Characterizing the Errors of Data-Driven Dependency Parsing Models

Ryan McDonald
Google Inc.
76 Ninth Avenue
New York, NY 10011
ryanmcd@google.com

Joakim Nivre
Växjö University 35195 Växjö Sweden
Uppsala University 75126 Uppsala Sweden
nivre@msi.vxu.se

Abstract

We present a comparative error analysis of the two dominant approaches in data-driven dependency parsing: global, exhaustive, graph-based models, and local, greedy, transition-based models. We show that, in spite of similar performance overall, the two models produce different types of errors, in a way that can be explained by theoretical properties of the two models. This analysis leads to new directions for parser development.

1 Introduction

Syntactic dependency representations have a long history in descriptive and theoretical linguistics and many formal models have been advanced (Hudson, 1984; Mel'čuk, 1988; Sgall et al., 1986; Maruyama, 1990). A dependency graph of a sentence represents each word and its syntactic modifiers through labeled directed arcs, as shown in Figure 1, taken from the Prague Dependency Treebank (Böhmová et al., 2003). A primary advantage of dependency representations is that they have a natural mechanism for representing discontinuous constructions, arising from long distance dependencies or free word order, through *non-projective* dependency arcs, exemplified by the arc from *jedna* to *Z* in Figure 1.

Syntactic dependency graphs have recently gained a wide interest in the computational linguistics community and have been successfully employed for many problems ranging from machine translation (Ding and Palmer, 2004) to ontology construction (Snow et al., 2004). In this work we focus on a common parsing paradigm called *data-driven dependency parsing*. Unlike grammar-based parsing, data-driven approaches learn to produce dependency graphs for sentences solely from an annotated corpus. The advantage of such models is that they are easily ported to any domain or language in which annotated resources exist.

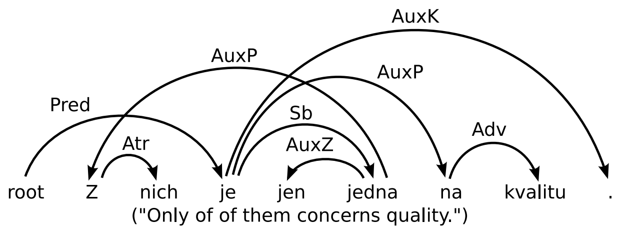

Figure 1: Example dependency graph.

As evident from the CoNLL-X shared task on dependency parsing (Buchholz and Marsi, 2006), there are currently two dominant models for data-driven dependency parsing. The first is what Buchholz and Marsi (2006) call the "all-pairs" approach, where every possible arc is considered in the construction of the optimal parse. The second is the "stepwise" approach, where the optimal parse is built stepwise and where the subset of possible arcs considered depend on previous decisions. Theoretically, these models are extremely different. The all-pairs models are globally trained, use exact (or near exact) inference algorithms, and define features over a limited history of parsing decisions. The stepwise models use local training and greedy inference algorithms, but define features over a rich history of parse decisions. However, both models obtain similar parsing accuracies

	McDonald	Nivre
Arabic	66.91	66.71
Bulgarian	87.57	87.41
Chinese	85.90	86.92
Czech	80.18	78.42
Danish	84.79	84.77
Dutch	79.19	78.59
German	87.34	85.82
Japanese	90.71	91.65
Portuguese	86.82	87.60
Slovene	73.44	70.30
Spanish	82.25	81.29
Swedish	82.55	84.58
Turkish	63.19	65.68
Overall	80.83	80.75

Table 1: Labeled parsing accuracy for top scoring systems at CoNLL-X (Buchholz and Marsi, 2006).

on a variety of languages, as seen in Table 1, which shows results for the two top performing systems in the CoNLL-X shared task, McDonald et al. (2006) ("all-pairs") and Nivre et al. (2006) ("stepwise").

Despite the similar performance in terms of overall accuracy, there are indications that the two types of models exhibit different behaviour. For example, Sagae and Lavie (2006) displayed that combining the predictions of both parsing models can lead to significantly improved accuracies. In order to pave the way for new and better methods, a much more detailed error analysis is needed to understand the strengths and weaknesses of different approaches. In this work we set out to do just that, focusing on the two top performing systems from the CoNLL-X shared task as representatives of the two dominant models in data-driven dependency parsing.

2 Two Models for Dependency Parsing

2.1 Preliminaries

Let $L = \{l_1, \ldots, l_{|L|}\}$ be a set of permissible arc labels. Let $x = w_0, w_1, \ldots, w_n$ be an input sentence where w_0=*root*. Formally, a dependency graph for an input sentence x is a labeled directed graph $G = (V, A)$ consisting of a set of nodes V and a set of labeled directed arcs $A \subseteq V \times V \times L$, i.e., if $(i, j, l) \in A$ for $i, j \in V$ and $l \in L$, then there is an arc from node i to node j with label l in the graph. A dependency graph G for sentence x must satisfy the following properties:

1. $V = \{0, 1, \ldots, n\}$
2. If $(i, j, l) \in A$, then $j \neq 0$.
3. If $(i, j, l) \in A$, then for all $i' \in V - \{i\}$ and $l' \in L$, $(i', j, l') \notin A$.
4. For all $j \in V - \{0\}$, there is a (possibly empty) sequence of nodes $i_1, \ldots, i_m \in V$ and labels $l_1, \ldots, l_m, l \in L$ such that $(0, i_1, l_1)$,(i_1, i_2, l_2), $\ldots$, $(i_m, j, l) \in A$.

The constraints state that the dependency graph spans the entire input (1); that the node 0 is a root (2); that each node has at most one incoming arc in the graph (3); and that the graph is connected through directed paths from the node 0 to every other node in the graph (4). A dependency graph satisfying these constraints is a directed tree originating out of the root node 0. We say that an arc (i, j, l) is *non-projective* if not all words k occurring between i and j in the linear order are dominated by i (where dominance is the transitive closure of the arc relation).

2.2 Global, Exhaustive, Graph-Based Parsing

For an input sentence, $x = w_0, w_1, \ldots, w_n$ consider the dense graph $G_x = (V_x, A_x)$ where:

1. $V_x = \{0, 1, \ldots, n\}$
2. $A_x = \{(i, j, l) \mid \forall\, i, j \in V_x \text{ and } l \in L\}$

Let $D(G_x)$ represent the subgraphs of graph G_x that are valid dependency graphs for the sentence x. Since G_x contains all possible labeled arcs, the set $D(G_x)$ must necessarily contain all valid dependency graphs for x.

Assume that there exists a dependency arc scoring function, $s : V \times V \times L \rightarrow \mathbb{R}$. Furthermore, define the score of a graph as the sum of its arc scores,

$$s(G = (V, A)) = \sum_{(i,j,l) \in A} s(i, j, l)$$

The score of a dependency arc, $s(i, j, l)$ represents the likelihood of creating a dependency from word w_i to word w_j with the label l. If the arc score function is known a priori, then the parsing problem can be stated as,

$$G = \underset{G \in D(G_x)}{\arg\max}\, s(G) = \underset{G \in D(G_x)}{\arg\max} \sum_{(i,j,l) \in A} s(i,j,l)$$

This problem is equivalent to finding the highest scoring directed spanning tree in the graph G_x originating out of the root node 0, which can be solved for both the labeled and unlabeled case in $O(n^2)$ time (McDonald et al., 2005b). In this approach, non-projective arcs are produced naturally through the inference algorithm that searches over all possible directed trees, whether projective or not.

The parsing models of McDonald work primarily in this framework. To learn arc scores, these models use large-margin structured learning algorithms (McDonald et al., 2005a), which optimize the parameters of the model to maximize the score margin between the correct dependency graph and all incorrect dependency graphs for every sentence in a training set. The learning procedure is global since model parameters are set relative to the classification of the entire dependency graph, and not just over single arc attachment decisions. The primary disadvantage of these models is that the feature representation is restricted to a limited number of graph arcs. This restriction is required so that both inference and learning are tractable.

The specific model studied in this work is that presented by McDonald et al. (2006), which factors scores over pairs of arcs (instead of just single arcs) and uses near exhaustive search for unlabeled parsing coupled with a separate classifier to label each arc. We call this system MSTParser, which is also the name of the freely available implementation.[1]

2.3 Local, Greedy, Transition-Based Parsing

A *transition system* for dependency parsing defines

1. a set C of *parser configurations*, each of which defines a (partially built) dependency graph G
2. a set T of *transitions*, each a function $t : C \rightarrow C$
3. for every sentence $x = w_0, w_1, \ldots, w_n$,
   (a) a unique *initial* configuration c_x
   (b) a set C_x of *terminal* configurations

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

A *transition sequence* $C_{x,m} = (c_x, c_1, \ldots, c_m)$ for a sentence x is a sequence of configurations such that $c_m \in C_x$ and, for every c_i ($c_i \neq c_x$), there is a transition $t \in T$ such that $c_i = t(c_{i-1})$. The dependency graph assigned to x by $C_{x,m}$ is the graph G_m defined by the terminal configuration c_m.

Assume that there exists a transition scoring function, $s : C \times T \rightarrow \mathbb{R}$. The score of a transition t in a configuration c, $s(c,t)$, represents the likelihood of taking transition t out of configuration c. The parsing problem consists in finding a terminal configuration $c_m \in C_x$, starting from the initial configuration c_x and taking the optimal transition $t^* = \arg\max_{t \in T} s(c,t)$ out of every configuration c. This can be seen as a greedy search for the optimal dependency graph, based on a sequence of locally optimal decisions in terms of the transition system.

Many transition systems for data-driven dependency parsing are inspired by shift-reduce parsing, where configurations contain a stack for storing partially processed nodes. Transitions in such systems add arcs to the dependency graph and/or manipulate the stack. One example is the transition system defined by Nivre (2003), which parses a sentence $x = w_0, w_1, \ldots, w_n$ in $O(n)$ time, producing a projective dependency graph satisfying conditions 1–4 in section 2.1, possibly after adding arcs $(0, i, l_r)$ for every node $i \neq 0$ that is a root in the output graph (where l_r is a special label for root modifiers). Nivre and Nilsson (2005) showed how the restriction to projective dependency graphs could be lifted by using graph transformation techniques to preprocess training data and post-process parser output, so-called *pseudo-projective parsing*.

To learn transition scores, these systems use discriminative learning methods, e.g., memory-based learning or support vector machines. The learning procedure is local since only single transitions are scored, not entire transition sequences. The primary advantage of these models is that features are not restricted to a limited number of graph arcs but can take into account the entire dependency graph built so far. The main disadvantage is that the greedy parsing strategy may lead to error propagation.

The specific model studied in this work is that presented by Nivre et al. (2006), which uses labeled pseudo-projective parsing with support vector machines. We call this system MaltParser, which is also the name of the freely available implementation.[2]

2.4 Comparison

These models differ primarily with respect to three important properties.

1. **Inference:** MaltParser uses a transition-based inference algorithm that greedily chooses the best parsing decision based on a trained classifier and current parser history. MSTParser instead uses near exhaustive search over a dense graphical representation of the sentence to find the dependency graph that maximizes the score.

2. **Training:** MaltParser trains a model to make a single classification decision (choose the next transition). MSTParser trains a model to maximize the global score of correct graphs.

3. **Feature Representation:** MaltParser can introduce a rich feature history based on previous parser decisions. MSTParser is forced to restrict the score of features to a single or pair of nearby parsing decisions in order to make exhaustive inference tractable.

These differences highlight an inherent trade-off between exhaustive inference algorithms plus global learning and expressiveness of feature representations. MSTParser favors the former at the expense of the latter and MaltParser the opposite.

3 The CoNLL-X Shared Task

The CoNLL-X shared task (Buchholz and Marsi, 2006) was a large-scale evaluation of data-driven dependency parsers, with data from 13 different languages and 19 participating systems. The official evaluation metric was the *labeled attachment score* (LAS), defined as the percentage of tokens, excluding punctuation, that are assigned both the correct head and the correct dependency label.[3]

The output of all systems that participated in the shared task are available for download and constitute a rich resource for comparative error analysis. The data used in the experiments below are the outputs of MSTParser and MaltParser for all 13 languages, together with the corresponding gold standard graphs used in the evaluation. We constructed the data by simply concatenating a system's output for every language. This resulted in a single output file for each system and a corresponding single gold standard file. This method is sound because the data sets for each language contain approximately the same number of tokens – 5,000. Thus, evaluating system performance over the aggregated files can be roughly viewed as measuring system performance through an equally weighted arithmetic mean over the languages.

It could be argued that a language by language comparison would be more appropriate than comparing system performance across all languages. However, as table Table 1 shows, the difference in accuracy between the two systems is typically small for all languages, and only in a few cases is this difference significant. Furthermore, by aggregating over all languages we gain better statistical estimates of parser errors, since the data set for each individual language is very small.

4 Error Analysis

The primary purpose of this study is to characterize the errors made by standard data-driven dependency parsing models. To that end, we present a large set of experiments that relate parsing errors to a set of linguistic and structural properties of the input and predicted/gold standard dependency graphs. We argue that the results can be correlated to specific theoretical aspects of each model – in particular the trade-off highlighted in Section 2.4.

For simplicity, all experiments report labeled parsing accuracies. Identical experiments using unlabeled parsing accuracies did not reveal any additional information. Furthermore, all experiments are based on the data from all 13 languages together, as explained in section 3.

4.1 Length Factors

It is well known that parsing systems tend to have lower accuracies for longer sentences. Figure 2 shows the accuracy of both parsing models relative to sentence length (in bins of size 10: 1–10, 11–20,

[2] http://w3.msi.vxu.se/users/nivre/research/MaltParser.html

[3] In addition, results were reported for *unlabeled attachment score* (UAS) (tokens with the correct head) and *label accuracy* (LA) (tokens with the correct label).

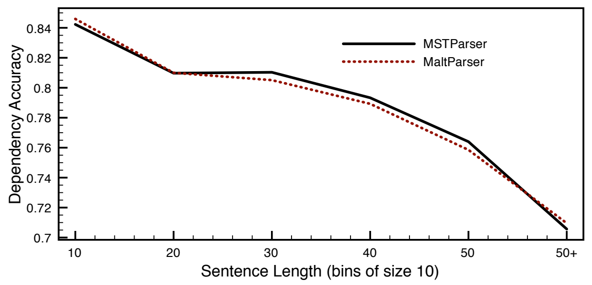

Figure 2: Accuracy relative to sentence length.

etc.). System performance is almost indistinguishable. However, MaltParser tends to perform better on shorter sentences, which require the greedy inference algorithm to make less parsing decisions. As a result, the chance of error propagation is reduced significantly when parsing these sentences. The fact that MaltParser has a higher accuracy (rather than the same accuracy) when the likelihood of error propagation is reduced comes from its richer feature representation.

Another interesting property is accuracy relative to dependency length. The length of a dependency from word w_i to word w_j is simply equal to $|i - j|$. Longer dependencies typically represent modifiers of the root or the main verb in a sentence. Shorter dependencies are often modifiers of nouns such as determiners or adjectives or pronouns modifying their direct neighbours. Figure 3 measures the precision and recall for each system relative to dependency lengths in the predicted and gold standard dependency graphs. Precision represents the percentage of predicted arcs of length d that were correct. Recall measures the percentage of gold standard arcs of length d that were correctly predicted.

Here we begin to see separation between the two systems. MSTParser is far more precise for longer dependency arcs, whereas MaltParser does better for shorter dependency arcs. This behaviour can be explained using the same reasoning as above: shorter arcs are created before longer arcs in the greedy parsing procedure of MaltParser and are less prone to error propagation. Theoretically, MSTParser should not perform better or worse for edges of any length, which appears to be the case. There is still a slight degradation, but this can be attributed to long dependencies occurring more frequently in constructions with possible ambiguity. Note that even though the area under the curve is much larger for MSTParser, the number of dependency arcs with a length greater than ten is much smaller than the number with length less than ten, which is why the overall accuracy of each system is nearly identical. For all properties considered here, bin size generally shrinks in size as the value on the x-axis increases.

4.2 Graph Factors

The structure of the predicted and gold standard dependency graphs can also provide insight into the differences between each model. For example, measuring accuracy for arcs relative to their distance to the artificial root node will detail errors at different levels of the dependency graph. For a given arc, we define this distance as the number of arcs in the reverse path from the modifier of the arc to the root. Figure 4 plots the precision and recall of each system for arcs of varying distance to the root. Precision is equal to the percentage of dependency arcs in the predicted graph that are at a distance of d and are correct. Recall is the percentage of dependency arcs in the gold standard graph that are at a distance of d and were predicted.

Figure 4 clearly shows that for arcs close to the root, MSTParser is much more precise than MaltParser, and vice-versa for arcs further away from the root. This is probably the most compelling graph given in this study since it reveals a clear distinction: MSTParser's precision degrades as the distance to the root increases whereas MaltParser's precision increases. The plots essentially run in opposite directions crossing near the middle. Dependency arcs further away from the root are usually constructed early in the parsing algorithm of MaltParser. Again a reduced likelihood of error propagation coupled with a rich feature representation benefits that parser substantially. Furthermore, MaltParser tends to overpredict root modifiers, because all words that the parser fails to attach as modifiers are automatically connected to the root, as explained in section 2.3. Hence, low precision for root modifiers (without a corresponding drop in recall) is an indication that the transition-based parser produces fragmented parses.

The behaviour of MSTParser is a little trickier to explain. One would expect that its errors should be distributed evenly over the graph. For the most part this is true, with the exception of spikes at the ends

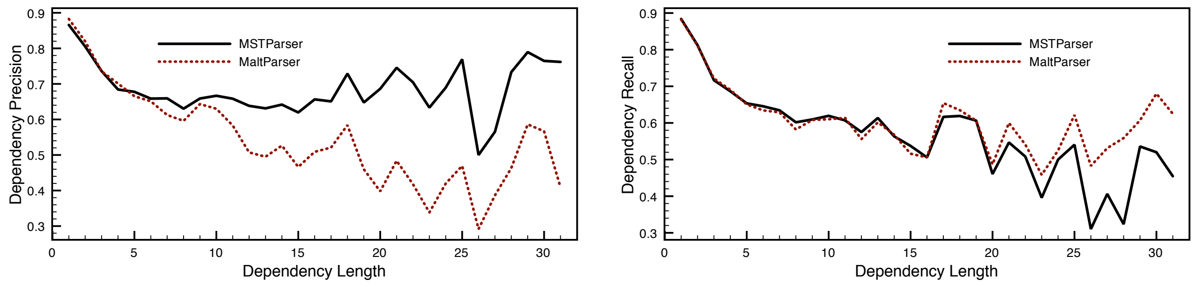

Figure 3: Dependency arc precision/recall relative to predicted/gold dependency length.

of the plot. The high performance for root modification (distance of 1) can be explained through the fact that this is typically a low entropy decision – usually the parsing algorithm has to determine the main verb from a small set of possibilities. On the other end of the plot there is a sharp downwards spike for arcs of distance greater than 10. It turns out that MSTParser over-predicts arcs near the bottom of the graph. Whereas MaltParser pushes difficult parsing decisions higher in the graph, MSTParser appears to push these decisions lower.

The next graph property we will examine aims to quantify the local neighbourhood of an arc within a dependency graph. Two dependency arcs, (i, j, l) and (i', j', l') are classified as siblings if they represent syntactic modifications of the same word, i.e., $i = i'$. Figure 5 measures the precision and recall of each system relative to the number of predicted and gold standard siblings of each arc. There is not much to distinguish between the parsers on this metric. MSTParser is slightly more precise for arcs that are predicted with more siblings, whereas MaltParser has slightly higher recall on arcs that have more siblings in the gold standard tree. Arcs closer to the root tend to have more siblings, which ties this result to the previous ones.

The final graph property we wish to look at is the degree of non-projectivity. The degree of a dependency arc from word w to word u is defined here as the number of words occurring between w and u that are not descendants of w and modify a word that does not occur between w and u (Nivre, 2006). In the example from Figure 1, the arc from *jedna* to *Z* has a degree of one, and all other arcs have a degree of zero. Figure 6 plots dependency arc precision and recall relative to arc degree in predicted and gold standard dependency graphs. MSTParser is more precise when predicting arcs with high degree and MaltParser vice-versa. Again, this can be explained by the fact that there is a tight correlation between a high degree of non-projectivity, dependency length, distance to root and number of siblings.

4.3 Linguistic Factors

It is important to relate each system's accuracy to a set of linguistic categories, such as parts of speech and dependency types. Therefore, we have made an attempt to distinguish a few broad categories that are cross-linguistically identifiable, based on the available documentation of the treebanks used in the shared task.

For parts of speech, we distinguish *verbs* (including both main verbs and auxiliaries), *nouns* (including proper names), *pronouns* (sometimes also including determiners), *adjectives*, *adverbs*, *adpositions* (prepositions, postpositions), and *conjunctions* (both coordinating and subordinating). For dependency types, we distinguish a general *root* category (for labels used on arcs from the artificial root, including either a generic label or the label assigned to predicates of main clauses, which are normally verbs), a *subject* category, an *object* category (including both direct and indirect objects), and various categories related to *coordination*.

Figure 7 shows the accuracy of the two parsers for different parts of speech. This figure measures labeled dependency accuracy relative to the part of speech of the modifier word in a dependency relation. We see that MaltParser has slightly better accuracy for nouns and pronouns, while MSTParser does better on all other categories, in particular conjunctions. This pattern is consistent with previous results insofar as verbs and conjunctions are often involved in dependencies closer to the root that span

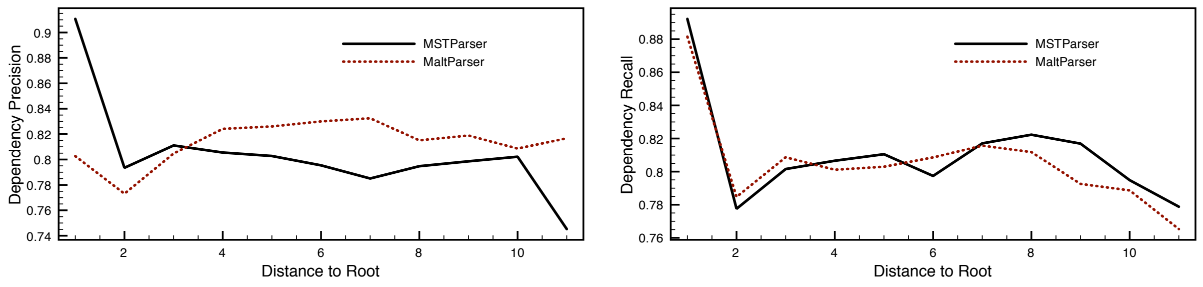

Figure 4: Dependency arc precision/recall relative to predicted/gold distance to root.

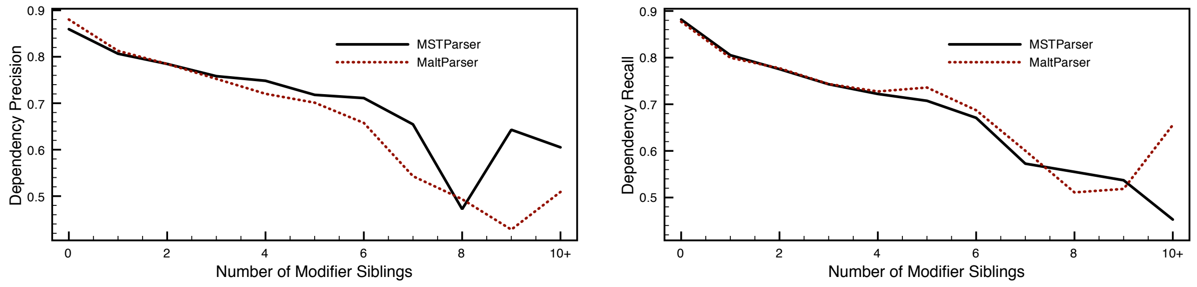

Figure 5: Dependency arc precision/recall relative to number of predicted/gold siblings.

longer distances, while nouns and pronouns are typically attached to verbs and therefore occur lower in the graph, with shorter distances. Empirically, adverbs resemble verbs and conjunctions with respect to root distance but group with nouns and pronouns for dependency length, so the former appears to be more important. In addition, both conjunctions and adverbs tend to have a high number of siblings, making the results consistent with the graph in Figure 5.

Adpositions and especially adjectives constitute a puzzle, having both high average root distance and low average dependency length. Adpositions do tend to have a high number of siblings on average, which could explain MSTParser's performance on that category. However, adjectives on average occur the furthest away from the root, have the shortest dependency length and the fewest siblings. As such, we do not have an explanation for this behaviour.

In the top half of Figure 8, we consider precision and recall for dependents of the root node (mostly verbal predicates), and for subjects and objects. As already noted, MSTParser has considerably better precision (and slightly better recall) for the root category, but MaltParser has an advantage for the nominal categories, especially subjects. A possible explanation for the latter result, in addition to the length-based and graph-based factors invoked before, is that

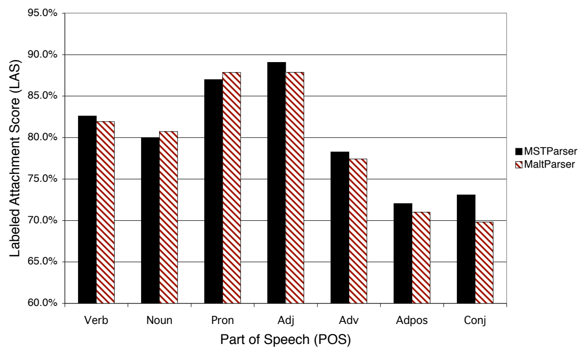

Figure 7: Accuracy for different parts of speech.

MaltParser integrates labeling into the parsing process, so that previously assigned dependency labels can be used as features, which may be important to disambiguate subjects and objects.

Finally, in the bottom half of Figure 8, we display precision and recall for coordinate structures, divided into different groups depending on the type of analysis adopted in a particular treebank. The category CCH (coordinating conjunction as head) contains conjunctions analyzed as heads of coordinate structures, with a special dependency label that does not describe the function of the coordinate structure in the larger syntactic structure, a type of category found in the so-called Prague style analysis of coordination and used in the data sets for Arabic, Czech,

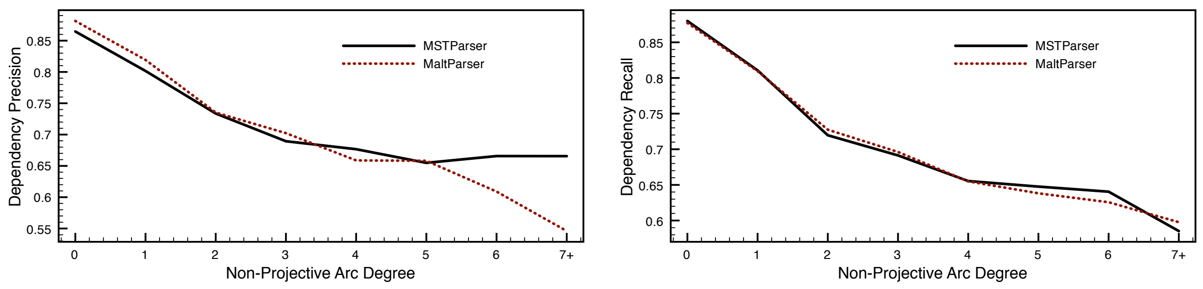

Figure 6: Dependency arc precision/recall relative to predicted/gold degree of non-projectivity.

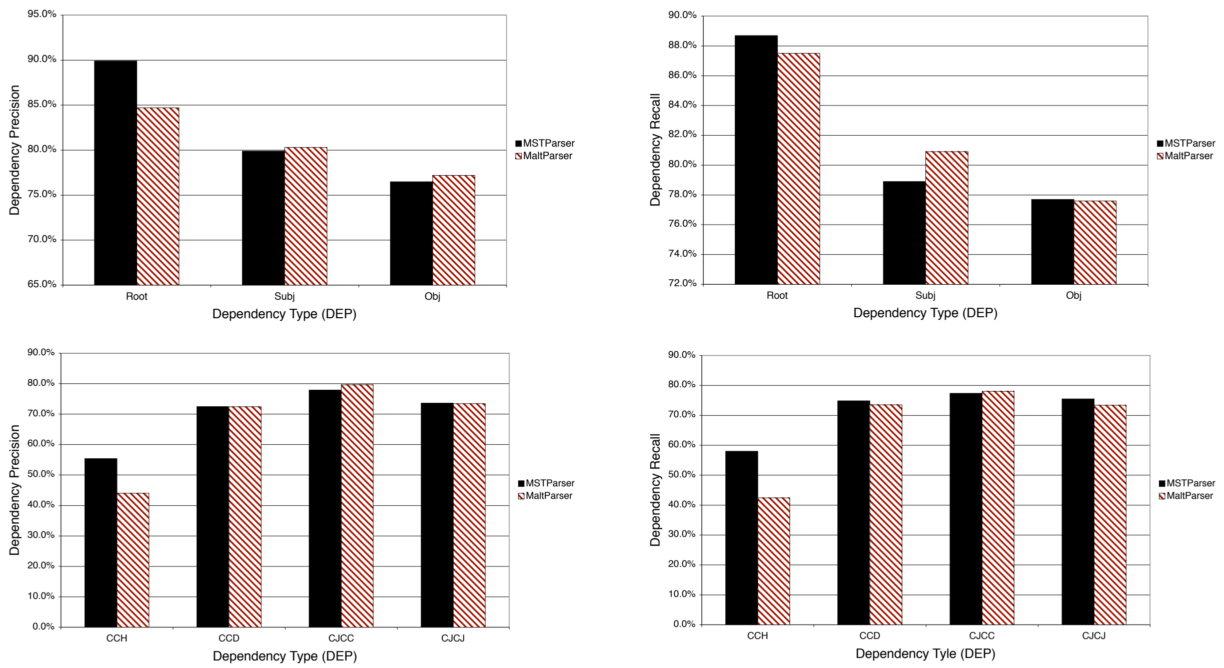

Figure 8: Precision/recall for different dependency types.

and Slovene. The category CCD (coordinating conjunction as dependent) instead denotes conjunctions that are attached as dependents of one of the conjuncts with a label that only marks them as conjunctions, a type of category found in the data sets for Bulgarian, Danish, German, Portuguese, Swedish and Turkish. The two remaining categories contain conjuncts that are assigned a dependency label that only marks them as conjuncts and that are attached either to the conjunction (CJCC) or to another conjunct (CJCJ). The former is found in Bulgarian, Danish, and German; the latter only in Portuguese and Swedish. For most of the coordination categories there is little or no difference between the two parsers, but for CCH there is a difference in both precision and recall of almost 20 percentage points to MSTParser's advantage. This can be explained by noting that, while the categories CCD, CJCC, and CJCJ denote relations that are *internal* to the coordinate structure and therefore tend to be local, the CCH relations hold between the coordinate structure and its head, which is often a relation that spans over a greater distance and is nearer the root of the dependency graph. It is likely that the difference in accuracy for this type of dependency accounts for a large part of the difference in accuracy noted earlier for conjunctions as a part of speech.

4.4 Discussion

The experiments from the previous section highlight the fundamental trade-off between global training and exhaustive inference on the one hand and expressive feature representations on the other. Error propagation is an issue for MaltParser, which typically performs worse on long sentences, long dependency arcs and arcs higher in the graphs. But this is offset by the rich feature representation available to these models that result in better decisions for frequently occurring arc types like short dependencies or subjects and objects. The errors for MSTParser are spread a little more evenly. This is expected, as the inference algorithm and feature representation should not prefer one type of arc over another.

What has been learned? It was already known that the two systems make different errors through the work of Sagae and Lavie (2006). However, in that work an arc-based voting scheme was used that took only limited account of the properties of the words connected by a dependency arc (more precisely, the overall accuracy of each parser for the part of speech of the dependent). The analysis in this work not only shows that the errors made by each system are different, but that they are different in a way that can be predicted and quantified. This is an important step in parser development.

To get some upper bounds of the improvement that can be obtained by combining the strengths of each models, we have performed two oracle experiments. Given the output of the two systems, we can envision an oracle that can optimally choose which single parse or combination of sub-parses to predict as a final parse. For the first experiment the oracle is provided with the single best parse from each system, say $G = (V, A)$ and $G' = (V', A')$. The oracle chooses a parse that has the highest number of correctly predicted labeled dependency attachments. In this situation, the oracle accuracy is 84.5%. In the second experiment the oracle chooses the tree that maximizes the number of correctly predicted dependency attachments, subject to the restriction that the tree must only contain arcs from $A \cup A'$. This can be computed by setting the weight of an arc to 1 if it is in the correct parse and in the set $A \cup A'$. All other arc weights are set to negative infinity. One can then simply find the tree that has maximal sum of arc weights using directed spanning tree algorithms. This technique is similar to the parser voting methods used by Sagae and Lavie (2006). In this situation, the oracle accuracy is 86.9%.

In both cases we see a clear increase in accuracy: 86.9% and 84.5% relative to 81% for the individual systems. This indicates that there is still potential for improvement, just by combining the two existing models. More interestingly, however, we can use the analysis to get ideas for new models. Below we sketch some possible new directions:

1. **Ensemble systems:** The error analysis presented in this paper could be used as inspiration for more refined weighting schemes for ensemble systems of the kind proposed by Sagae and Lavie (2006), making the weights depend on a range of linguistic and graph-based factors.

2. **Hybrid systems:** Rather than using an ensemble of several parsers, we may construct a single system integrating the strengths of each parser described here. This could defer to a greedy inference strategy during the early stages of the parse in order to benefit from a rich feature representation, but then default to a global exhaustive model as the likelihood for error propagation increases.

3. **Novel approaches:** The two approaches investigated are each based on a particular combination of training and inference methods. We may naturally ask what other combinations may prove fruitful. For example, what about globally trained, greedy, transition-based models? This is essentially what Daumé III et al. (2006) provide, in the form of a general search-based structured learning framework that can be directly applied to dependency parsing. The advantage of this method is that the learning can set model parameters relative to errors resulting directly from the search strategy – such as error propagation due to greedy search. When combined with MaltParser's rich feature representation, this could lead to significant improvements in performance.

5 Conclusion

We have presented a thorough study of the difference in errors made between global exhaustive graph-based parsing systems (MSTParser) and local greedy transition-based parsing systems (MaltParser). We have shown that these differences can be quantified and tied to theoretical expectations of each model, which may provide insights leading to better models in the future.

References

A. Böhmová, J. Hajič, E. Hajičová, and B. Hladká. 2003. The PDT: A 3-level annotation scenario. In A. Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*, chapter 7. Kluwer Academic Publishers.

S. Buchholz and E. Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proc. CoNLL*.

Hal Daumé III, John Langford, and Daniel Marcu. 2006. Search-based structured prediction. In Submission.

Y. Ding and M. Palmer. 2004. Synchronous dependency insertion grammars: A grammar formalism for syntax based statistical MT. In *Workshop on Recent Advances in Dependency Grammars (COLING)*.

R. Hudson. 1984. *Word Grammar*. Blackwell.

H. Maruyama. 1990. Structural disambiguation with constraint propagation. In *Proc. ACL*.

R. McDonald, K. Crammer, and F. Pereira. 2005a. Online large-margin training of dependency parsers. In *Proc. ACL*.

R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proc. HLT/EMNLP*.

R. McDonald, K. Lerman, and F. Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proc. CoNLL*.

I.A. Mel'čuk. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press.

J. Nivre and J. Nilsson. 2005. Pseudo-projective dependency parsing. In *Proc. ACL*.

J. Nivre, J. Hall, J. Nilsson, G. Eryigit, and S. Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proc. CoNLL*.

J. Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proc. IWPT*.

J. Nivre. 2006. Constraints on non-projective dependency parsing. In *Proc. EACL*.

K. Sagae and A. Lavie. 2006. Parser combination by reparsing. In *Proc. HLT/NAACL*.

P. Sgall, E. Hajičová, and J. Panevová. 1986. *The Meaning of the Sentence in Its Pragmatic Aspects*. Reidel.

R. Snow, D. Jurafsky, and A. Y. Ng. 2004. Learning syntactic patterns for automatic hypernym discovery. In *Proc. NIPS*.