Online Learning of Approximate Dependency Parsing Algorithms

Ryan McDonald **Fernando Pereira**
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104
{ryantm,pereira}@cis.upenn.edu

Abstract

In this paper we extend the maximum spanning tree (MST) dependency parsing framework of McDonald et al. (2005c) to incorporate higher-order feature representations and allow dependency structures with multiple parents per word. We show that those extensions can make the MST framework computationally intractable, but that the intractability can be circumvented with new approximate parsing algorithms. We conclude with experiments showing that discriminative online learning using those approximate algorithms achieves the best reported parsing accuracy for Czech and Danish.

1 Introduction

Dependency representations of sentences (Hudson, 1984; Meĺčuk, 1988) model head-dependent syntactic relations as edges in a directed graph. Figure 1 displays a dependency representation for the sentence *John hit the ball with the bat*. This sentence is an example of a projective (or nested) tree representation, in which all edges can be drawn in the plane with none crossing. Sometimes a non-projective representations are preferred, as in the sentence in Figure 2.[1] In particular, for freer-word order languages, non-projectivity is a common phenomenon since the relative positional constraints on dependents is much less rigid. The dependency structures in Figures 1 and 2 satisfy the *tree constraint*: they are weakly connected graphs with a unique root node, and each non-root node has a exactly one parent. Though trees are more common, some formalisms allow for words to modify multiple parents (Hudson, 1984).

Recently, McDonald et al. (2005c) have shown that treating dependency parsing as the search for the highest scoring maximum spanning tree (MST) in a graph yields efficient algorithms for both projective and non-projective trees. When combined with a discriminative online learning algorithm and a rich feature set, these models provide state-of-the-art performance across multiple languages. However, the parsing algorithms require that the score of a dependency tree factors as a sum of the scores of its edges. This *first-order factorization* is very restrictive since it only allows for features to be defined over single attachment decisions. Previous work has shown that conditioning on neighboring decisions can lead to significant improvements in accuracy (Yamada and Matsumoto, 2003; Charniak, 2000).

In this paper we extend the MST parsing framework to incorporate higher-order feature representations of bounded-size connected subgraphs. We also present an algorithm for acyclic dependency graphs, that is, dependency graphs in which a word may depend on multiple heads. In both cases parsing is in general intractable and we provide novel approximate algorithms to make these cases tractable. We evaluate these algorithms within an online learning framework, which has been shown to be robust with respect approximate inference, and describe experiments displaying that these new models lead to state-of-the-art accuracy for English and the best accuracy we know of for Czech and Danish.

2 Maximum Spanning Tree Parsing

Dependency-tree parsing as the search for the maximum spanning tree (MST) in a graph was

[1] Examples are drawn from McDonald et al. (2005c).

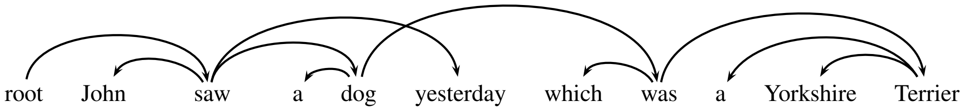


Figure 2: An example non-projective dependency structure.

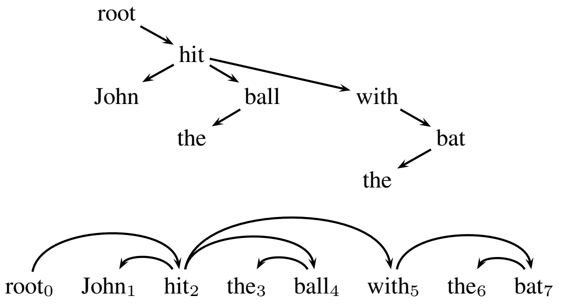


Figure 1: An example dependency structure.

proposed by McDonald et al. (2005c). This formulation leads to efficient parsing algorithms for both projective and non-projective dependency trees with the Eisner algorithm (Eisner, 1996) and the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967) respectively. The formulation works by defining the score of a dependency tree to be the sum of edge scores,

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{(i,j)\in \boldsymbol{y}} s(i,j)$$

where $\boldsymbol{x} = x_1 \cdots x_n$ is an input sentence and $\boldsymbol{y}$ a dependency tree for $\boldsymbol{x}$. We can view $\boldsymbol{y}$ as a set of tree edges and write $(i,j) \in \boldsymbol{y}$ to indicate an edge in $\boldsymbol{y}$ from word x_i to word x_j. Consider the example from Figure 1, where the subscripts index the nodes of the tree. The score of this tree would then be,

$$s(0,2) + s(2,1) + s(2,4) + s(2,5)$$
$$+ s(4,3) + s(5,7) + s(7,6)$$

We call this *first-order* dependency parsing since scores are restricted to a single edge in the dependency tree. The score of an edge is in turn computed as the inner product of a high-dimensional feature representation of the edge with a corresponding weight vector,

$$s(i,j) = \mathbf{w} \cdot \mathbf{f}(i,j)$$

This is a standard linear classifier in which the weight vector **w** are the parameters to be learned during training. We should note that $\mathbf{f}(i,j)$ can be based on arbitrary features of the edge and the input sequence $\boldsymbol{x}$.

Given a directed graph $G = (V, E)$, the maximum spanning tree (MST) problem is to find the highest scoring subgraph of G that satisfies the tree constraint over the vertices V. By defining a graph in which the words in a sentence are the vertices and there is a directed edge between all words with a score as calculated above, McDonald et al. (2005c) showed that dependency parsing is equivalent to finding the MST in this graph. Furthermore, it was shown that this formulation can lead to state-of-the-art results when combined with discriminative learning algorithms.

Although the MST formulation applies to any directed graph, our feature representations and one of the parsing algorithms (Eisner's) rely on a linear ordering of the vertices, namely the order of the words in the sentence.

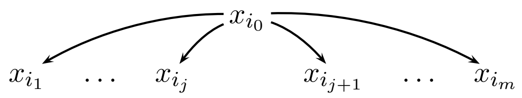
2.1 Second-Order MST Parsing

Restricting scores to a single edge in a dependency tree gives a very impoverished view of dependency parsing. Yamada and Matsumoto (2003) showed that keeping a small amount of parsing history was crucial to improving parsing performance for their locally-trained shift-reduce SVM parser. It is reasonable to assume that other parsing models might benefit from features over previous decisions.

Here we will focus on methods for parsing *second-order* spanning trees. These models factor the score of the tree into the sum of adjacent edge pair scores. To quantify this, consider again the example from Figure 1. In the second-order spanning tree model, the score would be,

$$s(0,-,2) + s(2,-,1) + s(2,-,4) + s(2,4,5)$$
$$+ s(4,-,3) + s(5,-,7) + s(7,-,6)$$

Here we use the second-order score function $s(i,k,j)$, which is the score of creating a pair of adjacent edges, from word x_i to words x_k and x_j. For instance, $s(2,4,5)$ is the score of creating the edges from *hit* to *with* and from *hit* to *ball*. The score functions are relative to the left or right of the parent and we never score adjacent edges that are on different sides of the parent (for instance, there is no $s(2,1,4)$ for the adjacent edges from *hit* to *John* and *ball*). This independence between left and right descendants allow us to use a $O(n^3)$ second-order projective parsing algorithm, as we will see later. We write $s(x_i, -, x_j)$ when x_j is the first left or first right dependent of word x_i. For example, $s(2, -, 4)$ is the score of creating a dependency from *hit* to *ball*, since *ball* is the first child to the right of *hit*. More formally, if the word x_{i_0} has the children shown in this picture,



the score factors as follows:

$$\begin{gathered}\textstyle\sum_{k=1}^{j-1} s(i_0, i_{k+1}, i_k) + s(i_0, -, i_j) \\ + s(i_0, -, i_{j+1}) + \textstyle\sum_{k=j+1}^{m-1} s(i_0, i_k, i_{k+1})\end{gathered}$$

This second-order factorization subsumes the first-order factorization, since the score function could just ignore the middle argument to simulate first-order scoring. The score of a tree for second-order parsing is now

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{(i,k,j) \in \boldsymbol{y}} s(i, k, j)$$

where k and j are adjacent, same-side children of i in the tree $\boldsymbol{y}$.

The second-order model allows us to condition on the most recent parsing decision, that is, the last dependent picked up by a particular word, which is analogous to the the Markov conditioning of in the Charniak parser (Charniak, 2000).

2.2 Exact Projective Parsing

For projective MST parsing, the first-order algorithm can be extended to the second-order case, as was noted by Eisner (1996). The intuition behind the algorithm is shown graphically in Figure 3, which displays both the first-order and second-order algorithms. In the first-order algorithm, a word will gather its left and right dependents independently by gathering each half of the subtree rooted by its dependent in separate stages. By splitting up chart items into left and right components, the Eisner algorithm only requires 3 indices to be maintained at each step, as discussed in detail elsewhere (Eisner, 1996; McDonald et al., 2005b). For the second-order algorithm, the key insight is to delay the scoring of edges until pairs of dependents have been gathered. This allows for the collection of pairs of adjacent dependents in a single stage, which allows for the incorporation of second-order scores, while maintaining cubic-time parsing.

```
**2-order-non-proj-approx**$(\boldsymbol{x}, s)$
    Sentence $\boldsymbol{x} = x_0 \ldots x_n$, $x_0 = root$
    Weight function $s : (i,k,j) \to \mathbb{R}$
    Let $\boldsymbol{y} =$ **2-order-proj**$(\boldsymbol{x}, s)$
    while true
      $m = -\infty, c = -1, p = -1$
      for $j : 1 \cdots n$
        for $i : 0 \cdots n$
          $\boldsymbol{y}' = \boldsymbol{y}[i \to j]$
          if $\neg\text{tree}(\boldsymbol{y}')$ or $\exists k : (i,k,j) \in \boldsymbol{y}$ continue
          $\delta = s(\boldsymbol{x}, \boldsymbol{y}') - s(\boldsymbol{x}, \boldsymbol{y})$
          if $\delta > m$
            $m = \delta, c = j, p = i$
        end for
      end for
      if $m > 0$
        $\boldsymbol{y} = \boldsymbol{y}[p \to c]$
      else $return\ \boldsymbol{y}$
    end while
```

Figure 4: Approximate second-order non-projective parsing algorithm.

The Eisner algorithm can be extended to an arbitrary m^{th}-order model with a complexity of $O(n^{m+1})$, for $m > 1$. An m^{th}-order parsing algorithm will work similarly to the second-order algorithm, except that we collect m pairs of adjacent dependents in succession before attaching them to their parent.

2.3 Approximate Non-projective Parsing

Unfortunately, second-order non-projective MST parsing is NP-hard, as shown in appendix A. To circumvent this, we designed an approximate algorithm based on the exact $O(n^3)$ second-order projective Eisner algorithm. The approximation works by first finding the highest scoring projective parse. It then rearranges edges in the tree, one at a time, as long as such rearrangements increase the overall score and do not violate the tree constraint. We can easily motivate this approximation by observing that even in non-projective languages like Czech and Danish, most trees are primarily projective with just a few non-projective edges (Nivre and Nilsson, 2005). Thus, by starting with the highest scoring projective tree, we are typically only a small number of transformations away from the highest scoring non-projective tree.

The algorithm is shown in Figure 4. The expression $\boldsymbol{y}[i \to j]$ denotes the dependency graph identical to $\boldsymbol{y}$ except that x_i's parent is x_i instead

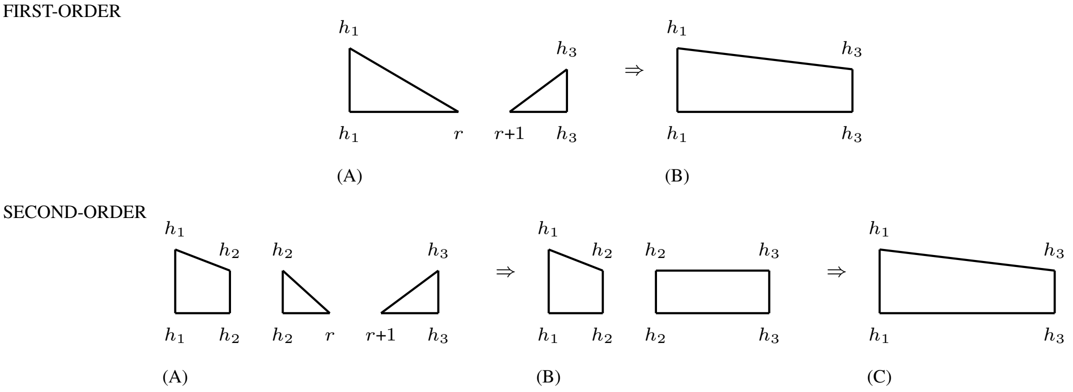


Figure 3: A $O(n^3)$ extension of the Eisner algorithm to second-order dependency parsing. This figure shows how h_1 creates a dependency to h_3 with the second-order knowledge that the last dependent of h_1 was h_2. This is done through the creation of a *sibling* item in part (B). In the first-order model, the dependency to h_3 is created after the algorithm has forgotten that h_2 was the last dependent.

of what it was in $\boldsymbol{y}$. The test $\text{tree}(\boldsymbol{y})$ is true iff the dependency graph $\boldsymbol{y}$ satisfies the tree constraint.

In more detail, line 1 of the algorithm sets $\boldsymbol{y}$ to the highest scoring second-order projective tree. The loop of lines 2–16 exits only when no further score improvement is possible. Each iteration seeks the single highest-scoring parent change to $\boldsymbol{y}$ that does not break the tree constraint. To that effect, the nested loops starting in lines 4 and 5 enumerate all (i, j) pairs. Line 6 sets $\boldsymbol{y}'$ to the dependency graph obtained from $\boldsymbol{y}$ by changing x_j's parent to x_i. Line 7 checks that the move from $\boldsymbol{y}$ to $\boldsymbol{y}'$ is valid by testing that x_j's parent was not already x_i and that $\boldsymbol{y}'$ is a tree. Line 8 computes the score change from $\boldsymbol{y}$ to $\boldsymbol{y}'$. If this change is larger than the previous best change, we record how this new tree was created (lines 9-10). After considering all possible valid edge changes to the tree, the algorithm checks to see that the best new tree does have a higher score. If that is the case, we change the tree permanently and re-enter the loop. Otherwise we exit since there are no single edge switches that can improve the score.

This algorithm allows for the introduction of non-projective edges because we do not restrict any of the edge changes except to maintain the tree property. In fact, if any edge change is ever made, the resulting tree is guaranteed to be non-projective, otherwise there would have been a higher scoring projective tree that would have already been found by the exact projective parsing algorithm. It is not difficult to find examples for which this approximation will terminate without returning the highest-scoring non-projective parse.

It is clear that this approximation will always terminate — there are only a finite number of dependency trees for any given sentence and each iteration of the loop requires an increase in score to continue. However, the loop could potentially take exponential time, so we will bound the number of edge transformations to a fixed value M. It is easy to argue that this will not hurt performance. Even in freer-word order languages such as Czech, almost all non-projective dependency trees are primarily projective, modulo a few non-projective edges. Thus, if our inference algorithm starts with the highest scoring projective parse, the best non-projective parse only differs by a small number of edge transformations. Furthermore, it is easy to show that each iteration of the loop takes $O(n^2)$ time, resulting in a $O(n^3 + Mn^2)$ runtime algorithm. In practice, the approximation terminates after a small number of transformations and we do not need to bound the number of iterations in our experiments.

We should note that this is one of many possible approximations we could have made. Another reasonable approach would be to first find the highest scoring *first-order* non-projective parse, and then re-arrange edges based on second order scores in a similar manner to the algorithm we described. We implemented this method and found that the results were slightly worse.

3 Danish: Parsing Secondary Parents

Kromann (2001) argued for a dependency formalism called *Discontinuous Grammar* and annotated a large set of Danish sentences using this formalism to create the Danish Dependency Treebank (Kromann, 2003). The formalism allows for a

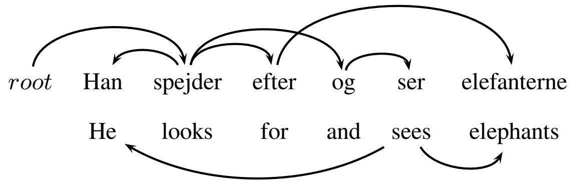


Figure 5: An example dependency tree from the Danish Dependency Treebank (from Kromann (2003)).

word to have multiple parents. Examples include verb coordination in which the subject or object is an argument of several verbs, and relative clauses in which words must satisfy dependencies both inside and outside the clause. An example is shown in Figure 5 for the sentence *He looks for and sees elephants*. Here, the pronoun *He* is the subject for both verbs in the sentence, and the noun *elephants* the corresponding object. In the Danish Dependency Treebank, roughly 5% of words have more than one parent, which breaks the single parent (or tree) constraint we have previously required on dependency structures. Kromann also allows for cyclic dependencies, though we deal only with acyclic dependency graphs here. Though less common than trees, dependency graphs involving multiple parents are well established in the literature (Hudson, 1984). Unfortunately, the problem of finding the dependency structure with highest score in this setting is intractable (Chickering et al., 1994).

To create an approximate parsing algorithm for dependency structures with multiple parents, we start with our approximate second-order non-projective algorithm outlined in Figure 4. We use the non-projective algorithm since the Danish Dependency Treebank contains a small number of non-projective arcs. We then modify lines 7-10 of this algorithm so that it looks for the change in parent *or* the addition of a new parent that causes the highest change in overall score and does not create a cycle[2]. Like before, we make one change per iteration and that change will depend on the resulting score of the new tree. Using this simple new approximate parsing algorithm, we train a new parser that can produce multiple parents.

4 Online Learning and Approximate Inference

In this section, we review the work of McDonald et al. (2005b) for online large-margin dependency parsing. As usual for supervised learning, we assume a training set $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^T$, consisting of pairs of a sentence $\boldsymbol{x}_t$ and its correct dependency representation $\boldsymbol{y}_t$.

The algorithm is an extension of the Margin Infused Relaxed Algorithm (MIRA) (Crammer and Singer, 2003) to learning with structured outputs, in the present case dependency structures. Figure 6 gives pseudo-code for the algorithm. An online learning algorithm considers a single training instance for each update to the weight vector $\mathbf{w}$. We use the common method of setting the final weight vector as the average of the weight vectors after each iteration (Collins, 2002), which has been shown to alleviate overfitting.

On each iteration, the algorithm considers a single training instance. We parse this instance to obtain a predicted dependency graph, and find the smallest-norm update to the weight vector $\mathbf{w}$ that ensures that the training graph outscores the predicted graph by a margin proportional to the loss of the predicted graph relative to the training graph, which is the number of words with incorrect parents in the predicted tree (McDonald et al., 2005b). Note that we only impose margin constraints between the single highest-scoring graph and the correct graph relative to the current weight setting. Past work on tree-structured outputs has used constraints for the k-best scoring tree (McDonald et al., 2005b) or even all possible trees by using factored representations (Taskar et al., 2004; McDonald et al., 2005c). However, we have found that a single margin constraint per example leads to much faster training with a negligible degradation in performance. Furthermore, this formulation relates learning directly to inference, which is important, since we want the model to set weights relative to the errors made by an approximate inference algorithm. This algorithm can thus be viewed as a large-margin version of the perceptron algorithm for structured outputs Collins (2002).

Online learning algorithms have been shown to be robust even with approximate rather than exact inference in problems such as word alignment (Moore, 2005), sequence analysis (Daumé and Marcu, 2005; McDonald et al., 2005a) and phrase-structure parsing (Collins and Roark, 2004). This robustness to approximations comes from the fact that the online framework sets weights *with respect to inference*. In other words, the learning method sees common errors due to

[2] We are not concerned with violating the tree constraint.

```
Training data: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{T}$
$\mathbf{w}^{(0)} = 0;\ \mathbf{v} = 0;\ i = 0$
for $n : 1..N$
  for $t : 1..T$
    $\min \left\| \mathbf{w}^{(i+1)} - \mathbf{w}^{(i)} \right\|$
    s.t. $s(\boldsymbol{x}_t, \boldsymbol{y}_t; \mathbf{w}^{(i+1)})$
              $-s(\boldsymbol{x}_t, \boldsymbol{y}'; \mathbf{w}^{(i+1)}) \geq L(\boldsymbol{y}_t, \boldsymbol{y}')$
    where $\boldsymbol{y}' = \arg\max_{y'} s(\boldsymbol{x}_t, \boldsymbol{y}'; \mathbf{w}^{(i)})$
    $\mathbf{v} = \mathbf{v} + \mathbf{w}^{(i+1)}$
    $i = i + 1$
$\mathbf{w} = \mathbf{v}/(N * T)$
```

Figure 6: MIRA learning algorithm. We write $s(\boldsymbol{x}, \boldsymbol{y}; \mathbf{w}^{(i)})$ to mean the score of tree $\boldsymbol{y}$ using weight vector $\mathbf{w}^{(i)}$.

approximate inference and adjusts weights to correct for them. The work of Daumé and Marcu (2005) formalizes this intuition by presenting an online learning framework in which parameter updates are made directly with respect to errors in the inference algorithm. We show in the next section that this robustness extends to approximate dependency parsing.

5 Experiments

The score of adjacent edges relies on the definition of a feature representation $\mathbf{f}(i, k, j)$. As noted earlier, this representation subsumes the first-order representation of McDonald et al. (2005b), so we can incorporate all of their features as well as the new second-order features we now describe. The old first-order features are built from the parent and child words, their POS tags, and the POS tags of surrounding words and those of words between the child and the parent, as well as the direction and distance from the parent to the child. The second-order features are built from the following conjunctions of word and POS identity predicates

x_i-pos, x_k-pos, x_j-pos
x_k-pos, x_j-pos
x_k-word, x_j-word
x_k-word, x_j-pos
x_k-pos, x_j-word

where x_i-pos is the part-of-speech of the i^{th} word in the sentence. We also include conjunctions between these features and the direction and distance from sibling j to sibling k. We determined the usefulness of these features on the development set, which also helped us find out that features such as the POS tags of words between the two siblings would not improve accuracy. We also ignored features over triples of words since this would explode the size of the feature space.

	English	
	Accuracy	Complete
1st-order-projective	90.7	36.7
2nd-order-projective	91.5	42.1

Table 1: Dependency parsing results for English.

	Czech	
	Accuracy	Complete
1st-order-projective	83.0	30.6
2nd-order-projective	84.2	33.1
1st-order-non-projective	84.1	32.2
2nd-order-non-projective	85.2	35.9

Table 2: Dependency parsing results for Czech.

We evaluate dependencies on per word accuracy, which is the percentage of words in the sentence with the correct parent in the tree, and on complete dependency analysis. In our evaluation we exclude punctuation for English and include it for Czech and Danish, which is the standard.

5.1 English Results

To create data sets for English, we used the Yamada and Matsumoto (2003) head rules to extract dependency trees from the WSJ, setting sections 2-21 as training, section 22 for development and section 23 for evaluation. The models rely on part-of-speech tags as input and we used the Ratnaparkhi (1996) tagger to provide these for the development and evaluation set. These data sets are exclusively projective so we only compare the projective parsers using the exact projective parsing algorithms. The purpose of these experiments is to gauge the overall benefit from including second-order features with exact parsing algorithms, which can be attained in the projective setting. Results are shown in Table 1. We can see that there is clearly an advantage in introducing second-order features. In particular, the complete tree metric is improved considerably.

5.2 Czech Results

For the Czech data, we used the predefined training, development and testing split of the Prague Dependency Treebank (Hajič et al., 2001), and the automatically generated POS tags supplied with the data, which we reduce to the POS tag set from Collins et al. (1999). On average, 23% of the sentences in the training, development and test sets have at least one non-projective dependency, though, less than 2% of total edges are ac-

	Danish		
	Precision	Recall	F-measure
2nd-order-projective	86.4	81.7	83.9
2nd-order-non-projective	86.9	82.2	84.4
2nd-order-non-projective w/ multiple parents	86.2	84.9	85.6

Table 3: Dependency parsing results for Danish.

tually non-projective. Results are shown in Table 2. McDonald et al. (2005c) showed a substantial improvement in accuracy by modeling non-projective edges in Czech, shown by the difference between two first-order models. Table 2 shows that a second-order model provides a comparable accuracy boost, even using an approximate non-projective algorithm. The second-order non-projective model accuracy of 85.2% is the highest reported accuracy for a single parser for these data. Similar results were obtained by Hall and Nóvák (2005) (85.1% accuracy) who take the best output of the Charniak parser extended to Czech and rerank slight variations on this output that introduce non-projective edges. However, this system relies on a much slower phrase-structure parser as its base model as well as an auxiliary reranking module. Indeed, our second-order projective parser analyzes the test set in 16m32s, and the non-projective approximate parser needs 17m03s to parse the entire evaluation set, showing that runtime for the approximation is completely dominated by the initial call to the second-order projective algorithm and that the post-process edge transformation loop typically only iterates a few times per sentence.

5.3 Danish Results

For our experiments we used the Danish Dependency Treebank v1.0. The treebank contains a small number of inter-sentence and cyclic dependencies and we removed all sentences that contained such structures. The resulting data set contained 5384 sentences. We partitioned the data into contiguous 80/20 training/testing splits. We held out a subset of the training data for development purposes.

We compared three systems, the standard second-order projective and non-projective parsing models, as well as our modified second-order non-projective model that allows for the introduction of multiple parents (Section 3). All systems use gold-standard part-of-speech since no trained tagger is readily available for Danish. Results are shown in Figure 3. As might be expected, the non-projective parser does slightly better than the projective parser because around 1% of the edges are non-projective. Since each word may have an arbitrary number of parents, we must use precision and recall rather than accuracy to measure performance. This also means that the correct training loss is no longer the Hamming loss. Instead, we use false positives plus false negatives over edge decisions, which balances precision and recall as our ultimate performance metric.

As expected, for the basic projective and non-projective parsers, recall is roughly 5% lower than precision since these models can only pick up at most one parent per word. For the parser that can introduce multiple parents, we see an increase in recall of nearly 3% absolute with a slight drop in precision. These results are very promising and further show the robustness of discriminative online learning with approximate parsing algorithms.

6 Discussion

We described approximate dependency parsing algorithms that support higher-order features and multiple parents. We showed that these approximations can be combined with online learning to achieve fast parsing with competitive parsing accuracy. These results show that the gain from allowing richer representations outweighs the loss from approximate parsing and further shows the robustness of online learning algorithms with approximate inference.

The approximations we have presented are very simple. They start with a reasonably good baseline and make small transformations until the score of the structure converges. These approximations work because freer-word order languages we studied are still primarily projective, making the approximate starting point close to the goal parse. However, we would like to investigate the benefits for parsing of more principled approaches to approximate learning and inference techniques such as the learning as search optimization framework of (Daumé and Marcu, 2005). This framework will possibly allow us to include effectively more global features over the dependency structure than those in our current second-order model.

Acknowledgments

This work was supported by NSF ITR grants 0205448.

References

E. Charniak. 2000. A maximum-entropy-inspired parser. In *Proc. NAACL*.

D.M. Chickering, D. Geiger, and D. Heckerman. 1994. Learning bayesian networks: The combination of knowledge and statistical data. Technical Report MSR-TR-94-09, Microsoft Research.

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

M. Collins and B. Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proc. ACL*.

M. Collins, J. Hajič, L. Ramshaw, and C. Tillmann. 1999. A statistical parser for Czech. In *Proc. ACL*.

M. Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proc. EMNLP*.

K. Crammer and Y. Singer. 2003. Ultraconservative online algorithms for multiclass problems. *JMLR*.

H. Daum´e and D. Marcu. 2005. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proc. ICML*.

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

J. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proc. COLING*.

J. Hajič, E. Hajicova, P. Pajas, J. Panevova, P. Sgall, and B. Vidova Hladka. 2001. The Prague Dependency Treebank 1.0 CDROM. Linguistics Data Consortium Cat. No. LDC2001T10.

K. Hall and V. N´ov´ak. 2005. Corrective modeling for non-projective dependency parsing. In *Proc. IWPT*.

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

M. T. Kromann. 2001. Optimaility parsing and local cost functions in discontinuous grammars. In *Proc. FG-MOL*.

M. T. Kromann. 2003. The danish dependency treebank and the dtag treebank tool. In *Proc. TLT*.

R. McDonald, K. Crammer, and F. Pereira. 2005a. Flexible text segmentation with structured multilabel classification. In *Proc. HLT-EMNLP*.

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

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

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

R. Moore. 2005. A discriminative framework for bilingual word alignment. In *Proc. HLT-EMNLP*.

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

A. Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Proc. EMNLP*.

B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. 2004. Max-margin parsing. In *Proc. EMNLP*.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proc. IWPT*.

A 2nd-Order Non-projective MST Parsing is NP-hard

Proof by a reduction from 3-D matching (3DM).

3DM: Disjoint sets X, Y, Z each with m distinct elements and a set $T \subseteq X \times Y \times Z$. Question: is there a subset $S \subseteq T$ such that $|S| = m$ and each $v \in X \cup Y \cup Z$ occurs in exactly one element of S.

Reduction: Given an instance of 3DM we define a graph in which the vertices are the elements from $X \cup Y \cup Z$ as well as an artificial *root* node. We insert edges from *root* to all $x_i \in X$ as well as edges from all $x_i \in X$ to all $y_i \in Y$ and $z_i \in Z$. We order the words s.t. the root is on the left followed by all elements of X, then Y, and finally Z. We then define the second-order score function as follows,

$s(root, x_i, x_j) = 0, \ \forall x_i, x_j \in X$
$s(x_i, -, y_j) = 0, \ \forall x_i \in X, y_j \in Y$
$s(x_i, y_j, z_k) = 1, \ \forall (x_i, y_j, z_k) \in T$

All other scores are defined to be $-\infty$, including for edges pairs that were not defined in the original graph.

Theorem: *There is a 3D matching iff the second-order MST has a score of* m. **Proof**: First we observe that no tree can have a score greater than m since that would require more than m pairs of edges of the form (x_i, y_j, z_k). This can only happen when some x_i has multiple $y_j \in Y$ children or multiple $z_k \in Z$ children. But if this were true then we would introduce a $-\infty$ scored edge pair (e.g. $s(x_i, y_j, y'_j)$). Now, if the highest scoring second-order MST has a score of m, that means that every x_i must have found a unique pair of children y_j and z_k which represents the 3D matching, since there would be m such triples. Furthermore, y_j and z_k could not match with any other x'_i since they can only have one incoming edge in the tree. On the other hand, if there is a 3DM, then there must be a tree of weight m consisting of second-order edges (x_i, y_j, z_k) for each element of the matching S. Since no tree can have a weight greater than m, this must be the highest scoring second-order MST. Thus if we can find the highest scoring second-order MST in polynomial time, then 3DM would also be solvable. ■