Parse correction with specialized models for difficult attachment types

Enrique Henestroza Anguiano, Marie Candito

Parse Correction with Specialized Models for Difficult Attachment Types

Enrique Henestroza Anguiano and Marie Candito
Alpage (Université Paris Diderot / INRIA)
Paris, France
henestro@inria.fr, marie.candito@linguist.jussieu.fr

Abstract

This paper develops a framework for syntactic dependency parse correction. Dependencies in an input parse tree are revised by selecting, for a given dependent, the best governor from within a small set of candidates. We use a discriminative linear ranking model to select the best governor from a group of candidates for a dependent, and our model includes a rich feature set that encodes syntactic structure in the input parse tree. The parse correction framework is parser-agnostic, and can correct attachments using either a generic model or specialized models tailored to difficult attachment types like coordination and pp-attachment. Our experiments show that parse correction, combining a generic model with specialized models for difficult attachment types, can successfully improve the quality of predicted parse trees output by several representative state-of-the-art dependency parsers for French.

1 Introduction

In syntactic dependency parse correction, attachments in an input parse tree are revised by selecting, for a given dependent, the best governor from within a small set of candidates. The motivation behind parse correction is that attachment decisions, especially traditionally difficult ones like pp-attachment and coordination, may require substantial contextual information in order to be made accurately. Because syntactic dependency parsers predict the parse tree for an entire sentence, they may not be able to take into account sufficient context when making attachment decisions, due to computational complexity. Assuming nonetheless that a predicted parse tree is mostly accurate, parse correction can revise difficult attachments by using the predicted tree's syntactic structure to restrict the set of candidate governors and extract a rich set of features to help select among them. Parse correction is also appealing because it is *parser-agnostic*: it can be trained to correct the output of any dependency parser.

In Section 2 we discuss work related to parse correction, pp-attachment and coordination resolution. In Section 3 we discuss dependency structure and various statistical dependency parsing approaches. In Section 4 we introduce the parse correction framework, and Section 5 describes the features and learning model used in our implementation. In Section 6 we present experiments in which parse correction revises the predicted parse trees of four state-of-the-art dependency parsers for French. We provide concluding remarks in Section 7.

2 Related Work

Previous research directly concerning parse correction includes that of Attardi and Ciaramita (2007), working on English and Swedish, who use an approach that considers a fixed set of revision rules: each rule describes movements in the parse tree leading from a dependent's original governor to a new governor, and a classifier is trained to select the correct revision rule for a given dependent. One drawback of this approach is that the classes lack semantic coherence: a sequence of movements does not necessarily have the same meaning across differ-

ent syntactic trees. Hall and Novák (2005), working on Czech, define a neighborhood of candidate governors centered around the original governor of a dependent, and a Maximum Entropy model determines the probability of each candidate-dependent attachment. We follow primarily from their work in our use of neighborhoods to delimit the set of candidate governors. Our main contributions are: specialized corrective models for difficult attachment types (coordination and pp-attachment) in addition to a general corrective model; more sophisticated features, feature combinations, and feature selection; and a ranking model trained directly to select the true governor from among a set of candidates.

There has also been other work on techniques similar to parse correction. Attardi and Dell'Orletta (2009) investigate *reverse revision*: a left-to-right transition-based model is first used to parse a sentence, then a right-to-left transition-based model is run with additional features taken from the left-to-right model's predicted parse. This approach leads to improved parsing results on a number of languages. While their approach is similar to parse correction in that it uses a predicted parse to inform a subsequent processing step, this information is used to improve a second parser rather than a model for correcting errors. McDonald and Pereira (2006) consider a method for recovering non-projective attachments from a graph representation of a sentence, in which an optimal projective parse tree has been identified. The parse tree's edges are allowed to be rearranged in ways that introduce non-projectivity in order to increase its overall score. This rearrangement approach resembles parse correction because it is a second step that can revise attachments made in the first step, but it differs in a number of ways: it is dependent on a graph-based parsing approach, it does not model errors made by the parser, and it can only output non-projective variants of the predicted parse tree.

As a process that revises the output of a syntactic parser, parse reranking is also similar to parse correction. A well-studied subject (e.g. the work of Charniak and Johnson (2005) and of Collins and Koo (2005)), parse reranking is concerned with the reordering of n-best ranked parse trees output by a syntactic parser. Parse correction has a number of advantages compared to reranking: it can be used with parsers that do not output n-best ranked parses, it can be easily restricted to specific attachment types, and its output space of parse trees is not limited to those appearing in an n-best list. However, parse reranking has the advantage of selecting the globally optimal parse for a sentence from an n-best list, while parse correction makes only locally optimal revisions in the predicted parse for a sentence.

2.1 Difficult Attachment Types

Research on pp-attachment traditionally formulates the problem in isolation, as in the work of Pantel and Lin (2000) and of Olteanu and Moldovan (2005). Examples consist of tuples of the form (v, n_1, p, n_2), where either v or n_1 is the true governor of the pp comprising p and n_2, and the task is to choose between v and n_1. Recently, Atterer and Schütze (2007) have criticized this formulation as unrealistic because it uses an oracle to select candidate governors, and they find that successful approaches for the isolated problem perform no better than state-of-the-art parsers on pp-attachment when evaluated on full sentences. With parse correction, candidate governors are identified automatically with no (v, n_1, p, n_2) restriction, and for several representative parsers we find that parse correction improves pp-attachment performance.

Research on coordination resolution has also often formulated the problem in isolation. Resnik (1999) uses semantic similarity to resolve noun-phrase coordination of the form (n_1, cc, n_2, n_3), where the coordinating conjunction cc coordinates either the heads n_1 and n_2 or the heads n_1 and n_3. The same criticism as the one made by Atterer and Schütze (2007) for pp-attachment might be applied to this approach to coordination resolution. In another formulation, the input consists of a raw sentence, and coordination structure is then detected and disambiguated using discriminative learning models (Shimbo and Hara, 2007) or coordination-specific parsers (Hara et al., 2009). Finally, other work has focused on introducing specialized features for coordination into existing syntactic parsing models (Hogan, 2007). Our approach is novel with respect to previous work by directly modeling the correction of coordination errors made by general-purpose dependency parsers.

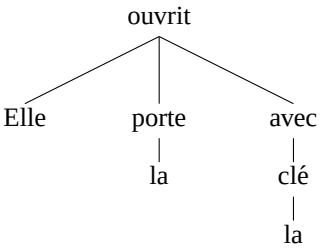

Figure 1: An unlabeled dependency tree for: *Elle ouvrit la porte avec la clé.* (*She opened the door with the key*).

3 Dependency Parsing

Dependency syntax involves the representation of syntactic information for a sentence in the form a directed graph, whose edges encode word-to-word relationships. An edge from a *governor* to a *dependent* indicates, roughly, that the presence of the dependent is syntactically legitimated by the governor. An important property of dependency syntax is that each word, except for the root of the sentence, has exactly one governor; dependency syntax is thus represented by trees. Figure 1 shows an example of an unlabeled dependency tree.[1] For languages like English or French, most sentences can be represented with a *projective* dependency tree: for any edge from word g to word d, g dominates any intervening word between g and d.

Dependency trees are appealing syntactic representations, closer than constituency trees to the semantic representations useful for NLP applications. This is true even with the projectivity requirement, which occasionally creates syntax-semantics mismatches. Dependency trees have recently seen a surge of interest, particularly with the introduction of supervised models for dependency parsing using linear classifiers. Such parsers fall into two main categories: transition-based parsing and graph-based parsing. Additionally, an alternative method for obtaining the dependency parse for a sentence is to parse the sentence with a constituency-based parser and then use an automatic process to convert the output into dependency structure.

[1]Edges are generally labeled with the surface grammatical function that the dependent bears with respect to its governor. In this paper we focus on unlabeled dependency parsing, setting aside labeling as a separate task.

3.1 Transition-Based Parsing

In transition-based dependency parsing, whose seminal works are that of Yamada and Matsumoto (2003) and Nivre (2003), the parsing process applies a sequence of incremental actions, which typically manipulate a buffer position in the sentence and a stack for built sub-structures. Actions are of the type "*read word from buffer*", "*build a dependency from node on top of the stack to node that begins the buffer*", etc. In a greedy version of this process, the action to apply at each step is deterministically chosen to be the best-scoring action according to a classifier, which is trained on a dependency treebank converted into sequences of actions. The strengths of this framework are $O(n)$ time complexity and a lack of restrictions on the locality of features. A major drawback is its greedy behavior: it can potentially make difficult attachment decisions early in the processing of a sentence, without being able to reconsider them when more information becomes available. Beamed versions of the algorithm (Johansson and Nugues, 2006) partially address this problem, but still do not provide a global optimization for selecting the output parse tree.

3.2 Graph-Based Parsing

In graph-based dependency parsing, whose seminal work is that of McDonald et al. (2005), the parsing process selects the globally optimal parse tree from a graph containing attachments (directed edges) between each pair of words (nodes) in a sentence. It finds the k-best scoring parse trees, both during training and at parse time, where the score of a tree is the sum of the scores of its *factors* (consisting of one or more linked edges). While large factors are desirable for capturing sophisticated linguistic constraints, they come at the cost of time complexity: for the projective case, adaptations of Eisner's algorithm (Eisner, 1996) are $O(n^3)$ for 1-edge factors (McDonald et al., 2005) or sibling 2-edge factors (McDonald and Pereira, 2006), and $O(n^4)$ for general 2-edge factors (Carreras, 2007) or 3-edge factors (Koo and Collins, 2010).

3.3 Constituency-Based Parsing

Beyond the two main approaches to dependency parsing, there is also the approach of constituency-

based parsing followed by a conversion step to dependency structure. We use the three-step parsing architecture previously tested for French by Candito et al. (2010a): (i) A constituency parse tree is output by the BerkeleyParser, which has been trained to learn a probabilistic context-free grammar with latent annotations (Petrov et al., 2006) that has parsing time complexity $O(n^3)$ (Matsuzaki et al., 2005); (ii) A functional role labeler using a Maximum Entropy model adds functional annotations to links between a verb and its dependents; (iii) Constituency trees are automatically converted into projective dependency trees, with remaining unlabeled dependencies assigned labels using a rule-based approach.

3.4 Baseline Parsers

In this paper, we use the following baseline parsers: MaltParser (Nivre et al., 2007) for transition-based parsing; MSTParser (McDonald et al., 2005) (with sibling 2-edge factors) and BohnetParser (Bohnet, 2010) (with general 2-edge factors) for graph-based parsing; and BerkeleyParser (Petrov et al., 2006) for constituency-based parsing.

For MaltParser and MSTParser, we use the best settings from a benchmarking of parsers for French (Candito et al., 2010b), except that we remove unsupervised word clusters as features. The parsing models are thus trained using features including predicted part-of-speech tags, lemmas and morphological features. For BohnetParser, we trained a new model using these same predicted features. For BerkelyParser, which was included in the benchmarking experiments, we trained a model using the so-called "desinflection" process that addresses data sparseness due to morphological variation: both at training and parsing time, terminal symbols are word forms in which redundant morphological suffixes are removed, provided the original part-of-speech ambiguities are kept (Candito et al., 2010b).

All models are trained on the French Treebank (FTB) (Abeillé and Barrier, 2004), consisting of 12,351 sentences from the *Le Monde* newspaper, either "desinflected" for the BerkeleyParser, or converted to projective dependency trees (Candito et al., 2010a) for the three dependency-native parsers.[2] For the dependency-native models, features include predicted part-of-speech (POS) tags from the MElt tagger (Denis and Sagot, 2009), as well as predicted lemmas and morphological features from the Le*fff* lexicon (Sagot, 2010). These models constitute the state-of-the-art for French dependency parsing: unlabeled attachment scores (UAS) on the FTB test set are 89.78% for MaltParser, 91.04% for MSTParser, 91.78% for BohnetParser, and 90.73% for BerkeleyParser.

[2]The projectivity constraint is linguistically valid for most French parses: the authors report $< 2\%$ non-projective edges in a hand-corrected subset of the converted FTB.

INPUT: Predicted parse tree T

LOOP: For each chosen dependent $d \in D$

- Identify candidates C_d from T
- Predict $\hat{c} = \underset{c \in C_d}{\operatorname{argmax}}\ S(c, d, T)$
- Update $T\{gov(d) \leftarrow \hat{c}\}$

OUTPUT: Corrected version of parse tree T

Figure 2: The parse correction algorithm.

4 Parse Correction

The parse correction algorithm is a post-processing step to dependency parsing, where attachments from the predicted parse tree of a sentence are corrected by considering alternative candidate governors for each dependent. This process can be useful for attachments made too early in transition-based parsing, or with features that are too local in MST-based parsing.

The input is the predicted parse T of a sentence. From T a set D of dependent nodes are chosen for attachment correction. For each $d \in D$ in left-to-right sentence order, a set C_d of candidate governors from T is identified, and then the highest scoring $c \in C_d$, using a function $S(c, d, T)$, is assigned as the new governor of d in T. Pseudo-code for parse correction is shown in Figure 2.[3]

[3]Contrary to Hall and Novák (2005), our iterative algorithm (along with the fact that C_d never includes nodes that are dominated by d) ensures that corrected structures are trees, so it does not require additional processing to eliminate cycles and preserve connectivity.

4.1 Choosing Dependents

Various criteria may be used to choose the set D of dependents to correct. In the work of Hall and Novák (2005) and of Attardi and Ciaramita (2007), D contains all nodes in the input parse tree. However, one advantage of parse correction is its ability to focus on specific attachment types, so an additional criterion for choosing dependents is to look separately at those dependents that correspond to difficult attachment types.

Analyzing errors made by the dependency parsers introduced in Section 3 on the development set of the FTB, we observe that two major sources of error across different parsers are coordination and pp-attachment. Coordination accounts for around 10% of incorrect attachments and has an error rate ranging from $30 - 40\%$, while pp-attachment accounts for around 30% of incorrect attachments and has an error rate of around 15%.

In this paper, we pay special attention to coordination and pp-attachment. Given the FTB annotation scheme, coordination can be corrected by changing the governor (first conjunct) of the coordinating conjunction that governs the second conjunct, and pp-attachment can be corrected by changing the governor of the preposition that heads the pp.[4] We thus train specialized corrective models for when the dependents are coordinating conjunctions and prepositions, in addition to a generic corrective model that can be applied to any dependent.[5]

4.2 Identifying Candidate Governors

The set of candidate governors C_d for a dependent d can be chosen in different ways. One method is to let every other node in T be a candidate governor for d. However, parser error analysis has shown that errors often occur in local contexts. Hall and Novák (2005) define a neighborhood as a set of nodes $N_m(d)$ around the original predicted governor c_o of d, where $N_m(d)$ includes all nodes in the parse tree T within graph distance m of d that pass through c_o. They find that around $2/3$ of the incorrect attachments in the output of Czech parses can be corrected by selecting the best governor from within $N_3(d)$. Similarly, in oracle experiments reported in section 6, we find that around $1/2$ of coordination and pp-attachments in the output of French parses can be corrected by selecting the best governor from within $N_3(d)$. We thus use neighborhoods to delimit the set of candidate governors.

While one can simply assign $C_d \leftarrow N_m(d)$, we add additional restrictions. First, in order to preserve projectivity within T, we keep in C_d only those c such that the update $T\{gov(d) \leftarrow c\}$ would result in a projective tree.[6] Additionally, we discard candidates with certain POS categories that are very unlikely to be governors: clitics and punctuation are always discarded, while determiners are discarded if the dependent is a preposition.

4.3 Scoring Candidate Governors

A new governor $\hat{c}$ for a dependent d is predicted by selecting the highest scoring candidate $c \in C_d$ according to a function $S(c, d, T)$, which takes into account features over c, d, and the parse tree T. We use a linear model for our scoring function, which allows for relatively fast training and prediction. Our scoring function uses a weight vector $\vec{w} \in \mathbb{F}$, where $\mathbb{F}$ is the feature space for dependents we wish to correct (either generic, or specialized for prepositions or for coordinating conjunction), as well as the mapping $\Phi : \mathbb{C}\times\mathbb{D}\times\mathbb{T} \rightarrow \mathbb{F}$ from combinations of candidate $c \in \mathbb{C}$, dependent $d \in \mathbb{D}$, and parse tree $T \in \mathbb{T}$, to vectors in the feature space $\mathbb{F}$. The scoring function returns the inner product of $\vec{w}$ and $\Phi(c, d, T)$:

$$S(c, d, T) = \vec{w} \cdot \Phi(c, d, T) \quad (1)$$

4.4 Algorithm Complexity

The time complexity of our algorithm is $O(n)$ in the length n of the input sentence, which is consistent with past work on parse correction by Hall and Novák (2005) and by Attardi and Ciaramita (2007).

[4] The FTB handles pp-attachment in a typical fashion, but coordination may be handled differently by other schemes (e.g. the coordinating conjunction governs both conjuncts).

[5] In our experiments, we never revise punctuation and clitic dependents. Since punctuation attachments mostly carry little meaning, they are often annotated inconsistently and ignored in parsing evaluations (including ours). Clitics are not revised because they have a very low attachment error rate (2%).

[6] We also keep candidates that would lead to a non-projective tree, as long as it would be projective if we ignored punctuation. This relaxation of the projectivity constraint leads to better oracle scores while retaining the key linguistic properties of projectivity.

Attachments for up to n dependents in a sentence are deterministically corrected in one pass. For each such dependent d, the algorithm uses a linear model to select a new governor after extracting features for a local set of candidate governors C_d, whose size does not dependent on n in the average case.[7] Locality in candidate governor identification and feature extraction preserves linear time complexity in the overall algorithm.

5 Model Learning

We now discuss our training setup, features, and learning approach for obtaining the weight vector $\vec{w}$.

5.1 Training Setup

The parse correction training set pairs gold parse trees with corresponding predicted parse trees output by a syntactic parser, and it is obtained using a jackknifing procedure to automatically parse the gold-annotated training section of a dependency treebank with a syntactic dependency parser.

We extract separate training sets for each type of dependent we wish to correct (generic, prepositions, coordinating conjunctions). Given p, then for each token d we wish to correct in a sentence in the training section, we note its true governor g_d in the gold parse tree of the sentence, identify a set of candidate governors C_d in the predicted parse T, and get feature vectors $\{\Phi(c, d, T) : c \in C_d\}$.

5.2 Feature Space

In order to learn an effective scoring function, we use a rich feature space $\mathbb{F}$ that encodes syntactic context surrounding a candidate-dependent pair (c, d) within a parse tree T. Our primary features are indicator functions for realizations of linguistic or tree-based feature classes.[8] From these primary features we generate more complex feature combinations of length up to P, which are then added to $\mathbb{F}$. Each combo represents a set of one or more primary features, and is an indicator function that fires if and only if all of its members do.

[7] Degenerate parse trees (e.g. flat trees) could lead to cases where $|C_d|$=n, but for linguistically coherent parse trees $|C_d|$ is rather $O(k^m)$, where k is the average -*arity* of syntactic parse trees and m is the neighborhood distance used.

[8] For instance, there is a binary feature that is 1 if feature class "POS of c" takes on the value "verb", and 0 otherwise.

5.2.1 Primary Feature Classes

The primary feature classes we use are listed below, grouped into categories corresponding to their use in different corrective models (d_{obj} is the object of the dependent, c_{gov} is the governor of the candidate, and c_{d-1} and c_{d+1} are the closest dependents of c linearly to the left and right, respectively, of d).

Generic features (always included)

- POS, lemma, and number of dependents of c
- POS and dependency label of c_{d-1}
- POS and dependency label of c_{d+1}
- POS of c_{gov}
- POS and lemma of d
- POS of d_{obj} and whether d_{obj} has a determiner
- Whether c is the predicted governor of d
- Binned linear distance between c and d
- Linear direction of c with respect to d
- POS sequence for nodes on path from c to d
- Graph distance between c and d
- Whether there is punctuation between c and d

Features exclusive to coordination

Whether d would coordinate two conjuncts that:

- Have the same POS
- Have the same word form
- Have number agreement
- Are both nouns with the same cardinality
- Are both proper nouns or both common nouns
- Are both prepositions with the same word form
- Are both prepositions with object of same POS

Features exclusive to pp-attachment

- Whether d immediately follows a punctuation
- Whether d heads a pp likely to be the agent of a passive verb
- If c is a coordinating conjunction, then whether c would coordinate two prepositions with the same word form, and whether there is at least one open-category word linearly between c and d (in which case c is an unlikely governor)

– If c is linearly after d, then whether there exists a plausible rival candidate to the left of d (implemented as whether there is a noun or adjective linearly before d, without any intervening finite verb)

5.2.2 Feature Selection

Feature combos allow our models to effectively sidestep linearity constraints, at the cost of an exponential increase in the size of the feature space $\mathbb{F}$. In order to accommodate combos, we use feature selection to help reduce the resulting space.

Our first feature selection technique is to apply a frequency threshold: if a feature or a combo appears less than K times among instances in our training set, we remove it from $\mathbb{F}$. In addition to making the feature space more tractable, frequency thresholding makes our scoring function less reliant on rare features and combos.

Following frequency thresholding, we employ an additional technique using conditional entropy (CE) that we term *CE-reduction*. Let Y be a random variable for whether or not an attachment is true, and let A be a random variable for different combos that can appear in an attachment. We calculate the CE of a combo a with respect to Y as follows,

$$H(Y|A{=}a) = -\sum_{y \in Y} p(y|a) \log p(y|a) \quad (2)$$

where the probability $p(y|a)$ is approximated from the training set as $freq(a, y)/freq(a)$, with example balancing used here to account for more false attachments ($Y = 0$) than true ones ($Y = 1$) in our training set. Having calculated the CE of each combo, we remove from $\mathbb{F}$ those combos for which a subset combo (or feature) exists with equal or lesser CE. This eliminates any overly specific combo a when the extra features encoded in a, compared to some subset b, do not help a explain Y any better than b.

5.3 Ranking Model

The ranking setting for learning is used when a model needs to discriminate between mutually exclusive candidates that vary from instance to instance. This is typically used in parse reranking (Charniak and Johnson, 2005), where for each sentence the model must select the correct parse from within an n-best list. Denis and Baldridge (2007) also show that ranking outperforms a binary classification approach to pronoun resolution (using a Maximum Entropy model), where for each pronominal anaphor the model must select the correct antecedent among candidates in a text.[9]

INPUT: Aggressiveness C, rounds R.
INITIALIZE: $\vec{w}_0 \leftarrow (0, ..., 0), \vec{w}_{avg} \leftarrow (0, ..., 0)$
REPEAT: R times
LOOP: For $t = 1, 2, \ldots, |X|$
· Get feature vectors $\{\vec{x}_{t,c} : c \in C_{d_t}\}$
· Get true governor $g_t \in C_{d_t}$
· Let $h_t = \underset{c \in C_{d_t} - \{g_t\}}{\text{argmax}} \ (\vec{w}_{t-1} \cdot \vec{x}_{t,c})$
· Let $m_t = (\vec{w}_{t-1} \cdot \vec{x}_{t,g_t}) - (\vec{w}_{t-1} \cdot \vec{x}_{t,h_t})$
IF: $m_t < 1$
· Let $\tau_t = \min\left\{C\ ,\ \frac{1-m_t}{\|\vec{x}_{t,g_t} - \vec{x}_{t,h_t}\|^2}\right\}$
· Set $\vec{w}_t \leftarrow \vec{w}_{t-1} + \tau_t(\vec{x}_{t,g_t} - \vec{x}_{t,h_t})$
ELSE:
· Set $\vec{w}_t \leftarrow \vec{w}_{t-1}$
· Set $\vec{w}_{avg} \leftarrow \vec{w}_{avg} + \vec{w}_t$
· Set $\vec{w}_0 \leftarrow \vec{w}_{|X|}$
OUTPUT: $\vec{w}_{avg}/(R \cdot |X|)$

Figure 3: Averaged PA-Ranking training algorithm.

In our ranking approach to parse correction (PA-Ranking), the weight vector is trained to select the true governor from a set of candidates C_d for a dependent d. The training set X is defined such that the t^{th} instance is a collection of feature vectors $\{\vec{x}_{t,c} = \Phi(c, d_t, T_t) : c \in C_{d_t}\}$, where C_{d_t} is the candidate set for the dependent d_t within the predicted parse T_t, and the class is the true governor g_t. Instances in which $g_t \notin C_{d_t}$ are discarded.

PA-Ranking training is carried out using a variation of the Passive-Aggressive algorithm (Crammer et al., 2006), which has been adapted to the ranking setting, implemented using the Polka library.[10] For each training iteration t, the margin is defined as

[9] We considered a binary training approach to parse correction in which the model is trained to independently classify candidates as true or false governors, as used by Hall and Novák (2005). However, we found that this approach performs no better (and often worse) than the ranking approach, and is less appropriate from a modeling standpoint.

[10] `http://polka.gforge.inria.fr/`

$m_t = (\vec{w}_{t-1} \cdot \vec{x}_{t,g_t}) - (\vec{w}_{t-1} \cdot \vec{x}_{t,h_t})$, where h_t is the highest scoring incorrect candidate. The algorithm is *passive* because an update to the weight vector is made if and only if $m_t < 1$, either for incorrect predictions ($m_t < 0$) or for correct predictions with insufficient margin ($0 \leq m_t < 1$). The new weight vector $\vec{w}_t$ is as close as possible to $\vec{w}_{t-1}$, subject to the *aggressive* constraint that the new margin be greater than 1. We use weight averaging, so the final output $\vec{w}_{avg}$ is the average over the weight vectors after each training step. Pseudo-code for the training algorithm is shown in Figure 3. The rounds parameter R determines the number of times to run through the training set, and the aggressiveness parameter C sets an upper limit on the update magnitude.

6 Experiments

We present experiments where we applied parse correction to the output of four state-of-the-art dependency parsers for French. We conducted our evaluation on the FTB using the standard training, development (dev), and test splits (containing 9,881, 1,235 and 1,235 sentences, respectively). To train our parse correction models, we generated specialized training sets corresponding to each parser by doing 10-fold jackknifing on the FTB training set (cf. Section 5.1). Each parser was run on the FTB dev and test sets, providing baseline unlabeled attachment score (UAS) results and output parse trees to be corrected.

6.1 Oracles and Neighborhood Size

To determine candidate neighborhood size, we considered an oracle scoring function that always selects the true governor of a dependent if it appears in the set of candidate governors, and otherwise selects the predicted governor. Results for this oracle on the dev set are shown in Table 1. The baseline corresponds to m=1, where the oracle just selects the predicted governor. Incrementing m to 2 and to 3 resulted in substantial gains in oracle UAS, but further incrementing m to 4 resulted in a relatively small additional gain. We found that average candidate set size increases about linearly in m, so we decided to use m=3 in order to have a high UAS upper bound without adding candidates that are very unlikely to be true governors.

		Neighborhood Size (m)			
		Base	2	3	4
Berkeley	Coords	67.2	76.5	82.8	84.8
	Preps	82.9	88.5	92.2	93.2
	Overall	90.1	94.0	96.0	96.5
Bohnet	Coords	70.1	80.6	85.6	87.7
	Preps	85.4	89.4	93.4	94.5
	Overall	91.2	94.4	96.1	96.6
Malt	Coords	60.9	72.2	78.2	80.5
	Preps	82.6	88.1	92.6	93.7
	Overall	89.3	93.2	95.1	95.8
MST	Coords	63.6	73.7	80.7	84.4
	Preps	84.7	89.4	93.4	94.4
	Overall	90.2	93.7	95.6	96.2
MST	Overall	Reranking top-100 parses: 95.4			

Table 1: Parse correction oracle UAS (%) for different neighborhood sizes, by dependent type (coordinating conjunctions, prepositions, or all dependents). Also, a reranking oracle for MSTParser using the top-100 parses.

We also compared the oracle for parse correction with an oracle for parse reranking, in which the parse with the highest UAS for a sentence is selected from the top-100 parses output by MSTParser. We found that for MSTParser, the oracle for parse correction using neighborhood size m=3 (95.6% UAS) is comparable to the oracle for parse reranking using the top-100 parses (95.4% UAS). This is an encouraging result, showing that parse correction is capable of the same improvement as parse reranking without needing to process an n-best list of parses.

6.2 Feature Space Parameters

For the feature space $\mathbb{F}$, we performed a grid search to find good values for the parameters K (frequency threshold), P (combo length), and CE-reduction. We found that P=3 with CE-reduction allowed for the most compactness without sacrificing correction performance, for all of our corrective models. Additionally, K=2 worked well for the coordinating conjunction models, while K=10 worked well for the preposition and generic models. CE-reduction proved useful in greatly reducing the feature space without lowering correction performance: it reduced the size of the coordinating conjunction models from 400k to 65k features each, the preposition models from 400k to 75k features each, and the generic models from 800k to 200k features each.

	Corrective Configuration	UAS (%)		
		Coords	Preps	Overall
Berkeley	Baseline	68.3	83.8	90.73
	Generic	69.4	84.9*	91.13*
	Specialized	71.5*	85.1*	91.23*
Bohnet	Baseline	70.5	86.1	91.78
	Generic	71.2	86.4	91.88
	Specialized	72.7*	86.2	91.88
Malt	Baseline	59.8	83.2	89.78
	Generic	63.2*	84.5*	90.39*
	Specialized	64.0*	85.0*	90.47*
MST	Baseline	60.5	85.9	91.04
	Generic	64.2*	86.2	91.25*
	Specialized	68.0*	86.2	91.36*

Table 2: Coordinating conjunction, preposition, and overall UAS (%) by corrective configuration on the test set. Significant improvements over the baseline starred.

6.3 Corrective Configurations

For our evaluation of parse correction, we compared two different configurations: *generic* (corrects all dependents using the generic model) and *specialized* (corrects coordinating conjunctions and prepositions using their respective specialized models, and corrects other dependents using the generic model). The PA-Ranking aggressiveness parameter C was set to 1 for our experiments, while the rounds parameter R was tuned separately for each corrective model using the dev set. For our final tests, we applied each combination of parser + corrective configuration by sequentially revising all dependents in the output parse that had a relevant POS tag given the corrective configuration. In the FTB test set, this amounted to an evaluation over 5,706 preposition tokens, 801 coordinating conjunction tokens, and 31,404 overall (non-punctuation) tokens.[11]

6.4 Results

Final results for the test set are shown in Table 2. The overall UAS of each parser (except BohnetParser) was significantly improved under both corrective configurations.[12] The *specialized* configuration performed as well as, and in most cases better than, the *generic* configuration, indicating the usefulness of specialized models and features for difficult attachment types. Interestingly, the lower the baseline parser's UAS, the larger the overall improvement from parse correction under the *specialized* configuration: MaltParser had the lowest baseline and the highest error reduction (6.8%), BerkeleyParser had the second-lowest baseline and the second-highest error reduction (5.4%), MSTParser had the third-lowest baseline and the third-highest error reduction (3.6%), and BohnetParser had the highest baseline and the lowest error reduction (1.2%). It may be that the additional errors made by a low-baseline parser, compared to a high-baseline parser, involve relatively simpler attachments that parse correction can better model.

Parse correction achieved significant improvements for coordination resolution under the *specialized* configuration for each parser. MaltParser and MSTParser had very low baseline coordinating conjunction UAS (around 60%), while BerkeleyParser and BohnetParser had higher baselines (around 70%). The highest error reduction was achieved by MSTParser (19.0%), followed by MaltParser (10.4%), BerkeleyParser (10.1%), and finally BohnetParser (7.5%). The result for MSTParser was surprising: although it had the second-highest baseline overall UAS, it shared the lowest baseline coordinating conjunction UAS and had the highest error reduction with parse correction. An explanation for this result is that the annotation scheme for coordination structure in the dependency FTB has the first conjunct governing the coordinating conjunction, which governs the second conjunct. Since MSTParser is limited to sibling 2-edge factors (cf. section 3), it is unable to jointly consider a full coordination structure. BohnetParser, which uses general 2-edge factors, can consider full coordination structures and consequently has a much higher baseline coordinating conjunction UAS than MSTParser.

Parse correction achieved significant but modest improvements in pp-attachment performance under the *specialized* configuration for MaltParser and BerkeleyParser. However, parse correction did not significantly improve pp-attachment performance for MSTParser or BohnetParser, the two parsers that had the highest baseline preposition UAS (around

[11] Since the MElt tagger and BerkeleyParser POS tagging accuracies were around 97%, the sets of tokens considered for revision differed slightly from the sets of tokens (with gold POS tags) used to calculate UAS scores.

[12] We used McNemar's Chi-squared test with $p = 0.05$ for all significance tests.

		Modification Type			
		$w \rightarrow c$	$c \rightarrow w$	$w \rightarrow w$	Mods
Berkeley	Coords	40	14	33	10.9 %
	Preps	118	39	41	3.5 %
	Overall	228	67	104	1.3 %
Bohnet	Coords	32	15	33	10.0 %
	Preps	52	46	32	2.3 %
	Overall	150	121	130	1.1 %
Malt	Coords	55	21	56	16.5 %
	Preps	149	50	76	4.8 %
	Overall	390	172	293	2.4 %
MST	Coords	80	20	51	18.9 %
	Preps	64	45	26	2.4 %
	Overall	183	88	117	1.1 %

Table 3: Breakdown of modifications made under the *specialized* configuration for each parser, by dependent type. $w \rightarrow c$ is wrong-to-correct, $c \rightarrow w$ is correct-to-wrong, $w \rightarrow w$ is wrong-to-wrong, and Mods is the percentage of tokens modified.

86%). These results are a bit disappointing, but they suggest that there may be a performance ceiling for pp-attachment beyond which rich lexical information (syntactic and semantic) or full sentence contexts are needed. For English, the average human performance on pp-attachment for the (v, n_1, p, n_2) problem formulation is just 88.2% when given only the four head-words, but increases to 93.2% when given the full sentence (Ratnaparkhi et al., 1994). If similar levels of human performance exist for French, additional sources of information may be needed to improve pp-attachment performance.

In addition to evaluating UAS improvements for parse correction, we took a closer look at the best corrective configuration (*specialized*) and analyzed the types of attachment modifications made (Table 3). In most cases there were around $2-3$ times as many error-correcting modifications ($w \rightarrow c$) as error-creating modifications ($c \rightarrow w$), and the overall % of tokens modified was very low overall (around 1-2%). Parse correction is thus conservative in the number of modifications made, and rather accurate when it does decide to modify an attachment.

Finally, we compared the running times of the four parsers, as well as that of parse correction, on the test set using a 2.66 GHz Intel Core 2 Duo machine. BerkeleyParser took 600s, BohnetParser took 450s using both cores (800s using a single core), MaltParser took 45s, and MSTParser took 1000s. A rough version of parse correction in the *specialized* configuration took around 200s (for each parser). An interesting result is that parse correction improves MaltParser the most while retaining an overall time complexity of $O(n)$, compared to $O(n^3)$ or higher for the other parsers. This suggests that linear-time transition-based parsing and parse correction could combine to form an attractive system that improves parsing performance while retaining high speed.

7 Conclusion

We have developed a parse correction framework for syntactic dependency parsing that uses specialized models for difficult attachment types. Candidate governors for a given dependent are identified in a neighborhood around the predicted governor, and a scoring function selects the best governor. We used discriminative linear ranking models with features encoding syntactic context, and we tested parse correction on coordination, pp-attachment, and generic dependencies in the outputs of four representative statistical dependency parsers for French. Parse correction achieved improvements in unlabeled attachment score for three out of the four parsers, with MaltParser seeing the greatest improvement. Since both MaltParser and parse correction run in $O(n)$ time, a combined system could prove useful in situations where high parsing speed is required.

Future work on parse correction might focus on developing specialized models for other difficult attachment types, such as verb-phrase attachment (verb dependents account for around 15% of incorrect attachments across all four parsers). Also, selectional preferences and subcategorization frames (from hand-built resources or extracted using distributional methods) could make for useful features in the pp-attachment corrective model; we suspect that richer lexical information is needed in order to increase the currently modest improvements achieved by parse correction on pp-attachment.

Acknowledgments

We would like to thank Pascal Denis for his help using the Polka library, and Alexis Nasr for his advice and comments. This work was partially funded by the ANR project Sequoia ANR-08-EMER-013.

References

A. Abeillé and N. Barrier. 2004. Enriching a French treebank. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation*, Lisbon, Portugal, May.

G. Attardi and M. Ciaramita. 2007. Tree revision learning for dependency parsing. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, pages 388–395, Rochester, New York, April.

G. Attardi and F. Dell'Orletta. 2009. Reverse revision and linear tree combination for dependency parsing. In *Proceedings of the 2009 Conference of the North American Chapter of the Association for Computational Linguistics*, pages 261–264, Boulder, Colorado, June.

M. Atterer and H. Schütze. 2007. Prepositional phrase attachment without oracles. *Computational Linguistics*, 33(4):469–476.

B. Bohnet. 2010. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 89–97, Beijing, China, August.

M. Candito, B. Crabbé, and P. Denis. 2010a. Statistical French dependency parsing: Treebank conversion and first results. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation*, Valetta, Malta, May.

M. Candito, J. Nivre, P. Denis, and E. Henestroza Anguiano. 2010b. Benchmarking of statistical dependency parsers for French. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 108–116, Beijing, China, August.

X. Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL*, pages 957–961, Prague, Czech Republic, June.

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

M. Collins and T. Koo. 2005. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–70.

K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. 2006. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585.

P. Denis and J. Baldridge. 2007. A ranking approach to pronoun resolution. In *Proceedings of the 20th International Joint Conference on Artifical intelligence*, pages 1588–1593, Hyderabad, India, January.

P. Denis and B. Sagot. 2009. Coupling an annotated corpus and a morphosyntactic lexicon for state-of-the-art POS tagging with less human effort. In *Proceedings of the 23rd Pacific Asia Conference on Language, Information and Computation*, Hong Kong, China, December.

J.M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 340–345, Santa Cruz, California, August.

K. Hall and V. Novák. 2005. Corrective modeling for non-projective dependency parsing. In *Proceedings of the Ninth International Workshop on Parsing Technologies*, pages 42–52, Vancouver, British Columbia, October.

K. Hara, M. Shimbo, H. Okuma, and Y. Matsumoto. 2009. Coordinate structure analysis with global structural constraints and alignment-based local features. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 967–975, Suntec, Singapore, August.

D. Hogan. 2007. Coordinate noun phrase disambiguation in a generative parsing model. In *In Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, number 1, page 680, Prague, Czech Republic, June.

R. Johansson and P. Nugues. 2006. Investigating multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 206–210, New York City, New York, June.

T. Koo and M. Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11, Uppsala, Sweden, July.

T. Matsuzaki, Y. Miyao, and J. Tsujii. 2005. Probabilistic CFG with latent annotations. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 75–82, Ann Arbor, Michigan, June.

R. McDonald and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 81–88, Trento, Italy, April.

R. McDonald, K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 91–98, Ann Arbor, Michigan, June.

J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kübler, S. Marinov, and E. Marsi. 2007. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(02):95–135.

J. Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies*, pages 149–160, Nancy, France, April.

M. Olteanu and D. Moldovan. 2005. PP-attachment disambiguation using large context. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 273–280, Vancouver, British Columbia, October.

P. Pantel and D. Lin. 2000. An unsupervised approach to prepositional phrase attachment using contextually similar words. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, volume 38, pages 101–108, Hong Kong, October.

S. Petrov, L. Barrett, R. Thibaux, and D. Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia, July.

A. Ratnaparkhi, J. Reynar, and S. Roukos. 1994. A maximum entropy model for prepositional phrase attachment. In *Proceedings of the Workshop on Human Language Technology*, pages 250–255, Plainsboro, New Jersey, March.

P. Resnik. 1999. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *Journal of Artificial Intelligence Research*, 11(95):130.

B. Sagot. 2010. The Lefff, a freely available, accurate and large-coverage lexicon for French. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation*, Valetta, Malta, May.

M. Shimbo and K. Hara. 2007. A discriminative learning model for coordinate conjunctions. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 610–619, Prague, Czech Republic, June.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies*, pages 195–206, Nancy, France, April.