Maximum Spanning Tree Algorithm for Non-projective Labeled Dependency Parsing

Nobuyuki Shimizu
Dept. of Computer Science
State University of New York at Albany
Albany, NY, 12222, USA
shimizu@cs.albany.edu

Abstract

Following (McDonald et al., 2005), we present an application of a maximum spanning tree algorithm for a directed graph to non-projective labeled dependency parsing. Using a variant of the voted perceptron (Collins, 2002; Collins and Roark, 2004; Crammer and Singer, 2003), we discriminatively trained our parser in an on-line fashion. After just one epoch of training, we were generally able to attain average results in the CoNLL 2006 Shared Task.

1 Introduction

Recently, we have seen dependency parsing grow more popular. It is not rare to see dependency relations used as features, in tasks such as relation extraction (Bunescu and Mooney, 2005) and machine translation (Ding and Palmer, 2005). Although English dependency relations are mostly projective, in other languages with more flexible word order, such as Czech, non-projective dependencies are more frequent. There are generally two methods for learning non-projective dependencies. You could map a non-projective dependency tree to a projective one, learn and predict the tree, then bring it back to the non-projective dependency tree (Nivre and Nilsson, 2005). Non-projective dependency parsing can also be represented as search for a maximum spanning tree in a directed graph, and this technique has been shown to perform well in Czech (McDonald et al., 2005). In this paper, we investigate the effectiveness of (McDonald et al., 2005) in the various languages given by the CoNLL 2006 shared task for non-projective labeled dependency parsing.

The paper is structured as follows: in section 2 and 3, we review the decoding and learning aspects of (McDonald et al., 2005), and in section 4, we describe the extension of the algorithm and the features needed for the CoNLL 2006 shared task.

2 Non-Projective Dependency Parsing

2.1 Dependency Structure

Let us define x to be a generic sequence of input tokens together with their POS tags and other morphological features, and y to be a generic dependency structure, that is, a set of edges for x. We use the terminology in (Taskar et al., 2004) for a generic structured output prediction, and define a *part*.

A *part* represents an edge together with its label. A part is a tuple $\langle DEPREL, i, j \rangle$ where i is the start point of the edge, j is the end point, and *DEPREL* is the label of the edge. The token at i is the head of the token at j.

Table 1 shows our formulation of building a non-projective dependency tree as a prediction problem. The task is to predict y, the set of parts (column 3, Table 1), given x, the input tokens and their features (column 1 and 2, Table 1).

In this paper we use the common method of factoring the score of the dependency structure as the sum of the scores of all the parts.

A dependency structure is characterized by its features, and for each feature, we have a correspond-

Token	POS	Edge Part
John	NN	$\langle$*SUBJ*$, 2, 1\rangle$
saw	VBD	$\langle$*PRED*$, 0, 2\rangle$
a	DT	$\langle$*DET*$, 4, 3\rangle$
dog	NN	$\langle$*OBJ*$, 2, 4\rangle$
yesterday	RB	$\langle$*ADJU*$, 2, 5\rangle$
which	WDT	$\langle$*MODWH*$, 7, 6\rangle$
was	VBD	$\langle$*MODPRED*$, 4, 7\rangle$
a	DT	$\langle$*DET*$, 10, 8\rangle$
Yorkshire	NN	$\langle$*MODN*$, 10, 9\rangle$
Terrier	NN	$\langle$*OBJ*$, 7, 10\rangle$
.	.	$\langle ., 10, 11\rangle$

Table 1: Example Parts

ing weight. The score of a dependency structure is the sum of these weights. Now, the dependency structures are factored by the parts, so that each feature is some type of a specialization of a part. Each part in a dependency structure maps to several features. If we sum up the weights for these features, we have the score for the part, and if we sum up the scores of the parts, we have the score for the dependency structure.

For example, let us say we would like to find the score of the part $\langle$*OBJ*$, 2, 4\rangle$. This is the edge going to the 4th token ”dog” in Table 1. Suppose there are two features for this part.

- There is an edge labeled with ”OBJ” that points to the right. (= *DEPREL*, $dir(i, j)$)
- There is an edge labeled with ”OBJ” starting at the token ”saw” which points to the right. (= *DEPREL*, $dir(i, j)$, $word_i$)

If a statement is never true during the training, the weight for it will be 0. Otherwise there will be a positive weight value. The score will be the sum of all the weights of the features given by the part.

In the upcoming section, we explain a decoding algorithm for the dependency structures, and later we give a method for learning the weight vector used in the decoding.

2.2 Maximum Spanning Tree Algorithm

As in (McDonald et al., 2005), the decoding algorithm we used is the Chu-Liu-Edmonds (CLE) algorithm (Chu and Liu, 1965; Edmonds, 1967) for finding the Maximum Spanning Tree in a directed graph. The following is a nice summary by (McDonald et al., 2005).

> Informally, the algorithm has each vertex in the graph greedily select the incoming edge with highest weight.

Note that the edge is coming from the parent to the child. This means that given a child node $word_j$, we are finding the parent, or the head $word_i$ such that the edge (i, j) has the highest weight among all i, $i \neq j$.

> If a tree results, then this must be the maximum spanning tree. If not, there must be a cycle. The procedure identifies a cycle and contracts it into a single vertex and recalculates edge weights going into and out of the cycle. It can be shown that a maximum spanning tree on the contracted graph is equivalent to a maximum spanning tree in the original graph (Leonidas, 2003). Hence the algorithm can recursively call itself on the new graph.

3 Online Learning

Again following (McDonald et al., 2005), we have used the single best MIRA (Crammer and Singer, 2003), which is a variant of the voted perceptron (Collins, 2002; Collins and Roark, 2004) for structured prediction. In short, the update is executed when the decoder fails to predict the correct parse, and we compare the correct parse y^t and the incorrect parse y' suggested by the decoding algorithm. The weights of the features in y' will be lowered, and the weights of the features in y^t will be increased accordingly.

4 Experiments

Our experiments were conducted on CoNLL-X shared task, with various datasets (Hajič et al., 2004; Simov et al., 2005; Simov and Osenova, 2003; Chen et al., 2003; Böhmová et al., 2003; Kromann, 2003; van der Beek et al., 2002; Brants et al., 2002; Kawata and Bartels, 2000; Afonso et al., 2002; Džeroski et al., 2006; Civit Torruella and Martí Antonín, 2002; Nilsson et al., 2005; Oflazer et al., 2003; Atalay et al., 2003) .

4.1 Dependency Relation

The CLE algorithm works on a directed graph with unlabeled edges. Since the CoNLL-X shared task

Given a part $\langle \textit{DEPREL}, i, j\rangle$
$\textit{DEPREL}, dir(i,j)$
$\textit{DEPREL}, dir(i,j), word_i$
$\textit{DEPREL}, dir(i,j), pos_i$
$\textit{DEPREL}, dir(i,j), word_j$
$\textit{DEPREL}, dir(i,j), pos_j$
$\textit{DEPREL}, dir(i,j), word_i, pos_i$
$\textit{DEPREL}, dir(i,j), word_j, pos_j$
$\textit{DEPREL}, dir(i,j), word_{i-1}$
$\textit{DEPREL}, dir(i,j), pos_{i-1}$
$\textit{DEPREL}, dir(i,j), word_{i-1}, pos_{i-1}$
$\textit{DEPREL}, dir(i,j), word_{j-1}$
$\textit{DEPREL}, dir(i,j), pos_{j-1}$
$\textit{DEPREL}, dir(i,j), word_{j-1}, pos_{j-1}$
$\textit{DEPREL}, dir(i,j), word_{i+1}$
$\textit{DEPREL}, dir(i,j), pos_{i+1}$
$\textit{DEPREL}, dir(i,j), word_{i+1}, pos_{i+1}$
$\textit{DEPREL}, dir(i,j), word_{j+1}$
$\textit{DEPREL}, dir(i,j), pos_{j+1}$
$\textit{DEPREL}, dir(i,j), word_{j+1}, pos_{j+1}$
$\textit{DEPREL}, dir(i,j), pos_{i-2}$
$\textit{DEPREL}, dir(i,j), pos_{i+2}$
$\textit{DEPREL}, dir(i,j)$, distance = $\lvert j-i \rvert$
additional features
$\textit{DEPREL}, dir(i,j), word_i, word_j$
$\textit{DEPREL}, dir(i,j), pos_{i+1}, pos_i, pos_{i+1}$
$\textit{DEPREL}, dir(i,j), pos_{i+1}, word_i, pos_{i+1}$
$\textit{DEPREL}, dir(i,j), word_i, pos_i, pos_j$
$\textit{DEPREL}, dir(i,j), pos_i, word_j, pos_j$

Table 2: Binary Features for Each Part

requires the labeling of edges, as a preprocessing stage, we created a directed complete graph without multi-edges, that is, given two distinct nodes i and j, exactly two edges exist between them, one from i to j, and the other from j to i. There is no self-pointing edge. Then we labeled each edge with the highest scoring dependency relation. This complete graph was given to the CLE algorithm and the edge labels were never altered in the course of finding the maximum spanning tree. The result is the non-projective dependency tree with labeled edges.

4.2 Features

The features we used to score each part (edge) $\langle \textit{DEPREL}, i, j\rangle$ are shown in Table 2. The index i is the position of the parent and j is that of the child.

$word_j$ = the word token at the position j.

pos_j = the coarse part-of-speech at j.

$dir(i,j)$ = R if $i < j$, and L otherwise.

No other features were used beyond the combinations of the CPOS tag and the word token in Table 2.

We have evaluated our parser on Arabic, Danish, Slovene, Spanish, Turkish and Swedish, and used the ”additional features” listed in Table 2 for all languages except for Danish and Swedish. The reason for this is simply that the model with the additional features did not fit in the 4 GB of memory used in the training.

Although we could do batch learning by running the online algorithm multiple times, we run the online algorithm just once. The hardware used is an Intel Pentinum D at 3.0 Ghz with 4 GB of memory, and the software was written in C++. The training time required was Arabic 204 min, Slovene 87 min, Spanish 413 min, Swedish 1192 min, Turkish 410 min, Danish 381 min.

5 Results

The results are shown in Table 3. Although our feature set is very simple, the results were around the averages. We will do error analysis of three notable languages: Arabic, Swedish and Turkish.

5.1 Arabic

Of 4990 words in the test set, 800 are prepositions. The prepositions are the most frequently found tokens after nouns in this set. On the other hand, our head attachment error was 44% for prepositions. Given the relatively large number of prepositions found in the test set, it is important to get the preposition attachment right to achieve a higher mark in this language. The obvious solution is to have a feature that connects the head of a preposition to the child of the preposition. However, such a feature effects the edge based factoring and the decoding algorithm, and we will be forced to modify the MST algorithm in some ways.

5.2 Swedish

Due to the memory constraint on the computer, we did not use the additional features for Swedish and our feature heavily relied on the CPOS tag. At the same time, we have noticed that relatively higher performance of our parser compared to the average coincides with the bigger tag set for CPOS for this corpus. This suggests that we should be using more fine grained POS in other languages.

5.3 Turkish

The difficulty with parsing Turkish stems from the large unlabeled attachment error rate on the nouns

Language	LAS	AV	SD
Arabic	62.83%	59.92%	6.53
Danish	75.81%	78.31%	5.45
Slovene	64.57%	65.61%	6.78
Spanish	73.17%	73.52%	8.41
Swedish	79.49%	76.44%	6.46
Turkish	54.23%	55.95%	7.71
Language	UAS	AV	SD
Arabic	74.27%	73.48%	4.94
Danish	81.72%	84.52%	4.29
Slovene	74.88%	76.53%	4.67
Spanish	77.58%	77.76%	7.81
Swedish	86.62%	84.21%	5.45
Turkish	68.77%	69.35%	5.51

Table 3: Labeled and Unlabeled Attachment Score

(39%). Since the nouns are the most frequently occurring words in the test set (2209 out of 5021 total), this seems to make Turkish the most challenging language for any system in the shared task. On the average, there are 1.8 or so verbs per sentence, and nouns have a difficult time attaching to the correct verb or postposition. This, we think, indicates that there are morphological features or word ordering features that we really need in order to disambiguate them.

6 Future Work

As well as making use of fine-grained POS tags and other morphological features, given the error analysis on Arabic, we would like to add features that are dependent on two or more edges.

6.1 Bottom-Up Non-Projective Parsing

In order to incorporate features which depend on other edges, we propose Bottom-Up Non-Projective Parsing. It is often the case that dependency relations can be ordered by how close one relation is to the root of dependency tree. For example, the dependency relation between a determiner and a noun should be decided before that between a preposition and a noun, and that of a verb and a preposition, and so on. We can use this information to do bottom-up parsing.

Suppose all words have a POS tag assigned to them, and every edge labeled with a dependency relation is attached to a specific POS tag at the end point. Also assume that there is an ordering of POS tags such that the edge going to the POS tag needs be decided before other edges. For example, (1) determiner, (2) noun, (3) preposition, (4) verb would be one such ordering. We propose the following algorithm:

- Assume we have tokens as nodes in a graph and no edges are present at first. For example, we have tokens ”I”, ”ate”, ”with”, ”a”, ”spoon”, and no edges between them.

- Take the POS tag that needs to be decided next. Find all edges that go to each token labeled with this POS tag, and put them in the graph. For example, if the POS is noun, put edges from ”ate” to ”I”, from ”ate” to ”spoon”, from ”with” to ”I”, from ”with” to ”spoon”, from ”I” to ”spoon”, and from ”spoon” to ”I”.

- Run the CLE algorithm on this graph. This selects the highest incoming edge to each token with the POS tag we are looking at, and remove cycles if any are present.

- Take the resulting forests and for each edge, bring the information on the child node to the parent node. For example, if this time POS was noun, and there is an edge to a preposition ”with” from a noun ”spoon”, then ”spoon” is absorbed by ”with”. Note that since no remaining dependency relation will attach to ”spoon”, we can safely ignore ”spoon” from now on.

- Go back and repeat until no POS is remaining and we have a dependency tree. Now in the next round, when deciding the score of the edge from ”ate” to ”with”, we can use the all information at the token ”with”, including ”spoon”.

7 Conclusion

We have extended non-projective unlabeled dependency parsing (McDonald et al., 2005) to a very simple non-projective labeled dependency and showed that the parser performs reasonably well with small number of features and just one iteration of training. Based on the analysis of the Arabic parsing results, we have proposed a bottom-up non-projective labeled dependency parsing algorithm that allows us to use features dependent on more than one edge, with very little disadvantage compared to the original algorithm.

References

A. Abeillé, editor. 2003. *Treebanks: Building and Using Parsed Corpora*, volume 20 of *Text, Speech and Language Technology*. Kluwer Academic Publishers, Dordrecht.

S. Afonso, E. Bick, R. Haber, and D. Santos. 2002. “Floresta sintá(c)tica”: a treebank for Portuguese. In *Proc. of the Third Intern. Conf. on Language Resources and Evaluation (LREC)*, pages 1698–1703.

N. B. Atalay, K. Oflazer, and B. Say. 2003. The annotation process in the Turkish treebank. In *Proc. of the 4th Intern. Workshop on Linguistically Interpreteted Corpora (LINC)*.

A. Böhmová, J. Hajič, E. Hajičová, and B. Hladká. 2003. The PDT: a 3-level annotation scenario. In Abeillé (Abeillé, 2003), chapter 7.

S. Brants, S. Dipper, S. Hansen, W. Lezius, and G. Smith. 2002. The TIGER treebank. In *Proc. of the First Workshop on Treebanks and Linguistic Theories (TLT)*.

R. Bunescu and R. Mooney. 2005. A shortest path dependency kernel for relation extraction. In *Proc. of the Joint Conf. on Human Language Technology and Empirical Methods in Natural Language Processing (HLT/EMNLP)*.

K. Chen, C. Luo, M. Chang, F. Chen, C. Chen, C. Huang, and Z. Gao. 2003. Sinica treebank: Design criteria, representational issues and implementation. In Abeillé (Abeillé, 2003), chapter 13, pages 231–248.

Y.J. Chu and T.H. Liu. 1965. On the shortest arborescence of a directed graph. In *Science Sinica*, page 14:13961400.

M. Civit Torruella and M^{a} A. Martí Antonín. 2002. Design principles for a Spanish treebank. In *Proc. of the First Workshop on Treebanks and Linguistic Theories (TLT)*.

M. Collins and B. Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proc. of the 42rd Annual Meeting of the ACL*.

M. Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proc. of Empirical Methods in Natural Language Processing (EMNLP)*.

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

Y. Ding and M. Palmer. 2005. Machine translation using probabilistic synchronous dependency insertion grammars. In *Proc. of the 43rd Annual Meeting of the ACL*.

S. Džeroski, T. Erjavec, N. Ledinek, P. Pajas, Z. Žabokrtsky, and A. Žele. 2006. Towards a Slovene dependency treebank. In *Proc. of the Fifth Intern. Conf. on Language Resources and Evaluation (LREC)*.

J. Edmonds. 1967. Optimum branchings. In *Journal of Research of the National Bureau of Standards*, page 71B:233240.

J. Hajič, O. Smrž, P. Zemánek, J. Šnaidauf, and E. Beška. 2004. Prague Arabic dependency treebank: Development in data and tools. In *Proc. of the NEMLAR Intern. Conf. on Arabic Language Resources and Tools*, pages 110–117.

Y. Kawata and J. Bartels. 2000. Stylebook for the Japanese treebank in VERBMOBIL. Verbmobil-Report 240, Seminar für Sprachwissenschaft, Universität Tübingen.

M. T. Kromann. 2003. The Danish dependency treebank and the underlying linguistic theory. In *Proc. of the Second Workshop on Treebanks and Linguistic Theories (TLT)*.

G. Leonidas. 2003. Arborescence optimization problems solvable by edmonds algorithm. In *Theoretical Computer Science*, page 301:427 437.

R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proc. of the Joint Conf. on Human Language Technology and Empirical Methods in Natural Language Processing (HLT/EMNLP)*.

J. Nilsson, J. Hall, and J. Nivre. 2005. MAMBA meets TIGER: Reconstructing a Swedish treebank from antiquity. In *Proc. of the NODALIDA Special Session on Treebanks*.

J. Nivre and J. Nilsson. 2005. Pseudo-projective dependency parsing. In *Proc. of the 43rd Annual Meeting of the ACL*.

K. Oflazer, B. Say, D. Zeynep Hakkani-Tür, and G. Tür. 2003. Building a Turkish treebank. In Abeillé (Abeillé, 2003), chapter 15.

K. Simov and P. Osenova. 2003. Practical annotation scheme for an HPSG treebank of Bulgarian. In *Proc. of the 4th Intern. Workshop on Linguistically Interpreteted Corpora (LINC)*, pages 17–24.

K. Simov, P. Osenova, A. Simov, and M. Kouylekov. 2005. Design and implementation of the Bulgarian HPSG-based treebank. In *Journal of Research on Language and Computation – Special Issue*, pages 495–522. Kluwer Academic Publishers.

B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. 2004. Max-margin parsing. In *Proc. of Empirical Methods in Natural Language Processing (EMNLP)*.

L. van der Beek, G. Bouma, R. Malouf, and G. van Noord. 2002. The Alpino dependency treebank. In *Computational Linguistics in the Netherlands (CLIN)*.