Corrective Dependency Parsing

Keith Hall and Václav Novák

Abstract We present a discriminative model for correcting errors in automatically generated dependency trees. We show that by focusing on "structurally local" errors, we can improve the overall quality of the dependency structure. Defining the task by way of a locality constraint allows us to search over a large set of alternate dependency trees simply by making small perturbations to individual dependency edges. This technique requires no additional data for training as it uses the original training data and parser to generate a set of parses from which the training examples are generated. We present experimental results on a Czech corpus using four different parsers, both projective and non-projective, showing the robustness of the technique.

1 Introduction

This article presents a discriminative modeling technique which corrects the errors made by an automatic parser. The model is similar to reranking; however, it does not require the generation of k-best lists as in McDonald, Pereira, Ribarov, and Hajič (2005b); McDonald and Pereira (2006), Charniak and Johnson (2005), and Hall (2007). The *corrective* strategy employed by our technique is to explore a set of candidate parses which are constructed by making structurally–local perturbations to an automatically generated parse tree. We train a model which makes local, corrective decisions in order to optimize for parsing performance. The technique is independent of the parser generating the first set of parses. We show in this article

Keith Hall
Google Research, Zurich, Switzerland, e-mail: kbhall@google.com

Václav Novák
Charles University, Prague, Czech Republic, e-mail: novak@ufal.mff.cuni.cz

that the only requirement for this technique is the ability to define a local neighborhood in which a large number of the errors occur.

The original motivation for the corrective technique was to extend the state-of-the-art dependency parsing technique based on constituency parsing (Collins, Ramshaw, Hajič, and Tillmann, 1999). Statistical parsing models have been shown to be successful in recovering labeled constituencies (Collins, 2003; Charniak and Johnson, 2005; Collins and Roark, 2004) as well as recovering dependency relationships (Collins et al, 1999; Levy and Manning, 2004; Dubey and Keller, 2003; McDonald et al, 2005b). The most effective models are lexicalized and include predictive models based on local context (e.g., the lexicalized probabilistic context-free grammars (PCFGs) used by Collins (2003) and Charniak (2000)). The embedded-bracketing constraint of constituency analysis restricts the types of dependency structures that can be encoded in derived trees.[1] A shortcoming of the constituency-based paradigm for parsing is that it is inherently incapable of representing non-projective dependency trees (we define non-projectivity in the following section). This is particularly problematic when parsing free word-order languages, such as Czech, due to the frequency of sentences with non-projective constructions.

We explore a corrective model which recovers non-projective dependency structures by training a classifier to select correct dependency pairs from a set of candidates based on parses generated by an automatic parser. We chose to use this model due to the observations that the dependency errors made by all of the parsers considered are commonly local errors. For the nodes with incorrect dependency links in the parser output, the correct governor of a node is often found within a local context of the proposed governor. By considering alternative dependencies based on local deviations of the parser output we constrain the set of candidate governors for each node during the corrective procedure. We present our previous results for two state-of-the-art constituency-based parsers (the Collins Czech parser (1999) and a version of the Charniak parser (2001) that was modified to parse Czech). We also present results in this article for experiments using state-of-the art dependency parsers, a projective parser based on the Eisner algorithm (Eisner, 1996) and a maximum spanning tree (MST) based non-projective parser as presented in McDonald, Crammer, and Pereira (2005a) and McDonald et al (2005b). In this work, we only explore techniques where exhaustive parsing[2] is used to generate the base set of parse trees. Additionally, there are greedy shift-reduce-based parsers which perform equally as well as the exhaustive techniques (Attardi, 2006; Nivre, 2006).

The technique proposed in this article is similar to that of recent parser reranking approaches (Collins, 2000; Charniak and Johnson, 2005; Hall, 2007); however, while reranking approaches allow a parser to generate a likely candidate set according to a generative model, we consider a set of candidates based on local perturbations of the single most likely tree generated. The primary reason for such an ap-

[1] In order to correctly capture the dependency structure, co-indexed movement *traces* are used in a form similar to Government and Binding theory, GPSG, etc.

[2] Exhaustive parsing assumes that the optimal parse under the model has been chosen; this is in contrast to greedy techniques, where the parse may not be optimal under the model.

proach is that we allow dependency structures which would never be hypothesized by the parser. Specifically, we allow for non-projective dependencies.

The corrective algorithm proposed in this article shares the motivation of the transformation-based learning work (Brill, 1995). We do consider local transformations of the dependency trees; however, the technique presented here is based on a generative model that maximizes the likelihood of good dependents. We consider a finite set of local perturbations of the tree and use a fixed model to select the best tree by independently choosing optimal dependency links.

In the next section we present an overview of the types of syntactic dependency trees we address in our experiments, specifically the Prague Dependency Treebank (PDT). In Section 3 we review the techniques used to adapt constituency parsers for dependency parsing as well as a brief overview of the other dependency parsing techniques we use. Section 4 describes corrective modeling as used in this work and Section 4.2 describes the model features with which we have experimented. Section 5 presents the results of a set of experiments we performed on data from the PDT with various baseline parsers.

2 Syntactic Dependency Trees

A dependency tree is a set of nodes $\Omega = \{w_0, w_1, \ldots, w_k\}$ where w_0 is the imaginary root node[3] and a set of dependency links $G = \{g_1, \ldots, g_k\}$ where g_i is an index into Ω representing the governor of w_i. In other words, $g_3 = 1$ indicates that the governor of w_3 is w_1. Finally, every node has exactly one governor except for w_0, which has no governor (the tree constraints).[4] The index of the nodes represents the surface order of the nodes in the sequence (i.e., w_i precedes w_j in the sentence if $i < j$).

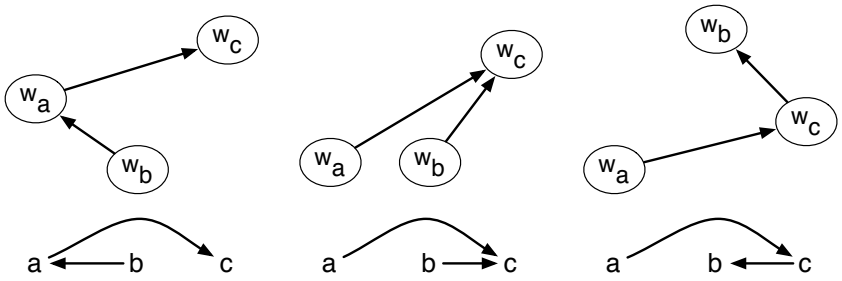

Fig. 1 Examples of projective and non-projective trees, using two different notations. The trees on the left and center are both projective. The tree on the right is non-projective.

[3] The imaginary root node simplifies notation.

[4] The dependency structures here are very similar to those described by Mel'čuk (1988); however the nodes of the dependency trees discussed in this article are limited to the words of the sentence and are always ordered according to the surface word-order.

A tree is *projective* if for every three nodes: w_a, w_b, and w_c where $a < b < c$; if w_a is governed by w_c then w_b is transitively governed by w_c or if w_c is governed by w_a then w_b is transitively governed by w_a.[5] Figure 1 shows examples of projective and non-projective trees. The rightmost tree, which is non-projective, contains a subtree consisting of w_a and w_c but not w_b; however, w_b occurs between w_a and w_c in the linear ordering of the nodes. Projectivity in a dependency tree is akin to the continuity constraint in a constituency tree; that is, the words within a constituent appear contiguously in the surface string. Such a constraint is implicitly imposed by trees generated from context free grammars (CFGs).

Strict word-order languages, such as English, exhibit non-projective dependency structures in a relatively constrained set of syntactic configurations (e.g., right-node raising). Traditionally, these movements are encoded in constituency analyses as *traces*. In languages with free word-order, such as Czech, constituency-based representations are overly constrained (Sgall, Hajičová, and Panevová, 1986); this causes word-order choice to influence the complexity of the syntactic analysis. Alternatively, syntactic dependency trees encode syntactic subordination relationships allowing the structure to be non-specific about the *surface word-order*. The relationship between a node and its subordinates expresses a sense of syntactic (functional) entailment.

In this work we explore the dependency structures encoded in the Prague Dependency Treebank (Hajič, 1998; Böhmová, Hajič, Hajičová, and Hladká, 2002). The PDT analytical layer is a set of Czech syntactic dependency trees; the nodes of which contain the word forms, morphological features, and syntactic annotations. The trees were annotated by hand and are intended as an intermediate stage in the annotation of the Tectogrammatical Representation (TR), a *deep-syntactic* or syntacto-semantic layer in the theory of language (Sgall et al, 1986). All current automatic techniques for generating TR structures are based on syntactic dependency parsing. We report results on two sets of data in order to make comparisons with relevant parsers. The PDT 1.0 data is used to compare the results of corrective models for constituency-based parsers. For corrective modeling results for dependency-based parsers we use the CoNLL 2007 version of the PDT 2.0 (Nivre, Hall, Kübler, McDonald, Nilsson, Riedel, and Yuret, 2007).

When evaluating the accuracy of dependency trees, we present unlabeled dependency scores. The current model specifically targets correcting the dependency structure and not relabeling the dependency edge.

3 Dependency Parsing Techniques

We review two approaches to dependency parsing for which we have examined the corrective modeling approach. The first is based on constituency analysis and the second on a two varieties of dependency parsing. Exploring more than one frame-

[5] Node w_a is said to transitively govern node w_b if w_b is a descendant of w_a in the dependency tree.

work for parsing, allows us to better understand the benefits of the corrective modeling paradigm.

3.1 Constituency Parsing for Dependency Trees

In Hall and Novák (2005) we motivate the use of constituency parses pragmatically; at the time the Collins and Charniak parsers were more accurate on the PDT than available dependency parsers. Note that both Charniak's and Collins' generative probabilistic models contain lexicalized dependency features.[6] From a generative modeling perspective, we use the constraints imposed by constituents (i.e., projectivity) to enable the encapsulation of syntactic substructures. This directly leads to efficient parsing algorithms such as the CKY algorithm and related agenda-based parsing algorithms (Manning and Schütze, 1999). Additionally, this allows for the efficient computation of the scores for the dynamic-programming state variables (i.e., the inside and outside probabilities) that are used in these statistical parsers. The computational complexity advantages of dynamic programming techniques along with efficient search techniques (Caraballo and Charniak, 1998; Klein and Manning, 2003) allow for richer predictive models which include local contextual information.

In an attempt to extend a constituency-based parsing model to train on dependency trees, Collins and colleagues transform the PDT dependency trees into constituency trees (Collins et al, 1999). In order to accomplish this task, they first normalize the trees to remove non-projectivities. Then, they create artificial constituents based on the parts-of-speech of the words associated with each dependency node. The mapping from dependency tree to constituency tree is not one-to-one. They describe a heuristic for choosing trees that works well with this parsing model.

3.1.1 Training a Constituency-based dependency Parser

We consider two approaches to creating projective trees from dependency trees exhibiting non-projectivities. The first is based on word-reordering and is the model that was used with the Collins parser. This algorithm identifies non-projective structures and deterministically reorders the words of the sentence to create projective trees. An alternative method, used by Charniak in the adaptation of his parser for Czech[7] and used by Nivre and Nilsson (2005), alters the dependency links by raising the governor to a higher node in the tree whenever a non-projectivity is observed.

[6] Bilexical dependencies are components of both the Collins and Charniak parsers and model the types of syntactic subordination that we encode in a dependency tree. (Bilexical models were also proposed by Eisner (Eisner, 1996)). In the absence of lexicalization, both parsers have dependency features that are encoded as head-constituent to sibling features.

[7] This information was provided by Eugene Charniak in a personal communication.

The trees are then transformed into Penn-Treebank-style constituencies using the technique described in Collins et al (1999).

Both of these techniques have advantages and disadvantages which we briefly outline here:

Reordering The dependency structure is preserved, but the training procedure will learn statistics for structures over word-strings that may not be part of the language. The parser, however, may be capable of constructing parses for any string of words if a smoothed grammar is being used.

Governor–Raising The dependency structure is corrupted leading the parser to incorporate arbitrary dependency statistics into the model. However, the parser is trained on true sentences, the words of which are in the correct linear order. We expect the parser to predict similar incorrect dependencies when sentences similar to the training data are observed.

Although the results presented in Collins et al (1999) used the reordering technique, we have experimented with his parser using the governor–raising technique and observed an increase in dependency accuracy. For the remainder of the article, we assume the governor–raising technique.

The process of generating dependency trees from parsed constituency trees is relatively straight-forward. Both the Collins and Charniak parsers provide head-word annotation on each constituent. This is precisely the information that we encode in an unlabeled dependency tree, so the dependency structure can simply be extracted from the parsed constituency trees. Furthermore, the constituency labels can be used to identify the dependency labels; however, we do not attempt to identify correct dependency labels in this work.

3.2 Dependency Parsing

Corrective modeling is robust in the sense that it is not constrained by the type of baseline parser being used. In our previous work, we focused on constituency-based dependency parsing and here we extend this to the output of other parsers. Note that our technique can be used for non-statistical (even unweighted) parsing techniques as it need only be given a proposed parse tree and the gold standard for training. Eisner (1996) introduced a model for dependency parsing based on CKY parsing which is a direct dependency parsing technique. The CKY algorithm, and all dynamic-programming algorithms, require a factorization of the search space that allows for the recursive definition of sub-problems. The Eisner algorithm is limited to generating projective dependency structures in the same way as the constituency-based techniques. However, the models used for dependency parsing can differ quite a bit from those used in constituency parsing. McDonald et al (2005a) present accurate dependency parsing models for an implementation of the Eisner algorithm.

Non-projective parsing can be restated as a graph search problem, where the goal of finding a maximum directed spanning tree (MST) is equivalent to finding the

maximum scoring parse. McDonald et al (2005b) introduced this approach along with effective graph-based models for dependency parsing. The short-coming of the MST approach is that the model scores must be *edge-factored*, meaning that the score for one edge cannot be conditioned on the existence of any other edge in the graph. This limits the expressivity of the models and has been shown to be a limiting factor in MST parsing (McDonald, Lerman, and Pereira, 2006; Hall, 2007).

3.3 Dependency Errors

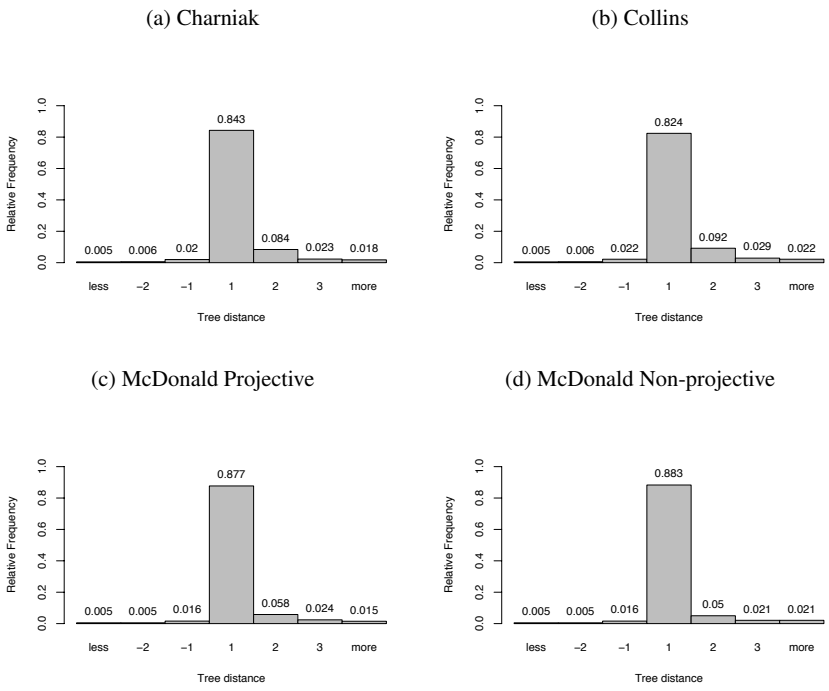

Fig. 2 Statistical distribution of correct governor positions in parsed output of the PDT development for the Charniak parser (a), Collins parser (b), McDonald projective (Eisner) parser (c), and McDonald non-projective (MST) parser (d).

We now discuss a quantitative measure for the types of dependency errors made by dependency parsing techniques. For node w_i and the correct governor $w_{g_i^*}$ the *distance* between the two nodes in the hypothesized dependency tree is:

$$dist(w_i, w_{g_i^*}) = \begin{cases} d(w_i, w_{g_i^*}) & \text{iff } w_{g_i^*} \text{ is ancestor of } w_i \\ d(w_i, w_{g_i^*}) & \text{iff } w_{g_i^*} \text{ is sibling/cousin of } w_i \\ -d(w_i, w_{g_i^*}) & \text{iff } w_{g_i^*} \text{ is descendant of } w_i \end{cases}$$

Ancestor, sibling, and descendant have the standard interpretation in the context of a tree[8]. The dependency distance $d(w_i, w_{g_i^*})$ is the undirected shortest-path from w_i to $w_{g_i^*}$ in the hypothesized dependency tree. The definition of the *dist* function makes a distinction between paths through the parent of w_i (positive values) and paths through children of w_i (negative values). For all the parsers examined, we found that many of the correct governors were actually hypothesized as siblings or grandparents (a *dist* values of 2) – an extremely local error.

Figure 2 shows a histogram of the fraction of nodes whose correct governor was within a particular *dist* in the hypothesized tree. A *dist* of 1 indicates the correct governor was selected by the parser; in these graphs, the density at $dist = 1$ (on the x axis) shows the baseline dependency accuracy of each parser. For the constituency-based parsers (Charniak and Collins), if we repaired only the nodes that are within a *dist* of 2 (grandparents and siblings), we can recover more than 50% of the incorrect dependency links (a raw accuracy improvement of up to 9%). We believe this distribution to be indirectly caused by the governor-raising projectivization routine. In the cases where non-projective structures can be repaired by raising the node's governor to its parent, the correct governor becomes a sibling of the node.

4 Corrective Modeling

Table 1 Corrective Modeling Procedure

```
CORRECT(W)
1  Parse sentence W using the constituency-based parser
2  Generate a dependency structure from the constituency tree
3  for w_i ∈ W
4  do for w_c ∈ N(w_{g_i^h})        // Local neighborhood of proposed governor
5     do l(c) ← P(g_i^* = c | w_i, N(w_{g_i^h}))
6        g_i' ← arg max_c l(c)      // Pick the governor in which we are most confident
```

The error analysis of the previous section suggests that by looking only at a local neighborhood of the proposed governor in the hypothesized trees, we can correct many of the incorrect dependencies. This fact motivates the corrective modeling procedure employed here.

[8] A cousin is a descendant of an ancestor and not an ancestor itself, which subsumes the definition of sibling.

Table 1 presents the pseudo-code for the corrective procedure. The set g^h contains the indices of governors as predicted by the parser. The set of governors predicted by the corrective procedure is denoted as g'. The procedure independently corrects each node of the parsed trees, meaning that there is potential for inconsistent governor relationships to exist in the proposed set; specifically, the resulting dependency graph may have cycles. We employ a greedy search to remove cycles when they are present in the output graph.

The final line of the algorithm picks the governor in which we are most confident. We use the correct-governor classification likelihood: $P(g_i^* = c|w_i, \mathcal{N}(w_{g_i^h}))$, as a measure of the confidence that w_c is the correct governor of w_i where the parser had proposed $w_{g_i^h}$ as the governor. In effect, we create a decision list using the most likely decision when it is valid (i.e., there are no cycles). If the dependency graph resulting from the most likely decisions does not result in a tree, we use the decision lists to greedily select the tree for which the product of the independent decisions is maximal. This is closely related to a greedy version of the Edmonds/Chu-Liu MST algorithm (see Tarjan (1977)).

Training the corrective model requires pairs of dependency trees; each pair contains a manually-annotated tree (i.e., the gold standard tree) and a tree generated by the parser. This data is trivially transformed into per-node samples. For each node w_i in the tree, there are $|\mathcal{N}(w_{g_i^h})|$ samples; one for each governor candidate in the local neighborhood.

One advantage to the type of corrective algorithm presented here is that it is completely disconnected from the parser used to generate the tree hypotheses. This means that the original parser need not be statistical or even constituency based. However, in order for this technique to work, the distribution of dependency errors must be relatively local, as is the case with the errors made by the parsers we explore in the empirical section below. This can be determined via data analysis using the *dist* metric. Determining the size of the local neighborhood is data/parser dependent. If subordinate nodes are considered as candidate governors, then a more robust cycle removal technique is required.

4.1 Maximum Entropy Estimation

We have chosen a Maximum Entropy (MaxEnt) model to estimate the governor distributions: $P(g_i^* = c|w_i, \mathcal{N}(w_{g_i^h}))$. In the next section, we outline the feature set with which we have experimented, noting that the features are selected based on linguistic intuition. An advantage of the MaxEnt framework is that we need not describe the generative process in terms of factoring the interdependencies of the features.

The maximum entropy principle states that we wish to find an estimate of $p(y|x) \in \mathscr{C}$ that maximizes the entropy over a sample set X for some set of observations Y, where $x \in X$ is an observation and $y \in Y$ is a outcome label assigned

to that observation,

$$H(p) \equiv - \sum_{x \in X, y \in Y} \tilde{p}(x) p(y|x) \log p(y|x)$$

The set $\mathscr{C}$ is the candidate set of distributions from which we wish to select $p(y|x)$. We define this set as the $p(y|x)$ that meets a feature-based expectation constraint. Specifically, we want the expected count of a feature, $f(x,y)$, to be equivalent under the distribution $p(y|x)$ and under the observed distribution $\tilde{p}(y|x)$.

$$\sum_{x \in X, y \in Y} \tilde{p}(x) p(y|x) f_i(x,y) = \sum_{x \in X, y \in Y} \tilde{p}(x) \tilde{p}(y|x) f_i(x,y)$$

$f_i(x,y)$ is a feature of our model with which we capture correlations between observations and outcomes. In the following section, we describe a set of features with which we have experimented to determine when a word is likely to be the correct governor of another word.

We incorporate the expected feature-count constraints into the maximum entropy objective using Lagrange multipliers additionally; constraints are added to ensure the distributions $p(y|x)$ are consistent probability distributions:

$$H(p) + \sum_i \alpha_i \sum_{x \in X, y \in Y} \left(\tilde{p}(x) p(y|x) f_i(x,y) - \tilde{p}(x) \tilde{p}(y|x) f_i(x,y) \right) + \gamma \sum_{y \in Y} p(y|x) - 1$$

Holding the α_i's constant, we compute the unconstrained maximum of the above Lagrangian form:

$$p_\alpha(y|x) = \frac{1}{Z_\alpha(x)} \exp(\sum_i \alpha_i f_i(x,y))$$
$$Z_\alpha(x) = \sum_{y \in Y} \exp(\sum_i \alpha_i f_i(x,y))$$

giving us the log-linear form of the distributions $p(y|x)$ in $\mathscr{C}$ (Z is a normalization constant). Finally, we compute the α_i's that maximize the objective function:

$$- \sum_{x \in X} \tilde{p}(x) \log Z_\alpha(x) + \sum_i \alpha_i \tilde{p}(x,y) f_i(x,y)$$

A number of algorithms have been proposed to efficiently compute the optimization described in this derivation. For a more detailed introduction to maximum entropy estimation see Berger, Della Pietra, and Della Pietra (1996).

4.2 Proposed Model

Given the above formulation of the MaxEnt estimation procedure, we define features over pairs of observations and outcomes. In our case, the observations are simply w_i, w_c, and $\mathcal{N}(w_{g_i^h})$ and the outcome is a binary variable indicating whether $c = g_i^*$ (i.e., w_c is the correct governor). In order to limit the dimensionality of the feature space, we consider feature functions over the outcome, the current node w_i, the candidate governor node w_c and the node proposed as the governor by the parser $w_{g_i^h}$.

Table 2 Description of the classes of features used

Feature Type	Id	Description
Form	F	the fully inflected word form as it appears in the data
Lemma	L	the morphologically reduced lemma
MTag	T	a subset of the morphological tag as described in (Collins et al, 1999)
POS	P	major part-of-speech tag (first field of the morphological tag)
ParserGov	G	true if candidate was proposed as governor by parser
ChildCount	C	the number of children
Agreement	$A(x,y)$	check for case/number agreement between word x and y

Table 2 describes the general classes of features used. We write F_i to indicate the form of the current child node, F_c for the form of the candidate, and F_g as the form of the governor proposed by the parser. A combined feature is denoted as L_iT_c and indicates that we observed a particular lemma for the current node with a particular tag of the candidate.

In all models, we include features containing the form, the lemma, the morphological tag, and the ParserGov feature. We have experimented with different sets of feature combinations. Each combination set is intended to capture some intuitive linguistic correlation. For example, the feature component L_iT_c will fire if a child's lemma L_i is observed with a candidate's morphological tag T_c. One intuition behind features of this sort is that it can capture phenomena surrounding particles; for example, in Czech, the governor of the reflexive particle *se* will likely be a verb.

4.3 Related Work

In Hall and Novák (2005), we proposed our corrective technique in order to improve upon the success of the constituency-based approach. Our approach is more general than the recovery of non-projective structure. In McDonald and Pereira (2006), a related technique is used to identify the optimal non-projective modification to a projective parser. In Attardi and Ciaramita (2007), a technique very similar to the one presented here was shown to improve the parsing quality of a greedy shift-reduce-based parser.

Nivre and Nilsson (2005) introduced a technique where the projectivization transformation is encoded in the non-terminals of constituents during parsing. This allows for a deterministic procedure that undoes the projectivization in the generated parse trees, creating non-projective structures. This technique could be incorporated into a statistical parsing framework; however, we believe that the sparsity of such non-projective configurations may be problematic when using smoothed, backed-off grammars. The deterministic procedure employed by Nivre and Nilsson enables their parser to greedily consider non-projective constructions when possible.

We mentioned above that our approach appears to be similar to that of reranking for statistical parsing (Collins, 2000; Charniak and Johnson, 2005; Hall, 2007). While it is true that we are improving upon the output of the automatic parser, we are not considering multiple alternate parses. Instead, we consider a complete set of alternate trees which are minimal perturbations of the best tree generated by the parser. In the context of dependency parsing, we do this in order to generate structures that constituency-based parsers are incapable of generating (i.e., non-projectivities).

Work by Smith and Eisner (2005) on *contrastive estimation* suggests similar techniques to generate local neighborhoods of a parse; however, the purpose in their work is to define an approximation to the partition function for log-linear estimation (e.g., the normalization factor in a MaxEnt model) to be used in unsupervised learning.

5 Empirical Results

In this section we report results from experiments on the PDT Czech dataset. Approximately 1.9% of the words' dependencies are non-projective and these occur in 23.2% of the sentences[9] (Hajičová et al, 2004). We repeat our previous results for the Charniak parser on PDT 1.0 and the Collins parsers on PDT 2.0. We use Ryan McDonald's parser [10] for both the projective (Eisner algorithm) and non-projective (MST algorithm) experiments on the CoNLL 2007 Czech datasets [11] (Nivre et al, 2007).

The Charniak parser was trained on the entire training set of the PDT 1.0 and then used to parse the same data. It is generally a problem to parse the training data, but in this case the Charniak parser performed only slightly better on the training data than on the development data. We train our model on the Collins trees generated via a 20-fold jack-knife training procedure.[12] We use Zhang Lee's implementation

[9] These statistics are for the complete PDT 1.0 dataset.

[10] `http://sourceforge.net/projects/mstparser`

[11] The CoNLL07 shared-task data is a subset of the PDT 2.0 data.

[12] Jack-knife cross-validation is the process of splitting the data into m sets, training on $m-1$ of these, and applying the trained model the remaining set. We do this m times, resulting in predictions for the entire training set while never using a model trained on the data for which we are making predictions.

of the MaxEnt estimator using the L-BFGS optimization algorithms and Gaussian smoothing.[13]

5.1 Constituency-based corrective models

Table 3 Model feature descriptions.

Model	Features	Description
Count	ChildCount	count of children for the three nodes
MTagL	$T_iT_c, L_iL_c, L_iT_c, T_iL_c, T_iP_g$	conjunctions of MTag and Lemmas
MTagF	$T_iT_c, F_iF_c, F_iT_c, T_iF_c, T_iP_g$	conjunctions of MTag and Forms
POSL	$P_i, P_c, P_g, P_iP_cP_g, P_iP_g, P_cL_c$	conjunctions of POS and Lemma
TTT	$T_iT_cT_g$	conjunction of tags for each of the three nodes
Agr	$A(T_i,T_c), A(T_i,T_g)$	binary feature if case/number agree
Trig	$L_iL_gT_c, T_iL_gT_c, L_iL_gL_c$	trigrams of Lemma/Tag

Table 4 Comparative results for different versions of our model on the Charniak and Collins parse trees for the PDT development data.

Model	Charniak Parse Trees		Collins Parse Trees	
	Devel. Accuracy	NonP Accuracy	Devel. Accuracy	NonP Accuracy
Baseline	84.3%	15.9%	82.4%	12.0%
Simple	84.3%	16.0%	82.5%	12.2%
Simple + Count	84.3%	16.7%	82.5%	13.8%
Simple + MtagL	84.8%	43.5%	83.2%	44.1%
Simple + MtagF	84.8%	42.2%	83.2%	43.2%
Simple + POS	84.3%	16.0%	82.4%	12.1%
Simple + TTT	84.3%	16.0%	82.5%	12.2%
Simple + Agr	84.3%	16.2%	82.5%	12.2%
Simple + Trig	84.9%	47.9%	83.1%	47.7%
All Features	**85.0%**	**51.9%**	**83.5%**	**57.5%**

Table 4 presents results on development data for the correction model applied to constituency-based parsers. The features of the **Simple** model are the form (F), lemma (L), and morphological tag (M) for each node, the parser-proposed governor node, and the candidate node; this model also contains the ParserGov feature. We

[13] Using held-out development data, we determined a Gaussian prior parameter setting of 4 worked best. The optimal number of training iterations was chosen on held-out data for each experiment. This was generally in the order of a couple hundred iterations of L-BFGS. The MaxEnt modeling implementation can be found at `http://homepages.inf.ed.ac.uk/s0450736/maxent_toolkit.html`.

show the results for the simple model augmented with feature sets of the categories described in Table 2. Table 3 provides a short description of each of the models. As we believe the **Simple** model provides the minimum information needed to perform this task, we experimented with the effect of each feature class being added to the model. The final row of Table 4 contains results for the model which includes all features from all other models.

Table 5 Alternative non-projectivity scores for different versions of our model on the Charniak and Collins parse trees.

Model	Charniak Parse Trees			Collins Parse Trees		
	Precision	Recall	F-measure	Precision	Recall	F-measure
Baseline	N/A	0.0%	0.0%	N/A	0.0%	0.0%
Simple	22.6%	0.3%	0.6%	5.0%	0.2%	0.4%
Simple + Count	37.3%	1.1%	2.1%	16.8%	2.0%	3.6%
Simple + MtagL	78.0%	29.7%	43.0%	62.4%	35.0%	44.8%
Simple + MtagF	78.7%	28.6%	42.0%	62.0%	34.3%	44.2%
Simple + POS	23.3%	0.3%	0.6%	2.5%	0.1%	0.2%
Simple + TTT	20.7%	0.3%	0.6%	6.1%	0.2%	0.4%
Simple + Agr	40.0%	0.5%	1.0%	5.7%	0.2%	0.4%
Simple + Trig	74.6%	35.0%	47.6%	52.3%	40.2%	45.5%
All Features	75.7%	39.0%	51.5%	48.1%	51.6%	49.8%

We define **NonP Accuracy** as the accuracy for the nodes which were non-projective in the original trees. Although both the Charniak and the Collins parser can never produce non-projective trees, the baseline NonP accuracy is greater than zero; this is due to the parser making mistakes in the tree such that the originally non-projective node's dependency is correct and in a projective configuration.

Alternatively, we report the Non-Projective Precision and Recall for our experiment suite in Table 5. Here the numerator of the precision is the number of nodes that are non-projective in the correct tree and end up in a non-projective configuration; however, this new configuration may be based on incorrect dependencies. Recall is the obvious counterpart to precision. These values correspond to the NonP accuracy results reported in Table 4. From these tables, we see that the most effective features (when used in isolation) are the conjunctive MTag/Lemma, MTag/Form, and Trigram MTag/Lemma features.

Table 6 shows the results of the full model run on the evaluation data for the Collins and Charniak parse trees. It appears that the Charniak parser fares better on the evaluation data than does the Collins parser. However, the corrective model is still successful at recovering non-projective structures. Overall, we see a significant improvement in the dependency accuracy.

We have performed a review of the errors that the corrective process makes and observed that the model does a poor job dealing with punctuation. This is shown in Table 7 along with other types of nodes on which we performed well and poorly, respectively. Collins et al (1999) explicitly added features to their parser to im-

Table 6 Final results on PDT evaluation datasets for Collins' and Charniak's trees with and without the corrective model

Model	Dependency Accuracy	NonP Accuracy
Collins	81.6%	N/A
Collins + Corrective	**82.8%**	**53.1**%
Charniak	84.4%	N/A
Charniak + Corrective	**85.1%**	**53.9**%

Table 7 Categorization of corrections and errors made by our model on trees from the Charniak parser. *root* is the artificial root node of the PDT tree. For each node position (child, proposed parent, and correct parent), the top five words are reported (based on absolute count of occurrences). The particle 'se' occurs frequently explaining why it occurs in the top five good and top five bad repairs.

Top Five Words for Good/Bad Repairs	
Well repaired child	se i si až jen
Well repaired false governor	v však li na o
Well repaired real governor	a je stát ba ,
Poorly repaired child	, se na že -
Poorly repaired false governor	a , však musí li
Poorly repaired real governor	*root* sklo , je -

prove punctuation accuracy. The PARSEVAL evaluation metric for constituency-based parsing explicitly ignores punctuation in determining the correct boundaries of constituents (Harrison, Abney, Fleckenger, Gdaniec, Grishman, Hindle, Ingria, Marcus, Santorini, and Strzalkowski, 1991) and so should the dependency evaluation. However, the reported results include punctuation for comparative purposes.

5.2 Dependency-based parsing

Table 8 Results for the corrective modeling approach on the dependency-based parsers. Scores are the unlabeled accuracy as reported by the MaltEval tools. Scores for the best hyper-parameter settings on the development data are reported in parentheses. Scores for the evaluation data are for the models which performed best on development data.

Model	Baseline (Development) Evaluation	Unlabeled Dependency Accuracy (Development) Evaluation
Projective	(87.71) 82.92	(88.19) 83.13
Non–Projective	(88.28) 83.51	(88.44) 83.36

We explored the efficacy of our corrective modeling technique on the output of parsers that directly model and generate dependency structures. Recall that the maximum improvement achievable by allowing structurally local corrections (sibling and grandparent) is less than in the case of constituency parsers; this is depicted in graphs (c) and (d) of Figure 2. The results of our experiments are presented in Table 8.[14] We explored the same feature-set as with the constituency-based parsers, and found that the simple model features performed best on the development data. The model hyper-parameters were selected to maximize performance on the development dataset.The model hyper-parameters are the maximum number of training iterations and the mixture of the Gaussian regularizer.

Interestingly, we were able to improve performance of the projective dependency-parser (a second-order feature model parser by the Eisner algorithm). The accuracy difference for the non-projective dependency-parser (a first-order model parsed by the MST algorithm) on both development and evaluation data are not significant. The current model is not robust enough to improve on the non-projective parser results.

5.3 Characterization of Corrective Decisions

Table 9 Categorization of corrections made by our model on the evaluation datasets.

	Charniak	Collins	Projective	Non-projective
Correct to incorrect	13.0%	20.0%	28.6%	50.0%
Incorrect to incorrect	21.5%	25.8%	19.0%	17.5%
Incorrect to correct	65.5%	54.2%	52.4%	32.5%

The central goal of our technique is to learn a model which is able to discriminate between good and bad corrections. In Table 9 we present statistics for the repairs made on the evaluation datasets. Our model is relatively weak in the sense that it is usually only making good corrections on slightly more than 50% of the data (with the exception of the non-projective experiment where very few corrections were made). We believe that the simplicity of our model, especially the limited structural locality of our features, is not discriminative enough when selected from a set of candidates, even when that set is relatively small.

[14] The MaltEval (`http://w3.msi.vxu.se/~jni/malteval/`) tool was used for evaluation of the dependency-based parsers.

6 Conclusion

Corrective modeling is an approach to repair the output from a system where more information can be used in the model. In this article, we present a corrective model for dependency parsing using a Maximum Entropy trained discriminative classifier. We show that many of the errors are structurally local for a set of parsers parsing the Czech PDT data. Our algorithm presents a simple framework for modeling a corrective procedure; the model we proposed shows a gain in performance for most of the parsers examined. A key aspect of this modeling framework is the independence between this model and the baseline parser which generates the trees we correct.

Acknowledgements This work was partially supported by U.S. NSF grants IIS–9982329 and OISE–0530118, by the Czech Ministry of Education grant LC536 and Czech Academy of Sciences grant 1ET201120505.

References

Attardi G (2006) Experiments with a multilanguage non-projective dependency parser. In: Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X), Association for Computational Linguistics, New York City, pp 166–170

Attardi G, Ciaramita M (2007) Tree revision learning for dependency parsing. In: Proceedings of Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics, Association for Computational Linguistics, Rochester, New York, pp 388–395

Berger AL, Della Pietra SA, Della Pietra VJ (1996) A maximum entropy approach to natural language processing. Computational Linguistics 22(1):39–71

Böhmová A, Hajič J, Hajičová E, Hladká BV (2002) The Prague Dependency Treebank: Three-level annotation scenario. In: Abeille A (ed) In Treebanks: Building and Using Syntactically Annotated Corpora, Dordrecht, Kluwer Academic Publishers, The Neterlands, pp 103–127

Brill E (1995) Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. Computational Linguistics 21(4):543–565

Caraballo S, Charniak E (1998) New figures of merit for best-first probabilistic chart parsing. Computational Linguistics 24(2):275–298

Charniak E (2000) A maximum-entropy-inspired parser. In: Proceedings of the 2000 Conference of the North American Chapter of the Association for Computational Linguistics., ACL, New Brunswick, NJ, pp 132–139

Charniak E (2001) Immediate-head parsing for language models. In: Proceedings of 39th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Toulouse, France, pp 124–131

Charniak E, Johnson M (2005) Coarse-to-fine *n*-best parsing and MaxEnt discriminative reranking. In: Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05), Association for Computational Linguistics, Ann Arbor, Michigan, pp 173–180

Collins M (2000) Discriminative reranking for natural language parsing. In: Proceedings of the 17th International Conference on Machine Learning 2000, Morgan Kaufmann, San Francisco, CA, pp 175–182

Collins M (2003) Head-driven statistical models for natural language processing. Computational Linguistics 29(4):589–637

Collins M, Roark B (2004) Incremental parsing with the perceptron algorithm. In: Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics, Barcelona, Spain, pp 111–118

Collins M, Ramshaw L, Hajič J, Tillmann C (1999) A statistical parser for Czech. In: Proceedings of the 37th annual meeting of the Association for Computational Linguistics, pp 505–512

Dubey A, Keller F (2003) Probabilistic parsing for German using sister-head dependencies. In: Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, Sapporo, pp 96–103

Eisner J (1996) Three new probabilistic models for dependency parsing: An exploration. In: Proceedings of the 16th International Conference on Computational Linguistics (COLING), pp 340–345

Hajič J (1998) Building a syntactically annotated corpus: The Prague Dependency Treebank. In: Issues of Valency and Meaning, Karolinum, Praha, pp 106–132

Hajičová E, Havelka J, Sgall P, Veselá K, Zeman D (2004) Issues of projectivity in the Prague Dependency Treebank. Prague Bulletin of Mathematical Linguistics 81:5–22

Hall K (2007) *k*-best spanning tree parsing. In: Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Prague, Czech Republic, pp 392–399

Hall K, Novák V (2005) Corrective modeling for non-projective dependency parsing. In: Proceedings of the Ninth International Workshop on Parsing Technology, Association for Computational Linguistics, Vancouver, British Columbia, pp 42–52

Harrison P, Abney S, Fleckenger D, Gdaniec C, Grishman R, Hindle D, Ingria B, Marcus M, Santorini B, Strzalkowski T (1991) Evaluating syntax performance of parser/grammars of english. In: Proceedings of the Workshop on Evaluating Natural Language Processing Systems, ACL

Klein D, Manning CD (2003) Factored A* search for models over sequences and trees. In: Proceedings of IJCAI 2003, pp 1246–1251

Levy R, Manning C (2004) Deep dependencies from context-free statistical parsers: Correcting the surface dependency approximation. In: Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics, Barcelona, Spain, pp 327–334

Manning CD, Schütze H (1999) Foundations of statistical natural language processing. MIT Press

McDonald R, Pereira F (2006) Online learning of approximate dependency parsing algorithms. In: Proceedings of the Annual Meeting of the European Association for Computational Linguistics, pp 81–88

McDonald R, Crammer K, Pereira F (2005a) Online large-margin training of dependency parsers. In: Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Ann Arbor, Michigan, pp 91–98

McDonald R, Pereira F, Ribarov K, Hajič J (2005b) Non-projective dependency parsing using spanning tree algorithms. In: Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, pp 523–530

McDonald R, Lerman K, Pereira F (2006) Multilingual dependency analysis with a two-stage discriminative parser. In: Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X), Association for Computational Linguistics, New York City, pp 216–220

Mel'čuk I (1988) Dependency Syntax: Theory and Practice. SUNY Press, Albany, NY

Nivre J (2006) Inductive Dependency Parsing, Text, Speech and Language Technology, vol 34. Springer

Nivre J, Nilsson J (2005) Pseudo-projective dependency parsing. In: Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics, Ann Arbor, pp 99–106, URL `http://www.aclweb.org/anthology/P/P05/P05-1013`

Nivre J, Hall J, Kübler S, McDonald R, Nilsson J, Riedel S, Yuret D (2007) The CoNLL 2007 shared task on dependency parsing. In: Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007, Association for Computational Linguistics, Prague, Czech Republic, pp 915–932

Sgall P, Hajičová E, Panevová J (1986) The Meaning of the Sentence in Its Semantic and Pragmatic Aspects. Kluwer Academic, Boston

Smith NA, Eisner J (2005) Contrastive estimation: Training log-linear models on unlabeled data. In: Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05), Association for Computational Linguistics, Ann Arbor, Michigan, pp 354–362

Tarjan R (1977) Finding optimal branchings. Networks 7:25–35