 2007. Single malt or blended? A study in multilingual parser optimization. In *Proceedings of the CoNLL Shared Task*.

Huang, L. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL*, 586–594.

Judge, J., A. Cahill, and J. van Genabith. 2006. QuestionBank: Creating a corpus of parse-annotated questions. In *Proceedings of COLING-ACL*, 497–504.

Klein, D. and C. D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of ACL*, 423–430.

Kübler, S., R. McDonald, and J. Nivre. 2008. *Dependency Parsing*. Morgan and Claypool.

McDonald, R. and J. Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of EMNLP-CoNLL*, 122–131.

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

McDonald, R. and G. Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *Proceedings of IWPT*, 122–131.

McDonald, R., K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of ACL*, 91–98.

McDonald, R., K. Lerman, and F. Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of CoNLL*, 216–220.

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

Miyao, Y. and J. Tsujii. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of ACL*, 83–90.

Nivre, J., J. Hall, and J. Nilsson. 2006a. MaltParser: A data-driven parser-generator for dependency parsing. In *Proceedings of LREC*, 2216–2219.

Nivre, J., J. Hall, J. Nilsson, G. Eryiğit, and S. Marinov. 2006b. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of CoNLL*, 221–225.

Nivre, J., J. Hall, J. Nilsson, A. Chanev, G. Eryiğit, S. Kübler, S. Marinov, and E. Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13:95–135.

Nivre, J. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of ACL-IJCNLP*, 351–359.

Rimell, L., S. Clark, and M. Steedman. 2009. Unbounded dependency recovery for parser evaluation. In *Proceedings EMNLP*, 813–821.

Sagae, K. and A. Lavie. 2006. Parser combination by reparsing. In *Proceedings of NAACL HLT: Short Papers*, 129–132.