Comparing Non-projective Strategies for Labeled Graph-based Dependency Parsing

Anders Björkelund *Jonas Kuhn*
Institut für Maschinelle Sprachverarbeitung
University of Stuttgart
{anders,jonas}@ims.uni-stuttgart.de

ABSTRACT
We fill a gap in the systematically analyzed space of available techniques for state-of-the-art dependency parsing by comparing non-projective strategies for graph-based parsers. Using three languages with varying frequency of non-projective constructions, we compare the non-projective approximation algorithm with pseudo-projective parsing. We also analyze the differences between different encoding schemes for pseudo-projective parsing. We find only minor differences between the encoding schemes for pseudo-projective parsing, and that the non-projective approximation algorithm is superior to pseudo-projective parsing.

KEYWORDS: Multilingual Dependency Parsing, Non-projective Parsing, Pseudo-projective Parsing.

Proceedings of COLING 2012: Posters, pages 135–144,
COLING 2012, Mumbai, December 2012.

1 Introduction

One common justification for dependency syntax is that it, in contrast to constituent syntax, can represent long-distance dependencies between words through non-projective dependencies in a more straightforward way, without the use of traces or secondary edges. Informally, a dependency tree is said to be non-projective if it cannot be drawn without crossing edges. An example is shown in Figure 1.

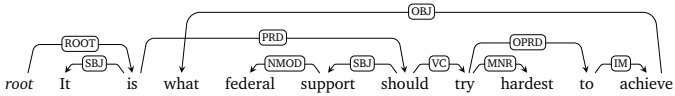

Figure 1: A non-projective sentence

Although there are decoding algorithms for graph-based parsers that are able to output non-projective trees directly (e.g. spanning tree algorithms (McDonald et al., 2005b) and ILP-based parsers (Riedel and Clarke, 2006, *inter alia*)), the chart-based algorithm of Eisner (1996), which is restricted to projective output, has shown very promising results in recent years. It typically outperforms the non-projective algorithms since it allows access to features involving pairs of edges.

A notable extension to the chart-based parsing algorithm that is able to output non-projective dependencies while still including edge-pair features is the non-projective approximation algorithm of McDonald and Pereira (2006).

Non-projective edges have also been handled by applying pre- and post-processing steps to the training and test data, allowing for the use of any labeled projective parsing algorithm, only to recover the non-projective edges after parsing, e.g. pseudo-projective parsing (Nivre and Nilsson, 2005).

In the CoNLL 2008 and 2009 Shared Tasks (Surdeanu et al., 2008; Hajič et al., 2009), some of the best systems used the chart-based parsing algorithm. Besides using slightly different feature sets, non-projective edges were handled differently – Bohnet (2009) used the non-projective approximation algorithm, while Johansson and Nugues (2008) and Che et al. (2009) used pseudo-projective parsing. Handling non-projective edges is unarguably an important aspect of a parser, however, little is known about whether one of the methods mentioned above is better than the other. With a fixed feature set, we compare pseudo-projective parsing with non-projective approximation using a state-of-the-art chart-based dependency parser (Bohnet, 2010). We also evaluate different encoding schemes for pseudo-projective parsing. More recently, highly accurate parsers that model non-projective edges directly in the parsing algorithm have been proposed, such as the ILP-based parser of Martins et al. (2010) as well as algorithms relying on non-projective head automata (Koo et al., 2010). It would be interesting to include these parsers in our study, however they only provide unlabeled trees. For now, we leave the extension of these parsers to the labeled case and the comparison to future work.

All experiments are performed on three languages that exhibit different typological properties and frequency of non-projective dependencies: Czech, English, and German. We find that non-projective approximation performs better than pseudo-projective parsing, although both methods clearly outperform a projective baseline. While similar studies have been carried out for transition-based parsers (Kuhlmann and Nivre, 2010), this is the first time non-projective strategies for graph-based algorithms are compared in a multilingual setting.

2 Background

We consider single-rooted dependency trees with one head per token, such as the one in Figure 1. We use the notation $x = x_0 \ldots x_n$ to denote a sentence with n tokens, where x_0 is a special *root* node. A labeled head-dependent relation (or edge) between a head h and dependent c with the label l is denoted $(h \xrightarrow{l} c)$. We omit the label when this is not relevant. A dependency tree for a sentence x is a set $y = \{(p_1 \xrightarrow{l_1} x_1), \ldots, (p_n \xrightarrow{l_n} x_n)\}$ of edges, such that each node except the root has exactly one head, and the graph is acyclic (i.e., it forms a single-rooted tree). A node x_i *dominates* another node x_j if x_i is an ancestor of x_j. An edge $(x_i \rightarrow x_j)$ is defined to be *projective* iff x_i dominates all words between x_i and x_j. Otherwise it is *non-projective*. Moreover, a dependency tree y is projective iff every edge is projective. Otherwise it is non-projective.

Graph-based Dependency Parsing algorithms solve the parsing problem by finding the highest scoring dependency tree for a sentence: $\hat{y} = \arg\max_y F(x, y)$, given a scoring function F. To make the search for the optimal tree tractable, the scoring function is decomposed into a sum over *factors* of the tree (McDonald et al., 2005a):

$$F(x, y) = \sum_{f \in factors(x,y)} \psi(f) \cdot w$$

where ψ is a feature-mapping function that maps a factor f to a vector in high-dimensional feature space and w a weight vector.

The chart-based algorithm of Eisner (1996) has the advantage that it can incorporate second-order factors while still remaining computationally feasible. The version we use is due to Carreras (2007) and makes use of second-order factors including sibling and grandchild relations. This factorization offers access to valuable features but comes at the cost of a time complexity of $O(Ln^4)$, where L is the number of edge labels. To reduce the impact of the factor L, edge filters are applied (Bohnet, 2010), constraining the search of edge labels to those observed in training for the same head and dependent POS-tags; this reduces execution time considerably.

The **Non-projective Approximation** algorithm (McDonald and Pereira, 2006) exploits the observation that, although the chart-based parsing algorithm is only able to output projective structures, the weight vector used to score the factors of the tree is not limited in this respect. Hence, starting from the highest scoring projective tree output by the chart-based algorithm, it iteratively tries to reattach all tokens, one at a time, everywhere in the sentence as long as the tree property holds. In each iteration, the highest scoring move, i.e., the move that increases the total score of the tree the most is executed. The process terminates when the increase is below a certain threshold.

Pseudocode[1] for the algorithm is given in Figure 2. The auxiliary function ALLOWED-LABELS(h, c, x) returns the labels permitted by the edge filters and TREE(y) returns true if y is a tree, and false otherwise. The notation $y[j \xrightarrow{k} i]$ denotes a tree identical to y, except that the head of x_i is replaced by x_j, and its label by k. This process could potentially take exponential time, although this is not a problem in practice, and the algorithm typically halts after a few moves (McDonald and Pereira, 2006).

Pseudo-projective Parsing can be used with any labeled projective parsing algorithm. The training data is pre-processed by applying lifting operations to the non-projective edges while

[1] From McDonald and Pereira (2006), but adapted to the labeled case.

input Sentence x, tree y, scoring function F, threshold t
returns (Non-)projective tree y'
  $score \leftarrow F(x, y)$
  while true
    $m \leftarrow -\infty, c \leftarrow -1, p \leftarrow -1, l \leftarrow null$
    for i : 1..n
      for j : 0..n
        $y' \leftarrow y[j \rightarrow i]$
        if $\neg$TREE(y') **continue**
        for $k \in$ ALLOWED-LABELS(i, j)
          $y' \leftarrow y[j \xrightarrow{k} i]$
          $s \leftarrow F(x, y')$
          if $s > m$
            $m \leftarrow s, c \leftarrow i, p \leftarrow j, l \leftarrow k$
    if $m - score > t$
      $score \leftarrow m, y \leftarrow y[p \xrightarrow{l} c]$
    else return y

Figure 2: Non-projective approximation

encoding information about the lifts into the edge labels in the tree. The parser is then trained on these projective trees and learns the encoding of the liftings. An inverse transformation that recovers the non-projective edges is then applied to the parser output (Nivre and Nilsson, 2005). This way, non-projective edges are introduced in a post-processing step allowing for the use of any projective parsing algorithm.

Nivre and Nilsson (2005) propose three label encoding schemes differing in terms of the granularity in the marking of lifts: **Head** - each lifted edge is marked as lifted, and additionally receives the label of its original head; **Path** - the lifted edge is marked as lifted, and all heads along the path it was lifted through get marked as having had a dependent lifted; **HeadPath** - a combination of Head and Path, where the lifted edge is marked as in the Head scheme *and* all heads along the path it was lifted get marked as in the Path scheme. Figure 3 shows the dependency tree from Figure 1 when the edge of *what* has been lifted using the HeadPath scheme.

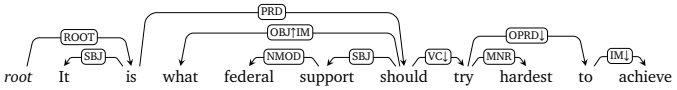

Figure 3: The same sentence as in Figure 1, but the non-projective edge has been lifted using the HeadPath scheme

Each of the encoding schemes leads to an increase in the set of edge labels (up to $2n(n+1)$ new labels for HeadPath (Nivre and Nilsson, 2005)), and thus to an increase in parsing time. Additionally, there is a possible data sparsity issue as a result of very infrequent lifted edges. It has therefore been proposed to cap the number of newly introduced labels and retain only the m most frequent new labels in the training data (Johansson and Nugues, 2008).

The inverse transformation looks for edges that are marked as lifted (i.e. of the form $l\uparrow$ or $l\uparrow l_{np}$). It then does a breadth-first search, starting from the head of this edge, looking for a new head for the dependent. Details depend on the encoding scheme: For Head, search halts at

the first token whose edge label matches the lifted edge (i.e. the first token with the label l_{np}); for Path, only the subtrees marked with ↓ are considered, and search halts at the deepest edge marked with ↓; for HeadPath the edge is reattached at the deepest token that matches l_{np}↓. Additionally, for HeadPath, the inverse transformation of Head is used as a fallback in case the search fails (Nivre and Nilsson, 2005).

3 Experiments and Results

The parser we employ (Bohnet, 2010)[2] uses non-projective approximation by default.[3] In the experiments involving pseudo-projective parsing, we switched off the non-projective approximation.

We use the three data sets from the CoNLL 2009 Shared Task (Hajič et al., 2009) that contain non-projective edges, namely Czech, English, and German. We use the standard data split. Since the frequency of non-projective edges is relatively small, we resort to a 10-fold cross-validation on the training set in order to get more reliable figures. A breakdown of the training sets for each language is shown in Table 1. We use the "predicted" layers of annotation, i.e. output of standard POS-taggers etc., for a realistic evaluation. We report labeled attachment score (LAS), i.e. the percentage of correctly assigned heads and edge labels, and labeled exact match (LEM) for complete sentences. The scores are micro-averaged, i.e., the parser output for all folds are concatenated and compared against the whole training set.

Following Kuhlmann and Nivre (2010) we also compute precision and recall for non-projective edges. They define recall as the percentage of tokens that have a non-projective dependency in the gold standard and receive the correct head and label in the parser output. Precision is defined as the percentage of tokens getting a non-projective dependency in the parser output receiving the correct head and label. As Kuhlmann and Nivre (2010) point out, these definitions are somewhat unusual since they have different numbers of true positives, and combining them through the unweighted harmonic mean is not meaningful. Hence we do not present any F-measures in the tables.

	Sentences	#NP edges (%)	% NP sentences
Czech	38,727	12,112 (1.86%)	22.42%
English	39,279	3,724 (0.39%)	7.63%
German	36,020	15,123 (2.33%)	28.10%

Table 1: Breakdown of the training sets of each language. NP means non-projective.

In the experiments we want to investigate three questions: (1) Are pseudo-projective parsing and non-projective approximation equally good, or is one better than the other? (2) What is the difference between the different label encoding schemes for pseudo-projective parsing? (3) How badly does label capping for pseudo-projective parsing degrade performance?

We also trained a baseline parser on trees that were projectivized, but received no augmented edge labels. All results are shown in Table 2. The rows with subscripted pseudo-projective encodings denote parsers that used a label cap (of 30). As an indication of how often the different parsers produce non-projective edges, the total number of non-projective edges are given in the last column.

[2] `http://code.google.com/p/mate-tools`

[3] The threshold t has already been tuned to 0.3 by Bohnet (2010), and we keep this fixed throughout the experiments.

During training and testing we experienced that the capped parsers required about as much time as the parsers that use the non-projective approximation, while the uncapped HeadPath parsers took about twice as much time. This is because the increase in decoding time due to the increased set of labels for the pseudo-projective parsers is roughly canceled out by the call to the non-projective approximation algorithm. When the cap is dropped, however, the pseudo-projective parsers are overwhelmed by the number of new labels and consequently need more time for decoding.

	All		Non-projective		
Czech	LAS	LEM	P	R	#NP
Baseline	81.10	25.90		5.40	0
Path_{30}	81.75	27.28	**76.86**	40.13	5,748
Head_{30}	**81.86**	**27.67**	71.23	**44.23**	6,868
HeadPath_{30}	81.73	27.51	71.64	39.28	5,973
Path	81.78	27.35	**76.48**	41.03	5,868
Head	**81.94**	27.87	70.10	48.18	7,716
HeadPath	**81.94**	**27.94**	70.40	**48.82**	7,727
NPA	**82.11**	**28.40**	68.95	**65.72**	11,394
English	LAS	LEM	P	R	#NP
Baseline	89.73	28.95		7.44	0
Path_{30}	89.74	29.08	**75.42**	23.58	834
Head_{30}	**89.80**	**29.37**	61.47	**39.02**	1,983
HeadPath_{30}	**89.80**	29.27	63.21	38.94	1,911
Path	89.77	29.41	**75.85**	23.85	824
Head	**89.83**	**29.44**	61.33	39.98	2,061
HeadPath	89.82	29.40	60.55	**40.44**	2,066
NPA	89.80	**29.52**	49.46	**43.77**	3,787
German	LAS	LEM	P	R	#NP
Baseline	86.01	30.94		4.74	0
Path_{30}	86.60	33.44	**70.36**	36.05	6,778
Head_{30}	**86.74**	**33.77**	62.45	40.12	8,741
HeadPath_{30}	86.64	33.58	64.32	**40.24**	8,416
Path	86.61	33.62	**69.78**	36.53	6,885
Head	**86.79**	**33.74**	60.27	41.65	9,424
HeadPath	86.75	33.66	60.78	**42.14**	9,359
NPA	**87.05**	**34.99**	65.37	**58.47**	14,208

Table 2: Results for pseudo-projective parsing and non-projective approximation (NPA). P and R denote precision and recall for non-projective edges. #NP denotes the total number of predicted non-projective edges.

Not surprisingly, our results indicate that handling non-projective edges is much more important in Czech and German. In these languages, the baseline is clearly outperformed by all other parsers. In English, non-projective approximation, and uncapped Head, HeadPath ($p < 0.001$), and Path ($p < 0.05$) are all significantly better than the baseline (using a paired t-test).

The non-projective approximation has a considerably higher recall and the amount of non-projective edges is closer to the real number (cf. Table 1), yet the precision does not seem to be severely penalized. The low recall for the pseudo-projective parsers is explained by the fact that these transformations rely on predicting corresponding labels (depending on encoding scheme) in *two* places – the predicted projective head, with an augmented label, and the reattachment location. Since the parser as such is not aware of this interdependency, it is possible that it predicts a tree with a lifted edge, but no appropriate place to reattach it, in which case the edge is left in place. The non-projective approximation algorithm does not have the same limitation

as it simply moves single edges around as long as it increases the overall scores.

Pseudo-projective parsing vs Non-projective approximation. Comparing non-projective approximation with the uncapped pseudo-projective parsers, we find that in Czech and German the non-projective approximation is significantly better than all the pseudo-projective parsers ($p < 0.001$). The difference in LAS, compared to the best pseudo-projective encoding, is roughly 0.25. Although this may seem tiny, the increase in exact match (LEM) is more than a point for German and about half a point for Czech. This improvement is important since, ultimately, a correct analysis of an entire tree is what we aim for. For English, the scores are much closer and only the improvement for the non-projective approximation over Path is significant ($p < 0.05$). The improvements in exact match are also rather small.

Pseudo-projective parsing. Considering pseudo-projective parsing alone, Path consistently predicts the fewest non-projective edges, leading to the highest precision but almost always the lowest recall. This is reasonable, as the requirement for Path to induce a non-projective edge is that it predicts both a lifted edge, *and* a path of edges from the head to an appropriate place to reattach it. Since these augmented labels are rather rare, it seems like the parser suffers from sparsity issues during training and underpredicts these edges.

The recall figures are highest for HeadPath, although it lags a bit behind Head for the capped version in Czech and English. This is because some of the most frequent labels in the HeadPath scheme are of the form $l{\downarrow}$, which means that the parser learns only very few lifted edges (i.e. edges of the form $l{\uparrow}l_{np}$).

Besides the slightly lower scores for Path, the overall difference between the encoding schemes appear very small. With the cap, the Head encoding appears to be a bit better, but HeadPath catches up when the cap is dropped.

Capping the number of new labels leads to slightly lower results in Czech and German, however only the increases for HeadPath against its capped counterpart in Czech ($p < 0.001$) and German ($p < 0.01$) are statistically significant.

4 Conclusion

We presented a comparative analysis of the non-projective approximation algorithm and pseudo-projective parsing using a graph-based parser. Our experimental results indicate that the non-projective approximation algorithm outperforms pseudo-projective parsing in overall accuracy for Czech and German. For English, where non-projective dependencies are relatively infrequent, the strategies are rather tied, albeit better than the baseline. In conclusion, the non-projective approximation algorithm is clearly superior for languages that more often exhibit long-distance dependencies.

Our evaluation of the different encoding schemes for pseudo-projective parsing reveals that the schemes are roughly equivalent in overall performance, and that capping the number of labels results only in a slight performance degradation.

In the future, we aim to extend our study to include parsers that handle non-projective edges in the immediate parsing process. We also intend to look more closely at the underlying phenomena that give rise to the non-projective edges.

Acknowledgments

This work was funded by the Deutsche Forschungsgemeinschaft (DFG) via the SFB 732 "Incremental Specification in Context", project D8.

References

Bohnet, B. (2009). Efficient parsing of syntactic and semantic dependency structures. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pages 67–72, Boulder, Colorado. Association for Computational Linguistics.

Bohnet, B. (2010). Top accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 89–97, Beijing, China. Coling 2010 Organizing Committee.

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

Che, W., Li, Z., Li, Y., Guo, Y., Qin, B., and Liu, T. (2009). Multilingual dependency-based syntactic and semantic parsing. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pages 49–54, Boulder, Colorado. Association for Computational Linguistics.

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

Hajič, J., Ciaramita, M., Johansson, R., Kawahara, D., Martí, M. A., Màrquez, L., Meyers, A., Nivre, J., Padó, S., Štěpánek, J., Straňák, P., Surdeanu, M., Xue, N., and Zhang, Y. (2009). The conll-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pages 1–18, Boulder, Colorado. Association for Computational Linguistics.

Johansson, R. and Nugues, P. (2008). Dependency-based syntactic–semantic analysis with propbank and nombank. In *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 183–187, Manchester, England. Coling 2008 Organizing Committee.

Koo, T., Rush, A. M., Collins, M., Jaakkola, T., and Sontag, D. (2010). Dual decomposition for parsing with non-projective head automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1288–1298, Cambridge, MA. Association for Computational Linguistics.

Kuhlmann, M. and Nivre, J. (2010). Transition-based techniques for non-projective dependency parsing. *Northern European Journal of Language Technology*, 2(1):1–19.

Martins, A., Smith, N., Xing, E., Aguiar, P., and Figueiredo, M. (2010). Turbo parsers: Dependency parsing by approximate variational inference. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 34–44, Cambridge, MA. Association for Computational Linguistics.

McDonald, R., Crammer, K., and Pereira, F. (2005a). Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 91–98, Ann Arbor, Michigan. Association for Computational Linguistics.

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

McDonald, R., Pereira, F., Ribarov, K., and Hajic, J. (2005b). Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

Nivre, J. and Nilsson, J. (2005). Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 99–106, Ann Arbor, Michigan. Association for Computational Linguistics.

Riedel, S. and Clarke, J. (2006). Incremental integer linear programming for non-projective dependency parsing. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 129–137, Sydney, Australia. Association for Computational Linguistics.

Surdeanu, M., Johansson, R., Meyers, A., Màrquez, L., and Nivre, J. (2008). The conll 2008 shared task on joint parsing of syntactic and semantic dependencies. In *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 159–177, Manchester, England. Coling 2008 Organizing Committee.