Reverse Revision and Linear Tree Combination for Dependency Parsing

Giuseppe Attardi
Dipartimento di Informatica
Università di Pisa
Pisa, Italy
attardi@di.unipi.it

Felice Dell'Orletta
Dipartimento di Informatica
Università di Pisa
Pisa, Italy
felice.dellorletta@di.unipi.it

1 Introduction

Deterministic *transition-based Shift/Reduce* dependency parsers make often mistakes in the analysis of long span dependencies (McDonald & Nivre, 2007).

Titov and Henderson (2007) address this accuracy drop by using a beam search instead of a greedy algorithm for predicting the next parser transition.

We propose a parsing method that allows reducing several of these errors, although maintaining a quasi linear complexity. The method consists in two steps: first the sentence is parsed by a deterministic *Shift/Reduce* parser, then a second deterministic *Shift/Reduce* parser analyzes the sentence in reverse using additional features extracted from the parse trees produced by the first parser.

Right-to-left parsing has been used as part of ensemble-based parsers (Sagae & Lavie, 2006; Hall et al., 2007). Nivre and McDonald (2008) instead use hints from one parse as features in a second parse, exploiting the complementary properties of graph-based parsers (Eisner, 1996; McDonald et al., 2005) and transition-based dependency parsers (Yamada & Matsumoto, 2003; Nivre & Scholz, 2004).

Also our method uses input from a previous parser but only uses parsers of a single type, deterministic transition-based *Shift/Reduce*, maintaining an overall linear complexity. In fact both the ensemble parsers and the stacking solution of Nivre-McDonald involve the computation of the maximum spanning tree (MST) of a graph, which require algorithms of quadratic time complexity (e.g. (Chu & Liu, 1965; Edmonds, 1967)).

We introduce an alternative linear combination method. The algorithm is greedy and works by combining the trees top down. We tested it on the dependency trees produced by three parsers, a *Left-to-Right* (*LR*), a *Right-to-Left* (*RL*) and a *stacked Right-to-Left* parser, or *Reverse Revision* parser (*Rev2*). [1] The experiments show that in practice its output often outperforms the results produced by calculating the MST.

2 Experiments

In the reported experiments we used *DeSR* (Attardi at al., 2007), a freely available implementation of a transition-based parser. The parser processes input tokens advancing on the input with *Shift* actions and accumulates processed tokens on a stack with *Reduce* actions. The parsing algorithm is fully deterministic and linear.

For the *LR* parser and the *Rev2* parser we employed an SVM classifier while a Maximum Entropy classifier, with lower accuracy, was used to create the training set for the *Rev2* parser. The reason for this appears to be that the output of a low accuracy parser with many errors provides a better source of learning to the stacked parser.

The *Rev2* parser exploits the same basic set of features as in the *LR* parser plus the additional features extracted from the output of the *LR* parser listed in Table 1, where: *PHLEMMA* is the lemma of the predicted head, *PHPOS* is the Part of Speech of the predicted head, *PDEP* is the predicted dependency label of a token to its predicted head, *PHDIST* indicates whether a token is located before or after

[1] The *stacked Left-to-Right* parser produced slightly worse results than *Rev2*.

Proceedings of NAACL HLT 2009: Short Papers, pages 261–264,
Boulder, Colorado, June 2009.

Feature	Tokens
PHHLEMMA	$w_0\ w_1$
PHDEP	$w_0\ w_1$
PHPOS	$s_0\ w_0\ w_1$
PHLEMMA	$s_0\ w_0\ w_1$
PDEP	$s_0\ w_0\ w_1$
PHDIST	$s_0\ w_0\ w_1$

Table 1: Additional features used in training the Revision parser.

its predicted head, *PHHLEMMA* is the lemma of the predicted grandparent and *PHDEP* is the predicted dependency label of the predicted head of a token to the predicted grandparent. s_0 refers to a token on top of the stack, w_i refers to word at the i-th relative position with respect to the current word and parsing direction. This feature model was used for all languages in our tests.

We present experiments and comparative error analysis on three representative languages from the CoNLL 2007 shared task (Nivre at al., 2007): Italian, Czech and English. We also report an evaluation on all thirteen languages of the CoNLL-X shared task (Buchholz & Marsi, 2006), for comparison with the results by Nivre and McDonald (2008).

Table 2 shows the Labeled Attachment Score (LAS), for the *Left-to-right* parser (*LR*), *Right-to-Left* (*RL*), *Reverse Revision* parser (*Rev2*), linear parser combination (Comb) and MST parser combination (CombMST).

Figure 1 and 2 present the accuracies of the *LR* and *Rev2* parsers for English relative to the dependency length and the length of sentences, respectively. For Czech and Italian the *RL* parser achieves higher accuracy than the *LR* parser and the *Rev2* parser even higher. The error analysis for Czech showed that the *Rev2* parser improves over the *LR* parser everywhere except in the Recall for dependencies of length between 10 and 14. Such an improvement has positive impact on the analysis of sentences longer than 10 tokens, like for Italian.

2.1 CoNLL-X Results

For direct comparison with the approach by Nivre and McDonald (2008), we present the results on the CoNLL-X corpora (Table 3): MST and MST$_{Malt}$ are the results achieved by the MST parser and the MST parser using hints from Maltparser, Malt and

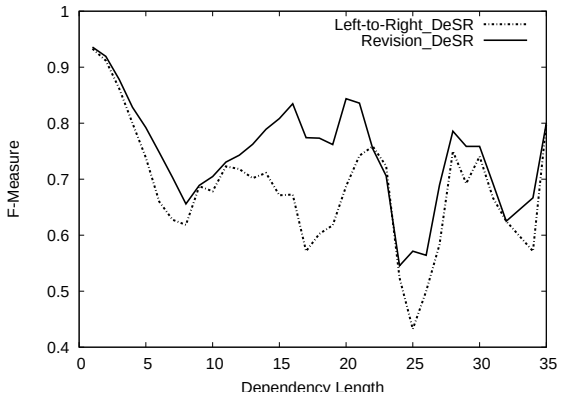

Figure 1: English. F-Measure relative to dependency length.

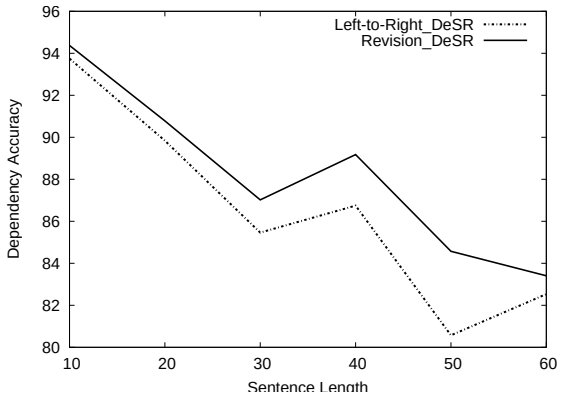

Figure 2: English. Accuracy relative to sentence length.

Malt$_{MST}$ the results of the opposite stacking.

2.2 Remarks

The *Rev2* parser, informed with data from the *LR* parser, achieves better accuracy in twelwe cases, statistically significantly better in eight.

The error analysis confirms that indeed the *Rev2* parser is able to reduce the number of errors made on long dependency links, which are a major weakness of a deterministic Shift/Reduce parser. The accuracy of the *Rev2* parser might be further improved by more sophisticated feature selection, choosing features that better represent hints to the second parsing stage.

3 Linear Voting Combination

Our final improvements arise by combining the outputs of the three parser models: the *LR* parser, the

Language	LR	RL	Rev2	Comb	CombMST	CoNLL 2007 Best
Czech	77.12	78.20	79.95	**80.57**	80.25	80.19
English	86.94	87.44	88.34	89.00	88.79	**89.61**
Italian	81.40	82.89	83.52	**84.56**	84.28	84.40

Table 2: LAS for selected CoNLL 2007 languages.

Language	**LR**	**RL**	**Rev2**	**Comb**	**CombMST**	**Conll-X Best**	**MST**	**MST**$_{Malt}$	**Malt**	**Malt**$_{MST}$
arabic	67.27	66.05	67.54	68.38	68.50	66.91	66.91	**68.64**	66.71	67.80
bulgarian	86.83	87.13	87.41	88.11	87.85	87.57	87.57	**89.05**	87.41	88.59
chinese	87.44	85.77	87.51	87.77	87.75	**89.96**	85.90	88.43	86.92	87.44
czech	79.84	79.46	81.78	82.22	82.22	80.18	80.18	**82.26**	78.42	81.18
danish	83.89	83.63	84.85	85.47	85.25	84.79	84.79	**86.67**	84.77	85.43
dutch	75.71	77.27	78.77	79.55	80.19	79.19	79.19	**81.63**	78.59	79.91
german	85.34	85.20	86.50	87.40	87.38	87.34	87.34	**88.46**	85.82	87.66
japanese	90.03	90.63	90.87	91.67	91.59	91.65	90.71	91.43	91.65	**92.20**
portuguese	86.84	87.00	87.86	88.14	88.20	87.60	86.82	87.50	87.60	**88.64**
slovene	73.64	74.40	75.32	75.72	75.48	73.44	73.44	**75.94**	70.30	74.24
spanish	81.63	81.61	81.85	83.33	83.13	82.25	82.25	**83.99**	81.29	82.41
swedish	82.95	81.62	82.91	83.69	83.69	84.58	82.55	**84.66**	84.58	84.31
turkish	64.91	61.92	63.33	65.27	65.23	65.68	63.19	64.29	65.58	**66.28**
Average	80.49	80.13	81.27	82.05	82.03	81.63	80.83	**82.53**	80.74	82.01

Table 3: Labeled attachment scores for CoNLL-X corpora.

RL parser and the *Rev2* parser.

Instead of using a general algorithm for calculating the MST of a graph, we exploit the fact that we are combining trees and hence we developed an approximate algorithm that has $O(kn)$ complexity, where n is the number of nodes in a tree and k is the number of trees being combined.

The algorithm builds the combined tree T incrementally, starting from the empty tree. We will argue that an invariant of the algorithm is that the partial result T is always a tree.

The algorithm exploits the notion of *fringe* F, i.e. the set of arcs whose parent is in T and that can be added to T without affecting the invariant. Initially F consists of the roots of all trees to be combined. The weight of each arc a in the fringe is the number of parsers that predicted a.

At each step, the algorithm selects from F an arc $a = (h, d, r)$ among those with maximum weight, having $h \in T$. Then it:

1. adds a to T
2. removes from F all arcs whose child is d
3. adds to F all arcs (h', d', r') in the original trees where $h' \in T$ and $d' \notin T$.

Step 3 guarantees that no cycles are present in T. The final T is connected because each added node is connected to a node in T. T is a local maximum because if there were another tree with higher score including arc (h, n, r), either it is present in T or its weight is smaller than the weight for node (h', n, r') in T, as chosen by the algorithm.

The algorithm has $O(kn)$ complexity. A sketch of the proof can be given as follows. Step 3 guarantees that the algorithm is iterated n times, where n is the number of nodes in a component tree. Using appropriate data structures to represent the *fringe* F, insert or delete operations take constant time. At each iteration of the algorithm the maximum number of removals from F (step 2) is constant and it is equal to k, hence the overall cost is $O(nk)$.

Table 2 shows the results for the three languages from CoNLL 2007. With respect to the best results at the CoNLL 2007 Shared Task, the linear parser combination achieves the best LAS for Czech and Italian, the second best for English.

The results for the CoNLL-X languages (Table 3) show also improvements: the *Rev2* parser is more

accurate than MST, except for Bulgarian, Dutch, German, and Spanish, where the difference is within 1%, and it is often better than the Malt$_{MST}$ stacking. The improvements of the *Rev2* over the *LR* parser range from 0.38% for Chinese to 3.84% for Dutch.

The column *CombMST* shows the results of combining parsers using the Chu-Liu-Edmonds MST algorithm and the same weighting scheme of Linear Combination algorithm. For most languages the Linear Combination algorithm leads to a better accuracy than the MST algorithm. The somewhat surprising result might be due indeed to the top down processing of the algorithm: since the algorithm chooses the best among the connections that are higher in the parse tree, this leads to a preference to long spanning links over shorter links even if these contribute higher weights to the MST.

Finally, the run time of the linear combination algorithm on the whole CoNLL-X test set is 11.2 sec, while the MST combination requires 92.5 sec.

We also tested weights based on the accuracy score of each parser for the POS of an arc head, but this produced less accurate results.

4 Conclusions

We presented a method for improving the accuracy of a dependency parser by using a parser that analyzes a sentence in reverse using hints from the trees produced by a forward parser.

We also introduced a new linear algorithm to perform parser combination.

Experiments on the corpora of languages from the CoNLL-X and the CoNLL 2007 shared tasks show that reverse revision parsing improves the accuracy over a transition-based dependency parser in all the tested languages. Further improvements are obtained by using a linear parser combination algorithm on the outputs of three parsers: a *LR* parser, a *RL* parser and a *Rev2* parser.

The combination parser achieves accuracies that are best or second best with respect to the results of the CoNLL 2007 shared task. Since all the individual parsers as well as the combination algorithm is linear, the combined parser maintains an overall linear computational time. On the languages from the CoNLL-X shared task the combination parser achieves often the best accuracy in ten out of thirteen languages but falls short of the accuracy achieved by integrating a graph-based with a transition based parser.

We expect that further tuning of the method might help reduce these differences.

References

G. Attardi, F. Dell'Orletta, M. Simi, A. Chanev and M. Ciaramita. 2007. Multilingual Dependency Parsing and Domain Adaptation using DeSR. In *Proc. of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*.

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

Y. J. Chu and T. H. Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*(14), 1396–1400.

J. Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards* (71B), 233–240.

J. M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proc. of COLING 1996*, 340–345.

J. Hall, et al. 2007. Single Malt or Blended? A Study in Multilingual Parser Optimization. In *Proc. of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*.

R. McDonald and J. Nivre. 2007. Characterizing the Errors of Data-Driven Dependency Parsing Models In *Proc. of EMNLP-CoNLL 2007*.

R. McDonald, F. Pereira, K. Ribarov and J. Hajič. 2005. Non-projective Dependency Parsing using Spanning Tree Algorithms. In *Proc. of HLT-EMNLP 2005*.

R. McDonald and F. Pereira. 2006. Online Learning of Approximate Dependency Parsing Algorithms. In *Proc. of EACL 2006*.

J. Nivre, et al. 2007. The CoNLL 2007 Shared Task on Dependency Parsing. In *Proc. of the CoNLL Shared Task Session of EMNLP/CoNLL-2007*.

J. Nivre and R. McDonald. 2008. Integrating Graph-Based and Transition-Based Dependency Parsers. In *Proc. of ACL 2008*.

J. Nivre and M. Scholz. 2004. Deterministic Dependency Parsing of English Text. In *Proc. of COLING 2004*.

K. Sagae and A. Lavie. 2006. Parser Combination by Reparsing. In *Proc. of HLT-NAACL 2006*.

I. Titov and J. Henderson. 2007. Fast and Robust Multilingual Dependency Parsing with a Generative Latent Variable Model In *Proc. of the CoNLL Shared Task Session of EMNLP/CoNNL-2007*.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis using support vector machines. In *Proc. of the 8th IWPT*. Nancy, France.