Vine parsing augmented Italian treebanks

Anders Søgaard and Christian Rishøj

Center for Language Technology
University of Copenhagen

Abstract. This brief article describes our contribution to the EVALITA 2009 Parsing Task, dependency track. The TUT and ISST treebanks are augmented with additional features. MIRA is used to find a weight matrix suited for the Covington algorithm, which is subsequently skewed by discriminatively learned hard constraints on dependency lengths. Our skewed algorithm is linear time and thus asymptotically faster than the cubic time Covington algorithm, but increases in performance are insignificant. Our overall system is thus non-competitive. It is shown, however, that it does contribute significantly to the performance of a more competitive ensemble-based system.

Keywords: Dependency Parsing, vine parsing, treebank augmentation.

1 Introduction

In the context of dependency parsing, vine parsing amounts to hard constraints on dependency length. Hard constraints on dependency length limit the search space considerably, and vine parsers such as [1–3] are *very* fast. The three models cited also significantly improve the performance of the non-vine baseline parser by adding constraints on dependency length. In all cases, however, the baseline parsers are non-competitive and informed only by simple maximum likelihood estimates.

The main objective of this work is to test whether vine parsing *easily* scales to state-of-the-art dependency parsing, i.e. whether discriminatively learned constraints on dependency length can be used directly to boost such models. In our experiments we use a (first order) MIRA-informed Covington algorithm [4, 5] as our baseline parser. Our results are largely negative, but it is important to see that this could not be predicted in advance:

- Information about dependency length is coarse-grained in the models that guide state-of-the-art dependency parsers, since lengths are binned to avoid data sparseness and/or there is a back-off strategy.
- Since most features are also rather local for tractability reasons, this means that the precision of dependency parsers decrease considerably with dependency length [6].

It is also important to note that our results do not disqualify the use of hard constraints in dependency parsing with rich models. On the contrary. The model

used in our experiments speeds up parsing considerably, taking us from cubic time to linear time, with little or no cost in accuracy. Furthermore, there may be more intelligent ways of importing constraints on dependency length in your models than the somewhat naïve approach taken here.

The overall architecture of the system is as follows: The MIRA procedure is run on an *augmented* (see Sect. 2) treebank and outputs a model. The Covington algorithm [4] is applied to produce a labeled parse. A weight matrix for unlabeled arc candidates is constructed simultaneously. The labeled parse is stored for later use, since our discriminative perceptron-style procedure for finding constraints on dependency length only works for unlabeled dependency parsing. The labeler subsequently greedily projects these labels onto the unlabeled parses. Using 10 90-10 splits of the training data, a range of POS-specific hard constraints on dependency length are tested, and those that lead to improvements are stored in a dictionary for that fold. The 10 fold dictionaries are combined by one out of several unification strategies (Sect. 3). A vine parsing algorithm is applied to the test data which makes use of the combined dictionary of POS-specific hard constraints on dependency length and the original model for the Covington algorithm. In our experiments we use the Italian treebanks: TUT, which comes in two sections, a newspaper section TUT(n) and a law text section TUT(l), JRC and ISST.

The treebank augmentation referred to above is done in a simple conversion script. The implementation of MIRA documented in McDonald et al. [5] is used with a default 10 iterations (considering 5 best parses). The Covington algorithm and the procedure for subsequent learning of hard constraints on dependency length were implemented in Java, resp. Python. The labeler was also implemented in Python. All system components and scripts that pipeline the overall procedure are available upon request.

Our choice of using the Covington algorithm as our point of departure was motivated by a quick and dirty experiment running four different parsing algorithms on the first fold in the TUT(l), trained on the remaining 9/10. The algorithms, apart from the Covington algorithm, are the vine parser in Søgaard and Kuhn [3] (SK), a spanning tree algorithm with limited cycle contraction (k-MST), the LR algorithm in Sagae and Tsujii [7] (KSDep), the transition-based arc-eager algorithm in Nivre et al. [8] (MaltParser), and the spanning tree algorithm described in McDonald et al. [9] (MSTParser). KSDep was trained with a 100 iterations; 500 iterations led to worse scores (83.53/77.42). The unlabeled Covington(u) algorithm is trained with dependency relations removed.

	SK	k-MST	KSDep	MaltParser	Covington(l)	MSTParser(u)	Covington(u)
UAS	70.53	84.18	84.26	84.76	86.03	86.14	**86.93**
LAS	-	-	77.67	79.48	80.71	-	-

Considering 5-best parses in training Covington(u) scores 87.69. This parameter setting was naïvely imported to the other treebanks too. Including second order features boosts performance too, but our vine parsing algorithm is currently only implemented for first order features.

2 Treebank augmentation

Three treebank augmentation schemes were tested.

The scheme PARENTHESIS+QUOTED_PHRASE is based on experiments with the EVALITA 2007 release of the TUT treebank, documented in Rishøj [10]. It simply adds a feature to tokens inside parentheses and quoted phrases. In 10-fold cross-validation on the training data, this leads to an overall significant improvement on TUT, but in fact it is only our predictions on TUT(l) that improves. Consequently, we did not apply this scheme to TUT(n). The scheme made our ISST parses worse. None of the schemes were applied to JRC.

The scheme COMBINETREEBANKS adds the predicted dependency label of each word from a parser trained on the other treebank to its features. So, for example, we train a parser on TUT and run it on (the complete) ISST and add the predicted dependency label to each word's feature list. The scheme is in a way comparable to parser stacking, e.g. [11], except that we stack two distinct treebanks. This leads to a small, but significant improvement, but the number of features extracted in training when this scheme is used is about three times as big as in our baseline system. This schema was only used for the TUT treebanks.

The third and final scheme LEMMA is not really an augmentation. It simply copies lemma information to the word field in the treebank and removes information in the lemma field. This was used in all treebanks, except JRC. The intuition is that lemma and morphological features are enough to reconstruct what is important, and that skipping word forms removes undesirable noise from our training data.

3 Constraints on dependency length

Our overall architecture gives us 10 dictionaries of POS-specific hard constraints on dependency length for each treebank. How do we combine these dictionaries? In our submitted results, we somewhat naïvely used the strategy T=1 (see below), but in this paper we present experiments with three different strategies. T is intuitively a threshold for how many folds a constraint must occur in to be included in the final dictionary. For each POS **maj**(POS) is the class of left, resp. right, constraints if arcs from items labeled POS are more frequently constrained to the left, resp. right. **maj**(POS,1) is then the set of upper bounds on arcs to the left, resp. right, in the various folds, while **maj**(POS,2) is the set of upper bounds on arcs to the right, resp. left. A constraint is written $\langle$ POS, i, $j\rangle$ with i, j upper bounds on length; it says that any arc from an item labeled POS can span at most i items if to the left and j items if to the right.

T=1 For each POS include the constraint **max(maj**(POS,1)) and **max(maj**(POS,2)).

T=5 For each POS include the constraint **max(maj**(POS,1)) and **max(maj**(POS,2)) if and only if $|\mathbf{maj}(\text{POS})| \geq 5$.

G+T=5 For each POS include the constraint **min(maj**(POS,1)) and **max(maj**(POS,2)) if and only if $|\mathbf{maj}(\text{POS})| \geq 5$.

A superior strategy would be to test all constraints in the dictionary by 10-fold cross-validation. Such experiments have not been carried out yet.

4 Results

This section presents the submitted results and the results obtained with the two other unification strategies. Note that the two other strategies lead to much better results. EM is the number of exact matches. 'All' is the average score for TUT and JRC.

T=1	LAS	UAS	EM	**T=5**	LAS	UAS	EM	**G+T=5**	LAS	UAS	EM
TUT(n)	72.84	81.93	10.00	TUT(n)	72.90	82.21	9.00	TUT(n)	**73.57**	83.28	8.00
TUT(l)	86.04	90.27	24.00	TUT(l)	**87.48**	91.93	25.00	TUT(l)	86.18	90.54	11.00
JRC	81.85	86.30	2.50	JRC	**84.16**	89.19	10.00	JRC	84.16	89.19	10.00
All	80.42	-	-	All	81.73	-	-	All	-	-	-
ISST	78.51	85.81	12.31	ISST	**79.25**	86.61	12.69	ISST	79.19	86.51	12.69

5 Other languages?

To test the applicability of POS-specific hard constraints on dependency length in the context of MIRA-informed projective dependency parsing, we present results on other languages too, incl. data sets from the CONLL-X Shared Task and EVALITA 2007.

Note that results on data sets from the CONLL-X Shared Task are excl. punctuation for comparability with shared task results. The official results reported using the MSTParser [9], the best scoring system in the contest, are listed for comparison (MST). The MSTParser used second order features to obtain the repored results, but note that our baseline system (BL) is slightly better than the reported results in the case of Turkish.

LAS	BL	T=1	T=5	G+T=5	Δ	MST
Arabic	**78.26**	77.41	77.98	77.88	-0.28	79.34
Danish	**89.48**	89.00	89.32	88.96	-0.16	90.58
Slovene	81.83	81.14	**81.85**	80.72	0.02	83.17
Swedish	**88.17**	87.87	88.15	87.93	-0.02	88.93
Turkish	74.83	74.11	**74.89**	74.75	0.06	74.67

Our vine parsing algorithm is only better than our baseline Covington algorithm in 2/5 cases, and never *significantly* so ($p \leq 0.05$). The relevant p-value is 0.46 for Slovene, 0.10 for Turkish. The results confirms that T=5 is generally the best strategy of the three strategies proposed here. Our results are only better than the official CONLL-X Shared Task results also reported in [9] in the case of Turkish.

Finally, we add results on the EVALITA 2007 data sets for comparability with last year's results, incl. punctuation. AttSimi lists the results submitted by [12], ranked as the best performing statistical parser in the shared task:

LAS	BL	T=1	T=5	G+T=5	Δ	AttSim
TUT(l)	**91.87**	91.79	**91.87**	**91.87**	0.00	91.37
TUT(n)	85.49	84.39	85.49	85.49	0.00	85.49

Any increase over a clean run of a MIRA-informed Covington algorithm is due to the treebank augmentation schemes.

Ensemble-based dependency parsing

To turn our non-competitive parser into a more competitive one we build an ensemble of our parser and some of the parsers tested in our first preliminary experiments, incl. KSDep, MaltParser (w. CONLL'07 settings for Italian), MSTParser (2nd order) and the MLE-informed non-projective vine parser in [3] (SK). Our procedure was simple: Attachment and labeling were treated as independent procedures. A 15 nearest neighbor classifier trained on the predictions of our input classifiers (10-fold CV) was used to weight each candidate arc. The final weight of an arc is the product of the classifier weight and the vote + 0.5. This means that our ensemble can learn to produce arcs that are not labeled. The same procedure was used for dependencies, except using Naive Bayes with 100-estimates. The algorithm was run twice on TUT(l) with the predictions from first round as input features for the second run. Our ensemble produced 20 label predictions in total not predicted by any input classifier, of which 10 were correct. Ours(-) is the ensemble performance with our vine parser's predictions removed.

	LAS	UAS	LA
SK	-	81.29	-
KSDep	66.68	76.89	75.45
MST	76.45	80.03	86.39
Malt	**90.49**	**93.50**	93.59
Ours	90.41	**93.50**	**94.46**
Ours(-)	88.62	92.89	93.41

Our results point to a disadvantage of using LAS as evaluation measure in dependency parsing. The set of correct arcs wrt. LAS is the intersection of the correct arcs wrt. UAS and the correct arcs wrt. LA. Consequently, LAS says nothing about correctness in terms of UAS and LA. Our UAS and LA scores are better than those of the optimized MaltParser in the above, but our LAS is considerably worse.[1] The main point, however, is not the improvement over MaltParser, but the contribution of our vine parser to this ensemble.

Acknowledgement

Thanks to Martin Haulrich for technical assistance with MIRA learning software and to Jonas Kuhn for letting us use part of his code in our scripts and in our implementation of k-MST.

[1] A similar result was obtained in an ensemble experiment on TUT(n) using only our predictions and those of the MaltParser. MaltParser scored LAS=**76.94**/UAS=85.80/LA=81.82, while our ensemble scored LAS=76.77/UAS=85.80/LA=**84.18** in second round and LAS=**76.94**/UAS=85.80/LA=**84.74** in sixth round (where it stabilizes). By alternating classifiers we obtained LAS=**77.05**/UAS=85.80/LA=**84.29**.

References

1. Eisner, J., Smith, N.A.: Parsing with soft and hard constraints on dependency length. In: Proceedings of IWPT. Vancouver, Canada (2005)
2. Dreyer, M., Smith, D.A., Smith, N.A.: Vine parsing and minimum risk reranking for speed and precision. In: Proceedings of CONLL. New York, NY (2006)
3. Søgaard, A., Kuhn, J.: Using a maximum entropy-based tagger to improve a very fast vine parser. In: Proceedings of IWPT (2009)
4. Covington, M.: A fundamental algorithm for dependency parsing. In: Proceedings of ACM Southeast (2001)
5. McDonald, R., Crammer, K., Pereira, F.: Online large-margin training of dependency parsers. In: Proceedings of ACL (2005)
6. Nivre, J., McDonald, R.: Integrating graph-based and transition-based dependency parsers. In: Proceedings of ACL-HLT (2008)
7. Sagae, K., Tsujii, J.: Dependency parsing and domain adaptation with lr models and parser ensembles. In: Proceedings of EMNLP-CONLL (2007)
8. Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., Marinov, S., Marsi, E.: MaltParser: a language-independent system for data-driven dependency parsing. Natural Language Engineering, vol. 13, issue 2, pp. 95-135 (2007)
9. McDonald, R., Lerman, K., Pereira, F.: Multilingual dependency analysis with a twostage discriminative parser. In: Proceedings of CONLL (2006)
10. Rishøj, C.: Feature engineering in data-driven dependency parsing. Master's thesis, University of Copenhagen (2009)
11. Martins, A., Das, D., Smith, N., Xing, E.: Stacking dependency parsers. In: Proceedings of EMNLP (2008)
12. Attardi, G., Simi, M.: DESR at the EVALITA dependency parsing task. In: Proceedings of EVALITA 2007 (2007)