Tree Revision Learning for Dependency Parsing

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

Massimiliano Ciaramita
Yahoo! Research Barcelona
Barcelona, Spain
massi@yahoo-inc.com

Abstract

We present a revision learning model for improving the accuracy of a dependency parser. The revision stage corrects the output of the base parser by means of revision rules learned from the mistakes of the base parser itself. Revision learning is performed with a discriminative classifier. The revision stage has linear complexity and preserves the efficiency of the base parser. We present empirical evaluations on the treebanks of two languages, which show effectiveness in relative error reduction and state of the art accuracy.

1 Introduction

A dependency parse tree encodes useful semantic information for several language processing tasks. Dependency parsing is a simpler task than constituent parsing, since dependency trees do not have extra non-terminal nodes and there is no need for a grammar to generate them. Approaches to dependency parsing either generate such trees by considering all possible spanning trees (McDonald et al., 2005), or build a single tree on the fly by means of shift-reduce parsing actions (Yamada & Matsumoto, 2003). In particular, Nivre and Scholz (2004) and Attardi (2006) have developed deterministic dependency parsers with linear complexity, suitable for processing large amounts of text, as required, for example, in information retrieval applications.

We investigate a novel revision approach to dependency parsing related to re-ranking and transformation-based methods (Brill, 1993; Brill, 1995; Collins, 2000; Charniak & Johnson, 2005; Collins & Koo, 2006). Similarly to re-ranking, the second stage attempts to improve the output of a base parser. Instead of re-ranking n-best candidate parses, our method works by revising a single parse tree, either the $first$-best or the one constructed by a deterministic shift-reduce parser, as in transformation-based learning. Parse trees are revised by applying rules which replace incorrect with correct dependencies. These rules are learned by comparing correct parse trees with incorrect trees produced by the base parser on a training corpus. We use the same training corpus on which the base parser was trained, but this need not be the case. Hence, we define a new learning task whose output space is a set of revision rules and whose input is a set of features extracted at each node in the parse trees produced by the parser on the training corpus. A statistical classifier is trained to solve this task.

The approach is more suitable for dependency parsing since trees do not have non-terminal nodes, therefore revisions do not require adding/removing nodes. However, the method applies to any parser since it only analyzes output trees. An intuitive motivation for this method is the observation that a dependency parser correctly identifies most of the dependencies in a tree, and only local corrections might be necessary to produce a correct tree. Performing several parses in order to generate multiple trees would often just repeat the same steps. This could be avoided by focusing on the points where attachments are incorrect. In the experiments reported below, on average, the revision stage performs 4.28

Proceedings of NAACL HLT 2007, pages 388–395,
Rochester, NY, April 2007.

corrections per sentence, or one every 6.25 tokens.

In our implementation we adopt a shift-reduce parser which minimizes computational costs. The resulting two-stage parser has complexity $O(n)$, linear in the length of the sentence. We evaluated our model on the treebanks of English and Swedish. The experimental results show a relative error reduction of, respectively, 16% and 11% with respect to the base parser, achieving state of accuracy on Swedish.

2 Dependency parsing

Detection of dependency relations can be useful in tasks such as information extraction (Culotta & Sorensen, 2004), lexical acquisition (Snow et al., 2005), ontology learning (Ciaramita et al., 2005), and machine translation (Ding & Palmer, 2005). A dependency parser is trained on a corpus annotated with lexical dependencies, which are easier to produce by annotators without deep linguistic knowledge and are becoming available in many languages (Buchholz & Marsi, 2006). Recent developments in dependency parsing show that deterministic parsers can achieve good accuracy (Nivre & Scholz, 2004), and high performance, in the range of hundreds of sentences per second (Attardi, 2006).

A dependency parser takes as input a sentence s and returns a dependency graph G. Let $D = \{d_1, d_2, ..., d_m\}$ be the set of permissible dependency types. A *dependency graph* for a sentence $s = \langle s_1, s_2, ..., s_n \rangle$ is a labeled directed graph $G = (s, A)$, such that:

(a) s is the set of nodes, corresponding to the tokens in the input string;
(b) A is a set of labeled arcs (w_i, d, w_j), $w_{i,j} \in s$, $d \in D$; w_j is called the *head*, w_i the *modifier* and d the *dependency label*;
(c) $\forall w_i \in s$ there is at most one arc $a \in A$, such that $a = (w_i, d, w_j)$;
(d) there are no cycles;

In statistical parsing a generator (e.g. a PCFG) is used to produce a number of candidate trees (Collins, 2000) with associated scores. This approach has been used also for dependency parsing, generating spanning trees as candidates and computing the maximum spanning tree using discriminative learning algorithms (McDonald et al., 2005).

$$Shift \quad \frac{\langle S,n|I,T,A\rangle}{\langle n|S,I,T,A\rangle} \tag{1}$$

$$Right \quad \frac{\langle s|S,n|I,T,A\rangle}{\langle S,n|I,T,A\cup\{(s,r,n)\}\rangle} \tag{2}$$

$$Left \quad \frac{\langle s|S,n|I,T,A\rangle}{\langle S,s|I,T,A\cup\{(n,r,s)\}\rangle} \tag{3}$$

$$Right_2 \quad \frac{\langle s_1|s_2|S,n|I,T,A\rangle}{\langle s_1|S,n|I,T,A\cup\{(s_2,r,n)\}\rangle} \tag{4}$$

$$Left_2 \quad \frac{\langle s_1|s_2|S,n|I,T,A\rangle}{\langle s_2|S,s_1|I,T,A\cup\{(n,r,s_2)\}\rangle} \tag{5}$$

$$Right_3 \quad \frac{\langle s_1|s_2|s_3|S,n|I,T,A\rangle}{\langle s_1|s_2|S,n|I,T,A\cup\{(s_3,r,n)\}\rangle} \tag{6}$$

$$Left_3 \quad \frac{\langle s_1|s_2|s_3|S,n|I,T,A\rangle}{\langle s_2|s_3|S,s_1|I,T,A\cup\{(n,r,s_3)\}\rangle} \tag{7}$$

$$Extract \quad \frac{\langle s_1|s_2|S,n|I,T,A\rangle}{\langle n|s_1|S,I,s_2|T,A\rangle} \tag{8}$$

$$Insert \quad \frac{\langle S,I,s_1|T,A\rangle}{\langle s_1|S,I,T,A\rangle} \tag{9}$$

Table 1. The set of parsing rules of the base parser.

Yamada and Matsumoto (2003) have proposed an alternative approach, based on deterministic bottom-up parsing. Instead of learning directly which tree to assign to a sentence, the parser learns which *Shift/Reduce* actions to use for building the tree. Parsing is cast as a classification problem: at each step the parser applies a classifier to the features representing its current state to predict the next action to perform. Nivre and Scholz (2004) proposed a variant of the model of Yamada and Matsumoto that reduces the complexity from the worst case quadratic to linear. Attardi (2006) proposed a variant of the rules that allows deterministic single-pass parsing and as well as handling non-projective relations. Several approaches to dependency parsing on multiple languages have been evaluated in the CoNLL-X Shared Task (Buchholz & Marsi, 2006).

3 A shift-reduce dependency parser

As a base parser we use DeSR, a shift-reduce parser described in (Attardi, 2006). The parser constructs dependency trees by scanning input sentences in a single left-to-right pass and performing Shift/Reduce parsing actions. The parsing algorithm is fully deterministic and has linear complexity. Its behavior can be described as repeatedly selecting and applying some parsing rules to transform its state.

The state of the parser is represented by a quadru-

ple $\langle S, I, T, A\rangle$: S is the stack, I is the list of (remaining) input tokens, T is a stack of saved tokens and A is the arc relation for the dependency graph, consisting of a set of labeled arcs (w_i, r, w_j), $w_i, w_j \in W$ (the set of tokens), and $d \in D$ (the set of dependencies). Given an input sentence s, the parser is initialized to $\langle \emptyset, s, \emptyset, \emptyset\rangle$, and terminates when it reaches the configuration $\langle s, \emptyset, \emptyset, A\rangle$.

Table 1 lists all parsing rules. The $Shift$ rule advances on the input, while the various $Left$, $Right$ variants create links between the next input token and some previous token on the stack. $Extract/Insert$ generalize the previous rules by respectively moving one token to the stack T and reinserting the top of T into S. An essential difference with respect to the rules of Yamada and Matsumoto (2003) is that the $Right$ rules move back to the input the top of the stack, allowing some further processing on it, which would otherwise require a second pass. The extra $Left$ and $Right$ rules (4-7, Table 1), and the $ExtractInsert$ rules (8 and 9, Table 1), are new rules added for handling non-projective trees. The algorithm works as follows:

Algorithm 1: DeSR

input: $s = w_1, w_2, ..., w_n$
begin
  $S \leftarrow \langle\rangle$
  $I \leftarrow \langle w_1, w_2, ..., w_n\rangle$
  $T \leftarrow \langle\rangle$
  $A \leftarrow \langle\rangle$
  while $I \neq \langle\rangle$ **do**
    $\mathbf{x} \leftarrow getContext(S, I, T, A)$
    $y \leftarrow estimateAction(\mathbf{w}, \mathbf{x})$
    $performAction(y, S, I, T, A)$
end

The function $getContext()$ extracts a vector $\mathbf{x}$ of contextual features around the current token, i.e., from a subset of I and S. $estimateAction()$ predicts a parsing action y given a trained model $\mathbf{w}$ and $\mathbf{x}$. In the experiments presented below, we used as features the lemma, Part-of-Speech, and dependency type of the following items:

- 2 top items from S;
- 4 items from I;
- 2 leftmost and 2 rightmost children from the top of S and I.

Step	Description
r	Up to root node
u	Up one parent
$-n$	Left to the n-th token
$+n$	Right to the n-th token
$[$	Head of previous constituent
$]$	Head of following constituent
$>$	First token of previous constituent
$<$	First token of following constituent
$d--$	Down to the leftmost child
$d++$	Down to the rightmost child
$d-1$	Down to the first left child
$d+1$	Down to the first right child
dP	Down to token with POS P

Table 2. Description of the atomic movements allowed on the graph relatively to a token w.

4 Revising parse trees

The base parser is fairly accurate and even when there are mistakes most sentence chunks are correct. The full correct parse tree can often be recovered by performing just a small number of revisions on the base parse. We propose to learn these revisions and to apply them to the single best tree output by the base parser. Such an approach preserves the deterministic nature of the parser, since revising the tree requires a second sequential step over the whole sentence. The second step may also improve accuracy by incorporating additional evidence, gathered from the analysis of the tree which is not available during the first stage of parsing.

Our approach introduces a second learning task in which a model is trained to revise parse trees. Several questions needs to be addressed: which tree transformations to use in revising the parse tree, how to determine which transformation to apply, in which order, and which features to use for learning.

4.1 Basic graph movements

We define a revision as a combination of atomic moves on a graph; e.g., moving a link to the following or preceding token in the sentence, up or down the graph following the directed edges. Table 2 summarizes the set of atomic steps we used.

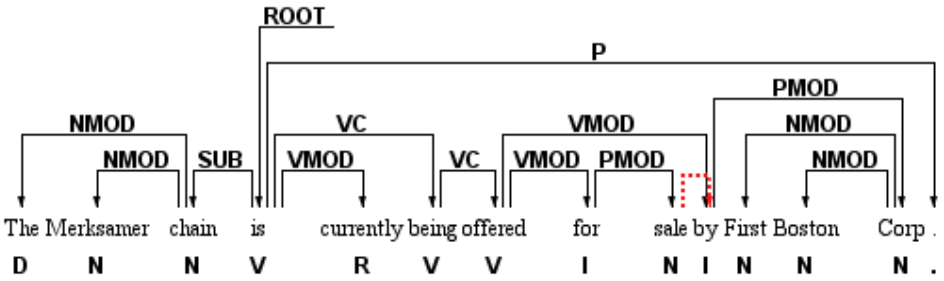

Figure 1. An incorrect dependency tree: the dashed arrow from "sale" to "by" should be replaced with the one from "offered" to "by".

4.2 Revision rules

A *revision rule* is a sequence of atomic steps on the graph which identifies the head of a modifier. As an example, Figure 1 depicts a tree in which the modifier "by" is incorrectly attached to the head "sale" (dashed arrow), rather than to the correct head "offered" (continuous arrow)[1]. There are several possible revision rules for this case: "uu", move up two nodes; -3, three tokens to the left, etc. To bound the complexity of feature extraction the maximum length of a sequence is bound to 4. A revision for a dependency relation is a link re-direction, which moves a single link in a tree to a different head. This is an elementary transformation which preserves the number of nodes in the tree.

A possible problem with these rules is that they are not tree-preserving, i.e. a tree may become a cyclic graph. For instance, rules that create a link to a descendant introduce cycles, unless the application of another rule will link one of the nodes in the path to the descendant to a node outside the cycle. To address these issues we apply the following heuristics in selecting the proper combination: rules that redirect to child nodes are chosen only when no other rule is applicable (upwards rule are safe), and shorter rules are preferred over longer ones. In our experiments we never observed the production of any cycles.

On Wall Street Journal Penn Treebank section 22 we found that the 20 most frequent rules are sufficient to correct 80% of the errors, see Table 3. This confirms that the atomic movements produce simple and effective revision rules.

[1] Arrows go from head to modifier as agreed among the participants to the CoNLL-X shared task.

COUNTS	RULE	TARGET LOCATION
983	uu	Up twice
685	-1	Token to the left
469	+1	Token to the right
265	[	Head of previous constituent
215	uuu	Up 3 times
197	+1u	Right, up
194	r	To root
174	-1u	Left, up
116	>u	Token after constituent, up
103	ud−−	Up down to leftmost child
90	V	To 1st child with POS verb
83	d+1	Down to first right child
82	uuuu	Up 4 times
74	<	Token before constituent
73	ud+1	Up down to 1st right child
71	uV	Up, down to 1st verb
61	ud-1	Up, down to last left child
56	ud+1d+1	Up, down to 1st right child twice
55	d+1d+1	Down to 1st right child twice
48	d−−	Down to leftmost child

Table 3. 20 most frequent revision rules in wsj22.

4.3 Tree revision problem

The tree revision problem can be formalized as follows. Let $G = (s, A)$ be a dependency tree for sentence $s = \langle w_1, w_2, ..., w_n \rangle$. A revision rule is a mapping $r : A \rightarrow A$ which, when applied to an arc $a = (w_i, d, w_j)$, returns an arc $a' = (w_i, d, w_s)$. A revised parse tree is defined as $r(G) = (s, A')$ such that $A' = \{r(a) : a \in A\}$.

This definition corresponds to applying the revisions to the original tree in a batch, as in (Brill, 1993). Alternatively, one could choose to apply the transformations incrementally, applying each one to the tree resulting from previous applications. We chose the first alternative, since the intermediate trees created during the transformation process may not be well-formed dependency graphs, and analyzing them in order to determine features for classifi-

cation might incur problems. For instance, the graph might have abnormal properties that differ from those of any other graph produced by the parser. Moreover, there might not be enough cases of such graphs to form a sufficiently large training set.

5 Learning a revision model

We frame the problem of revising a tree as a supervised classification task. Given a training set $S = (x_i, y_i)_{i=1}^N$, such that $x_i \in \mathbb{R}^d$ and $y_i \in Y$, our goal is to learn a classifier, i.e., a function $F : X \rightarrow Y$. The output space represents the revision rules, in particular we denote with y_1 the identity revision rule. Features represents syntactic and morphological properties of the dependency being examined in its context on the graph.

5.1 Multiclass perceptron

The classifier used in revision is based on the perceptron algorithm (Rosemblatt, 1958), implemented as a multiclass classifier (Crammer & Singer, 2003). One introduces a weight vector $\alpha_i \in \mathbb{R}^d$ for each $y_i \in Y$, in which $\alpha_{i,j}$ represents the weight associated with feature j in class i, and learn α with the perceptron from the training data using a winner-take-all discriminant function:

$$F(x) = \arg\max_{y \in Y} \langle x, \alpha_y \rangle \quad (10)$$

The only adjustable parameter in this model is the number of instances T to use for training. We chose T by means of validation on the development data, typically with a value around 10 times the size of the training data. For regularization purposes we adopt an average perceptron (Collins, 2002) which returns for each y, $\alpha_y = \frac{1}{T} \sum_{t=1}^{T} \alpha_y^t$, the average of all weight vectors α_y^t posited during training. The perceptron was chosen because outperformed other algorithms we experimented with (MaxEnt, MBL and SVM), particularly when including feature pairs, as discussed later.

5.2 Features

We used as features for the revision phase the same type of features used for training the parser (described in Section 3). This does not have to be the case in general. In fact, one might want to introduce features that are specific for this task. For example, global features of the full tree which might be not possible to represent or extract while parsing, as in statistical parse re-ranking (Collins & Koo, 2006).

The features used are lemma, Part-of-Speech, and dependency type of the following items: the current node, its parent, grandparent, great-grandparent, of the children thereof and, in addition, the previous and next tokens of the node. We also add as features all feature pairs that occurred more than 10 times, to reduce the size of the feature space. In alternative one could use a polynomial kernel. We preferred this option because, given the large size of the training data, a dual model is often impractical.

5.3 Revision model

Given a dependency graph $G = (s, A)$, for a sentence $s = \langle w_1, ..., w_n \rangle$, the revised tree is $R(G) = (s, A')$, where each dependency a'_i is equal to $F(a_i)$. In other words, the head in a_i has been changed, or not, according to the rule predicted by the classifier. In particular, we assume that revisions are independent of each other and perform a revision of a tree from left to right. As Table 3 suggests, there are many revision rules with low frequency. Rather than learning a huge classifier, for rules with little training data, we limit the number of classes to a value k. We experimented with values between 30 and 50, accounting for 98-99% of all rules, and eventually used 50, by experimenting with the development portion of the data. All rules that fall outside the threshold are collected in a single class y_0 of "unresolved" cases. If predicted, y_0, similarly to y_1, has no effect on the dependency.

Occasionally, in 59 sentences out of 2416 on section 23 of the Wall Street Journal Penn Treebank (Marcus et al., 1993), the shift-reduce parser fails to attach a node to a head, producing a disconnected graph. The disconnected node will appear as a root, having no head. The problem occurs most often on punctuations (66/84 on WSJ section 23), so it affects only marginally the accuracy scores (UAS, LAS) as computed in the CoNLL-X evaluation (Buchholz & Marsi, 2006). A final step of the revision deals with multiple roots, using a heuristic rule it selects one of the disconnected sub-trees as root, a verb, and attaches all sub-trees to it.

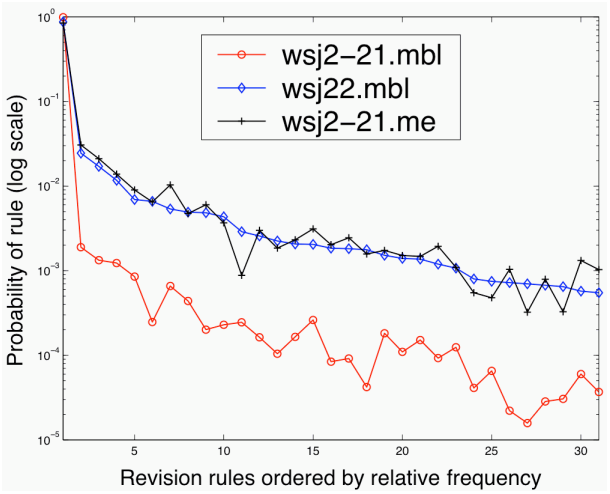

Figure 2. Frequency of the 30 most frequent rules obtained with different parsers on wsj22 and wsj2-21.

5.4 Algorithm complexity

The base dependency parser is deterministic and performs a single scan over the sentence. For each word it performs feature extraction and invokes the classifier to predict the parsing action. If prediction time is bound by a constant, as in linear classifiers, parsing has linear complexity. The revision pass is deterministic and performs similar feature extraction and prediction on each token. Hence, the complexity of the overall parser is $O(n)$. In comparison, the complexity of McDonald's parser (2006) is cubic, while the parser of Yamada and Matsumoto (2003) has a worst case quadratic complexity.

6 Experiments

6.1 Data and setup

We evaluated our method on English using the standard partitions of the Wall Street Journal Penn Treebank: sections 2-21 for training, section 22 for development, and section 23 for evaluation. The constituent trees were transformed into dependency trees by means of a script implementing rules proposed by Collins and Yamada[2]. In a second evaluation we used the Swedish Treebank (Nilsson et al., 2005) from CoNLL-X, approximately 11,000 sentences; for development purposes we performed cross-validation on the training data.

We trained two base parsers on the Penn Treebank: one with our own implementation of Maximum Entropy, one with the TiMBL library for Memory Based Learning (MBL, (Timbl, 2003)). We parsed sections 2 to 21 with each parser and produced two datasets for training the revision model: "wsj2-21.mbl" and "wsj2-21.me". Each dependency is represented as a feature vector (cf. Section 5.2), the prediction is a revision rule (cf. Section 4.2). For the smaller Swedish data we trained one base parser with MaxEnt and one with the SVM implementation in libSVM (Chang & Lin, 2001) using a polynomial kernel with degree 2.

[2]http://w3.msi.vxu.se/%7enivre/research/Penn2Malt.html

Parser	UAS	LAS
DeSR-ME	84.96	83.53
DeSR-MBL	88.41	86.85
Revision-MBL	89.11	86.39
Revision-ME	90.27	86.44
N&S	87.3	-
Y&M	90.3	-
MST-2	**91.5**	-

Table 4. Results on the Wall Street Journal Penn Treebank.

6.2 Results

On the Penn Treebank, the base parser trained with MBL (DeSR-MBL) achieves higher accuracy, 88.41 unlabeled accuracy score (UAS), than the same parser trained with MaxEnt (DeSR-ME), 84.96 UAS. The revision model trained on "wsj2-21.me" (Revision-ME) increases the accuracy of DeSR-ME to 88.01 UAS (+3%). The revision model trained on "wsj2-21.mbl" (DeSR-MBL) improves the accuracy of DeSR-MBL from 88.42 to 89.11 (+0.7%). The difference is mainly due to the fact that DeSR-MBL is quite accurate on the training data, almost 99%, hence "wsj2-21.mbl" contains less errors on which to train the revision parser. This is typical of the memory-based learning algorithm used in DeSR-MBL. Conversely, DeSR-ME achieves a score of of 85% on the training data, which is closer to the actual accuracy of the parser on unseen data. As an illustration, Figure 2 plots the distributions of revision rules in "wsj2-21.mbl" (DeSR-MBL), "wsj2-21.me" (DeSR-ME), and "wsj22.mbl" (DeSR-MBL) which represents the distribution of correct revision rules on the output of DeSR-MBL on the development set. The distributions of "wsj2-

Parser	UAS	LAS
DeSR-SVM	88.41	83.31
Revision-ME	**89.76**	83.13
Corston-Oliver& Aue	89.54	82.33
Nivre	89.50	84.58

Table 5. Results on the Swedish Treebank.

21.me" and "wsj22.mbl" are visibly similar, while "wsj2-21.mbl" is significantly more skewed towards not revising. Hence, the less accurate parser DeSR-ME might be more suitable for producing revision training data. Applying the revision model trained on "wsj2-21.me" (Revision-ME) to the output of DeSR-MBL the result is 90.27% UAS. A relative error reduction of 16.05% from the previous 88.41 UAS of DeSR-MBL. This finding suggests that it may be worth while experimenting with all possible revision-model/base-parser pairs as well as exploring alternative ways for generating data for the revision model; e.g., by cross-validation.

Table 4 summarizes the results on the Penn Treebank. Revision models are evaluated on the output of DeSR-MBL. The table also reports the scores obtained on the same data set by by the shift reduce parsers of Nivre and Scholz's (2004) and Yamada and Matsumoto (2003), and McDonald and Pereira's second-order maximum spanning tree parser (McDonald & Pereira, 2006). However the scores are not directly comparable, since in our experiments we used the settings of the CoNLL-X Shared Task, which provide correct POS tags to the parser.

On the Swedish Treebank collection we trained a revision model (Revision-ME) on the output of the MaxEnt base parser. We parsed the evaluation data with the SVM base parser (DeSR-SVM) which achieves 88.41 UAS. The revision model achieves 89.76 UAS, with a relative error reduction of 11.64%. Here we can compare directly with the best systems for this dataset in CoNLL-X. The best system (Corston-Oliver & Aue, 2006), a variant of the MST algorithm, obtained 89.54 UAS, while the second system (Nivre, 2006) obtained 89.50; cf. Table 5. Parsing the Swedish evaluation set (about 6,000 words) DeSR-SVM processes 1.7 words per second on a Xeon 2.8Ghz machine, DeSR-ME parses more than one thousand w/sec. In the revision step Revision-ME processes 61 w/sec.

7 Related work

Several authors have proposed to improve parsing via re-ranking (Collins, 2000; Charniak & Johnson, 2005; Collins & Koo, 2006). The base parser produces a list of n-best parse trees for a sentence. The re-ranker is trained on the output trees, using additional global features, with a discriminative model. These approaches achieve error reductions up to 13% (Collins & Koo, 2006). In transformation-based learning (Brill, 1993; Brill, 1995; Satta & Brill, 1995) the learning algorithm starts with a baseline assignment, e.g., the most frequent Part-of-Speech for a word, then repeatedly applies rewriting rules. Similarly to re-ranking our method aims at improving the accuracy of the base parser with an additional learner. However, as in transformation-based learning, it avoids generating multiple parses and applies revisions to arcs in the tree which it considers incorrect. This is consistent with the architecture of our base parser, which is deterministic and builds a single tree, rather than evaluating the best outcome of a generator.

With respect to transformation-based methods, our method does not attempt to build a tree but only to revise it. That is, it defines a different output space from the base parser's: the possible revisions on the graph. The revision model of Nakagawa et al. (2002) applies a second classifier for deciding whether the predictions of a base learner are accurate. However, the model only makes a binary decision, which is suitable for the simpler problem of POS tagging. The work of Hall and Novak (Hall & Novak, 2005) is the closest to ours. Hall and Novak develop a corrective model for constituency parsing in order to recover non-projective dependencies, which a standard constituent parser does not handle. The technique is applied to parsing Czech.

8 Conclusion

We presented a novel approach for improving the accuracy of a dependency parser by applying revision transformations to its parse trees. Experimental results prove that the approach is viable and promising. The proposed method achieves good accuracy and excellent performance using a deterministic shift-reduce base parser. As an issue for further investigation, we mention that in this framework, as

in re-ranking, it is possible to exploit global features in the revision phase; e.g., semantic features such as those produced by named-entity detection systems.

Acknowledgments

We would like to thank Jordi Atserias and Brian Roark for useful discussions and comments.

References

G. Attardi. 2006. Experiments with a Multilanguage Non-Projective Dependency Parser. In *Proceedings of CoNNL-X 2006*.

S. Buchholz and E. Marsi. 2006. Introduction to CoNNL-X Shared Task on Multilingual Dependency Parsing. In *Proceedings of CoNNL-X 2006*.

E. Brill. 1993. Automatic Grammar Induction and Parsing free Text: A Transformation-Based Approach. In *Proceedings of ACL 1993.*

E. Brill. 1995. *Transformation-Based Error-Driven Learning and Natural Language Processing.* Computational Linguistics 21(4): pp.543-565.

E. Charniak and M. Johnson. 2005. Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking. In *Proceedings of ACL 2005*.

C. Chang and C. Lin. 2001. LIBSVM: A Library for Support Vector Machines. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

M. Ciaramita, A. Gangemi, E. Ratsch, J. Sarić and I. Rojas. 2005. Unsupervised Learning of Semantic Relations between Concepts of a Molecular Biology Ontology. In *Proceedings of IJCAI 2005*.

M. Collins. 2000. Discriminative Reranking for Natural Language Parsing. In *Proceedings of ICML 2000.*

M. Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of EMNLP 2002.*

M. Collins and T. Koo. 2006. *Discriminative Reranking for Natural Language Parsing.* Computational Linguistics 31(1): pp.25-69.

K. Crammer and Y. Singer. 2003. *Ultraconservative Online Algorithms for Multiclass Problems.* Journal of Machine Learning Research 3: pp.951-991.

S. Corston-Oliver and A. Aue. 2006. Dependency Parsing with Reference to Slovene, Spanish and Swedish. In *Proceedings of CoNLL-X*.

A. Culotta and J. Sorensen. 2004. Dependency Tree Kernels for Relation Extraction. In *Proceedings of ACL 2004*.

W. Daelemans, J. Zavrel, K. van der Sloot, and A. van den Bosch. 2003. *Timbl: Tilburg memory based learner, version 5.0, reference guide.* Technical Report ILK 03-10, Tilburg University, ILK.

Y. Ding and M. Palmer. 2005. Machine Translation using Probabilistic Synchronous Dependency Insertion Grammars. In *Proceedings of ACL 2005*.

K. Hall and V. Novak. 2005. Corrective Modeling for Non-Projective Dependency Parsing. In *Proceedings of the 9th International Workshop on Parsing Technologies*.

M. Marcus, B. Santorini and M. Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2): pp. 313-330.

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

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

T. Nakagawa, T. Kudo and Y. Matsumoto. 2002. Revision Learning and its Applications to Part-of-Speech Tagging. In *Proceedings of ACL 2002*.

J. Nilsson, J. Hall and J. Nivre. 2005. MAMBA Meets TIGER: Reconstructing a Swedish Treebank from Antiquity. In *Proceedings of the NODALIDA*.

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

J. Nivre. 2006. Labeled Pseudo-Projective Dependency Parsing with Support Vector Machines. In *Proceedings of CoNLL-X*.

F. Rosemblatt. 1958. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psych. Rev.*, 68: pp. 386-407.

G. Satta and E. Brill. 1995, Efficient Transformation-Based Parsing. In *Proceedings of ACL 1996*.

R. Snow, D. Jurafsky and Y. Ng 2005. Learning Syntactic Patterns for Automatic Hypernym Discovery. In *Proceedings of NIPS 17*.

H. Yamada and Y. Matsumoto. 2003. Statistical Dependency Analysis with Support Vector Machines. In *Proceedings of the 9th International Workshop on Parsing Technologies.*