A System for Experiments with Dependency Parsers

Kiril Simov, Iliana Simova, Ginka Ivanova, Maria Mateva, Petya Osenova

Linguistic Modelling Laboratory, IICT-BAS Sofia, Bulgaria
kivs@bultreebank, iliana@bultreebank.org, givanova@bultreebank.org,
mateva.maria@gmail.com, petya@bultreebank.org

Abstract

In this paper we present a system for experimenting with combinations of dependency parsers. The system supports initial training of different parsing models, creation of parsebank(s) with these models, and different strategies for the construction of ensemble models aimed at improving the output of the individual models by voting. The system employs two algorithms for construction of dependency trees from several parses of the same sentence and several ways for ranking of the arcs in the resulting trees. We have performed experiments with state-of-the-art dependency parsers including MaltParser (Nivre et al., 2006), MSTParser (McDonald, 2006), TurboParser (Martins et al., 2010), and MATEParser (Bohnet, 2010), on the data from the Bulgarian treebank – BulTreeBank. Our best result from these experiments is slightly better then the best result reported in the literature for this language (Martins et al., 2013).

Keywords: Dependency Parsing, Ensemble Models for Parsing, Bulgarian Dependency Parsing, Treebanking

1. Introduction

Training dependency parsing models for a given language is an easy task if there exists a treebank for it. The main challenge is to construct the best possible parser exploiting the current available technology. Besides using one of the growing number of available parsers, one generally has two paths of improving: (1) developing a new framework for parsing; or (2) combining the strengths of existing ones. Since the first option is a challenge for most of the groups, we decided to build a system which would support researchers to cope with the second one. Here we present a system for experimenting with different parsing models and combining their outputs in an attempt to obtain better final results.

Our focus on ensemble models for parsing is motivated by a number of works like (McDonald and Nivre, 2007) and our previous experiments with MaltParser and MSTParser (Simov et al., 2013) which have shown the usefulness of this technique. In our original experiment we trained fourteen different parsing models using two of the most popular parsing systems: MaltParser and MSTParser. For this task we experimented with different parsing algorithms and sets of features. The result was a parsebank containing fourteen versions of the original treebank produced by each of the models. We used this parsebank for experimenting with methods for combining the outputs of different parsers. The choice of method was influenced by questions like: whether we would like to have complete dependency trees or also partial trees are acceptable; how the alternative arcs for a given node are selected - voting or machine-learning approaches; which measure we would like to improve (global for the whole tree or local for some subtrees).

The experience gained during these experiments was used for implementing a system to facilitate the whole process. For testing the system we have trained new models using MSTParser, TurboParser, and MATEParser. A combination of the new seven models and the existing fourteen was used to carry out the new voting experiments reported here. The best parser combination we obtained performs comparably to the best dependency parsing results for Bulgarian reported in literature.

The structure of the paper is as follows: first we report on the construction of a parsebank of parses; then we present the techniques for combining parsing models via voting and machine learning; in section 4 we describe a system which performs all experiments via parameters we identified during combination of parsers. The last section concludes the paper.

2. Preliminary Experiments on Bulgarian Dependency Parsing

In this section we present the initial experiments we performed in order to assemble different parsing models for Bulgarian. Our work is inspired by the in-depth analysis of performance of two of the most influential dependency parsing models: transition-based and graph-based, presented in (McDonald and Nivre, 2007). This analysis shows that the two frameworks make different errors on the same training and test datasets. The authors conclude the paper by proposing three approaches for using the advantages of both frameworks: (1) Ensemble systems – weighted combinations of both systems' outputs; (2) Hybrid systems – design of a single system integrating the strengths of each framework; and (3) Novel approaches – based on a combination of new training and inference methods. In their further work (see (Nivre and McDonald, 2008)) the authors present a hybrid system that combines the two models. The work presented in the paper is along the lines of their first suggestion – a system to facilitate the integration of the outputs of several parsing models.

2.1. Parser Models and Extension of the Treebank

The data used throughout our experiments consists of the dependency conversion[1] of the HPSG-based Treebank of Bulgarian – the BulTreeBank. This data set contains non-projective dependency trees, which are more suitable for

[1] www.bultreebank.org/dpbtb/

	MLT01	MLT02	MLT03	MLT04	MLT05	MLT06	MLT07
LAS	0.835	0.788	0.843	0.809	0.825	0.820	0,863
UAS	0.842	0.887	0.860	0.869	0.869	0.886	0.899

	MLT08	MLT09	MLT10	MLT11	MLT12	MST01	MST02
LAS	0.848	0.871	0.844	0.851	0.847	0.852	0.828
UAS	0.890	0.908	0.886	0.887	0.888	0.898	0.872

	MST03	MST04	MST05	MST06	TURBO01	TURBO02	MATE01
LAS	0.85	0.849	0.858	0.856	0.885	0.888	0.891
UAS	0.902	0.901	0.911	0.91	0.924	0.927	0.929

Table 1: Average LAS and UAS scores from the 10-fold cross validation for each of the tested parsing models.

describing the relatively free word order of Bulgarian sentences.

Initially we used it to train 12 models using MaltParser (Nivre et al., 2006) with different parsing algorithm settings and features, and two models with MSTParser (McDonald, 2006) with two different sets of features.

We evaluated each parsing model using 10-fold cross validation. The treebank was divided into 10 parts, and each model was trained using 90% of the data and tested on the remaining 10% for each of the 10 data partitions. The trained model was applied on the corresponding test set for evaluation purposes, but also the predicted parses for each sentence were stored. In this way, for each model we constructed a treebank of the suggested parses. At the end we had the original treebank plus fourteen treebanks of parses – the *parsebank*.

For the MSTParser models we selected the two best performing models on average with major difference in the scope of features. The first model uses features over a single edge, while the second one uses features over pairs of adjacent edges. The rest of the parameters were chosen for the parser's optimal labeled and unlabeled accuracy, on average: features for MST01 model: *loss-type*: punc; *second-order*: false; *training-iterations*: 15; *trainingk*: 1; *decode-type*: non-proj; *create-forest*: true; for MST02 model: *loss-type*: punc; *second-order*: true; *training-iterations*: 15; *trainingk*: 1; *decode-type*: non-proj; *create-forest*: true.

For MaltParser we chose the following three algorithms: Covington non-projective, Stack Eager and Stack Lazy. According to the Covington algorithm, each new token is attempted to be linked to the preceding token. In our study, we configured the Covington model with root and shift options set to true. During the parsing process, the root could be treated as a standard node and attached with a RightArc transition. The option Shift = true allows the parser to skip remaining tokens in Left. The Stack algorithms use a stack and a buffer, and produce a tree without post-processing by adding arcs between the two top nodes on the stack. Via a swap transition, we obtain non-projective dependency trees. The difference between the Eager algorithm and the Lazy algorithm is the time when the swap transition is applied (as soon as possible for the first algorithm and as long as possible respectively for the second one). The execution of algorithms with LIBLINEAR method is faster than algorithms with LIBSVM method and results are better. On the basis of these six models we constructed additional six ones by extending the set of node features.

Table 1 contains the average Labeled Attachment Scores (LAS) and Unlabeled Attachment Scores (UAS) for the fourteen models obtained in the 10-fold cross validation (MLT01-MLT12 and MST01-MST02).

2.2. Combining Parses by Voting

We use two algorithms for the construction of a single dependency tree from all parser models' predictions.

The first algorithm, denoted `LocTr`, is reported in (Attardi and Dell'Orletta, 2009). It constructs the dependency tree incrementally starting from an empty tree and then selecting the arc with the highest weight that could extend the current partial tree. The algorithm chooses the best arc *locally*.

The second algorithm, denoted `GloTr`, is the Chu-Liu-Edmonds algorithm for maximal spanning tree implemented in the MSTParser (McDonald, 2006). This algorithm starts with a complete dependency graph including all possible dependency arcs. Then it selects the maximal spanning tree on the basis of the weights assigned to the potential arcs. The arcs that are not proposed by any of the parsers are deleted. The algorithm is *global* with respect to selection of arcs.

These two voting algorithms are included in our system for experiments with dependency parsers. The user can specify which one should be used in their experiments, or alternatively compare the performance of both.

We investigated three voting settings: (1) the arcs in the dependency tree are ranked by the number of the parsers that predicted them; (2) the arcs are ranked by the sum of UAS measures for all parsers that predicted them; and (3) the arcs are ranked by the average of the UAS measures of the parsers that predicted them.

We ran both algorithms (`LocTr` and `GloTr`) for construction of dependency trees on the basis of combinations of dependency parses over results from all models and only for some of the models. Although the results are not drastically different, they show that combining only a few of the models could give better results. Table 2 shows the results from combining all models and the best combinations for 3, 4 and 5 models.

Models	**Algorithm**	**Rank01** Number		**Rank02** Sum		**Rank03** Average	
		LAS	UAS	LAS	UAS	LAS	UAS
All	`LocTr`	0.856	0.919	0.857	0.921	0.788	0.843
	`GloTr`	0.869	0.919	0.873	0.921	0.779	0.837
MLT08, MLT09, MLT11	`LocTr`	0.876	0.920	0.878	0.922	0.844	0.885
	`GloTr`	0.873	0.920	0.873	0.922	0.827	0.886
MLT07, MLT08, MLT09, MLT11	`LocTr`	0.872	0.918	0.877	0.922	0.830	0.871
	`GloTr`	0.846	0.898	0.847	0.909	0.828	0.884
MLT07, MLT08, MLT09, MLT11, MST01	`LocTr`	0.875	0.923	0.875	**0.924**	0.828	0.872
	`GloTr`	0.876	0.923	0.877	**0.924**	0.821	0.870

Table 2: Voting using algorithms `LocTr` and `GloTr` for tree construction.

Models	**Algorithm**	**Rank01** Number		**Rank02** Sum		**Rank03** Average	
		LAS	UAS	LAS	UAS	LAS	UAS
All	`LocTr`	0.862	0.929	0.863	0.929	0.819	0.878
	`GloTr`	0.8921	0.9288	0.8922	0.9289	0.8384	0.8716
MLT07, MLT09, MATE01, Turbo02	`LocTr`	0.893	0.933	0.897	0.936	0.883	0.921
	`GloTr`	0.9016	0.9330	**0.9054**	**0.9363**	0.8896	0.9199
MLT01, MLT09, MATE01, Turbo02	`LocTr`	0.89	0.932	0.894	0.936	0.882	0.922
	`GloTr`	0.8997	0.9322	0.9049	0.9359	0.8900	0.9199
MLT01, MLT03, MST01, MST04	`LocTr`	0.858	0.906	0.85	0.897	0.847	0.893
	`GloTr`	0.8634	0.9037	0.8526	0.8968	0.8493	0.8934
MLT09, MST03, MST01, MST04 MST06, Turbo01, Turbo02	`LocTr`	0.869	0.914	0.871	0.916	0.88	0.924
	`GloTr`	0.8711	0.9138	0.8742	0.9161	0.8826	0.9229
MLT02, MLT08, MST04	`LocTr`	0.817	0.86	0.825	0.87	0.849	0.894
	`GloTr`	0.8187	0.86	0.8281	0.8701	0.8508	0.8945

Table 3: Voting using algorithms `LocTr` and `GloTr` for tree construction with the new models.

2.3. Combining Parses by Machine Learning

We conducted two experiments where machine learning techniques were used for ranking the arcs suggested by the different parsing models. This was done with the help of the package `RandomForest`[2] of the system R[3]. The parsebank was once again divided into training and test parts in the same proportion: 90% and 10%.

Our goal was to evaluate each arc as correct or incorrect for a given context. Each arc (`Arc` or `ArcN`) was modeled by three features: relation (`Rel`), distance in words to the parent node (`Dist`) and direction of the parent node (`Dir`): `Left`, the parent node is on the left, and `Right` the parent node is on the right. The features for each word (`Word`) include: the word form (`WF`), lemma (`Lemma`), morphosyntactic tag (`POS`).

For the first experiments we have used all the arcs for a given word as a context and the trigrams around the word and the parent word.

The tuples generated from the training part of the treebank were used to train the `RandomForest` in regression mode, then the model was applied to test set to rank each arc. These weights were used by the algorithms `LocTr` and `GloTr`. The results are presented in Table 4.

In the second experiment the candidate arc was evaluated within the context of one alternative arc for a word.

All sub-vectors are the same except that as a context we use one arc `ArcAlt` and the grammatical features for its parent node: `AltParentPOS`. Some of the arcs could receive more than one weight because each arc could have more than one alternative. We used these weights to define three models: (1) prefer as a weight the maximum of the weights; (2) prefer as a weight the minimum of the weights; and (3) assign as a weight the multiplication of the weights. The results from this experiment are presented in Table 5.

These results show that machine learning can be applied successfully for improving the voting models. The results also show the difference between the two combining algorithms (`LocTr` and `GloTr`).

3. System for Experiments with Dependency Parsers

The results reported above show that there does not exist yet one good approach to parsing combinations. The performance of an ensemble model depends on the following parameters: the initial parsing models; the algorithms for selection of arcs (in our case `LocTr` and `GloTr`); the selections of ranks for the alternative arcs - via voting or via machine learning; and the number of the combined models. The search for optimal solution becomes unfeasible with so many parameters. For example, for the fourteen models only the possible combinations between them amount to 16369. This motivated us to implement a system which provides facilities for the following tasks:

- **Treebank declaration.** A treebank as a set of one or more files in CoNLL 2006 Shared Task format is

[2]http://cran.r-project.org/web/packages/randomForest/randomForest.pdf

[3]http://www.r-project.org/

Model	Algorithm	LAS	UAS
MLearn14	`LocTr`	0.859	0.92
MLearn14	`GloTr`	0.896	**0.925**

Table 4: Results for the first experiment with `RandomForest`.

Model	Algorithm	MinRank		MaxRank		MultRank	
		LAS	UAS	LAS	UAS	LAS	UAS
MLearn01	`LocTr`	0.844	0.903	0.843	0.902	0.828	0.887
MLearn01	`GloTr`	0.899	**0.929**	0.897	0.926	0.886	0.915

Table 5: Results for the second experiment with `RandomForest`.

provided. Additional columns could be added after the original columns. The treebank is named within the system. A partitioning of the data into N sets of approximately equal length is performed (if desired) in one of three possible ways: (1) sentences from the original data set are assigned to N data sets in consecutive order (sentence$_1$ is assigned to set$_1$, sentence$_2$ to set$_2$, etc.), (2) sentences are assigned randomly to the N data sets, or (3) the data is split into data sets by preserving the original order of the sentences (sentence$_1$ to sentence$_m$ are assigned to set$_1$, sentence$_{m+1}$ to sentence$_{m+n}$ are assigned to set$_2$, etc).

- **Training of initial parsing models.** For the moment we provide access to the four parsers. The user has the possibility to provide the name of the parser, the features to be used and the specific treebank to be used for training and testing if there is more than one treebank. The model is trained on each of the training part and applied on the test part and evaluated. The result is stored in the parsebank. Thus, at the end we have the average value of the scores (LAS and UAS) and the parsebank itself.

- **Ranking of arcs.** This part contains a simple rule language where for a given arc it determines its rank. The rule is described in the following format: `If` $context(Arc)$ `then` $r(Arc) = f(Arc)$, where $context(Arc)$ is a simple conjunction of elementary predicates like $pos(Arc) = string$; $f(Arc)$ returns a real number. The function $f(Arc)$ is an expression over elementary functions and could contain calls to external programs like machine learning systems. The rules are working in the following way: if $context(Arc)$ is true, then the value returned by $f(Arc)$ is stored as a ranking for this arc. The rules can be grouped together in named lists. The rules in the list are applied in sequence and the value from the first applied rule is stored as a rank for the arc. In this way we could rank the whole treebank for each list. The ranks are used for combination of parsers. Training of machine learning systems for ranking is not incorporated in the system yet.

- **Combination of models.** To perform a combination experiment the user needs to specify the treebank, the list of models to be combined, the ranking model, the algorithm for selection of arcs. On the basis of these parameters the system enumerates all the possible combinations, evaluates each of them, and generate the results for error analysis. In this way the user could find the best combination for his/her task.

Having this functionality we are able to reproduce all the experiments reported above. In addition, when more data becomes available we will be able to perform similar experiments for it in two ways: (1) by using the same parameters with the new data; (2) by combining models trained on different datasets.

4. New Experiments

In order to extend the coverage of the experiments to include other parsing systems and to try to achieve a better overall result we have trained several new parsing models and added them to the previous ones.
Four new MST parsing models were trained in addition to the two available ones. The new models differ from the existing ones in the choice of number of iterations and order of features: MST03 (*order*:1, *iters*:10), MST04 (*order*:1, *iters*:20), MST05 (*order*:2, *iters*:10), the best performing MST model in our experiments, and MST06 (*order*:2, *iters*:20).
Two models were trained using the TurboParser, which differ mainly in the complexity of features used during training. The default training option *model_type*=standard was chosen for model Turbo01. It includes the options: enable arc-factored parts, consecutive sibling parts, and grandparent parts. Turbo02 has been trained with a more complex setting *model_type*=full. It includes the options of Turbo01 and in addition enables arbitrary sibling parts, non-projectivity parts, grand-sibling third-order parts, and tri-sibling third-order parts.
In addition, a model was trained with the default settings of MATEParser (*training-iterations*=10, *threshold*=0.3).
The results from the 10-fold cross validation for the new models are presented in Table 1. Most of the new models outperform the old ones in terms of LAS and UAS scores. The best performing model is MATE01, with 92.9% UAS and 89.1% LAS, followed by TURBO02 with 92.7% UAS and 88.8% LAS.
The possible combinations of all 21 parser models are more than 2 million. Here we will list some of the results we obtained. They represent the best results, the worst results and some average results. Also the results for the combination of all models and for one of the minimal combinations. The conclusion is that the size of the combination does not

predict the best result of the ensemble model although the combinations with size around the middle (12-13 in our case) tend to have better results. The initial expectations on the basis of the previous experiments were that Rank2 (sum) would always be better that Rank1 (number), but the combination MLT01, MLT03, MST01, MST04 constitutes a contra example. The best results are LAS 0.9054 and UAS 0.9363 (see Table 3).

We are still working on extending the system to incorporate the machine learning module which we have previously experimented with. Therefore currently we only present results with the other voting techniques.

5. Conclusion and Future Work

In this paper we presented several approaches for combining parses produced by several parsing models. These approaches include three types of voting and two machine learning approaches. Also, for the construction of the combined trees we used two different algorithms – one performing local optimization and one performing global optimization.

On the basis of the experience we gained during our experiments we implemented a first version of a system to support the user in the design and implementation of such combination setups.

In future, we will extend the system with direct interaction with machine learning frameworks like R or Weka. Also, we are planing to extend the system for creation of web services for selected combination via the Storm framework. It is a distributed realtime computation system[4] for actual implementation of running selected models in parallel.

6. Acknowledgements

This research has received partial funding from the EC's FP7 (FP7/2007-2013) under grant agreement number 610516: "QTLeap: Quality Translation by Deep Language Engineering Approaches".

7. References

Attardi, Giuseppe and Dell'Orletta, Felice. (2009). Reverse revision and linear tree combination for dependency parsing. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 261–264, Boulder, Colorado, June. Association for Computational Linguistics.

Bohnet, Bernd. (2010). Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 89–97, Stroudsburg, PA, USA. Association for Computational Linguistics.

Martins, André F. T., Smith, Noah A., Xing, Eric P., Aguiar, Pedro M. Q., and Figueiredo, Mário A. T. (2010). Turbo parsers: Dependency parsing by approximate variational inference. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 34–44, Stroudsburg, PA, USA. Association for Computational Linguistics.

Martins, Andre, Almeida, Miguel, and Smith, Noah A. (2013). Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria, August. Association for Computational Linguistics.

McDonald, Ryan and Nivre, Joakim. (2007). Characterizing the errors of data-driven dependency parsing models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 122–131.

McDonald, Ryan. (2006). *Discriminative Training and Spanning Tree Algorithms for Dependency Parsing*. Ph.D. thesis.

Nivre, Joakim and McDonald, Ryan. (2008). Integrating graph-based and transition-based dependency parsers. In *Proceedings of ACL-08: HLT*, pages 950–958, Columbus, Ohio, June. Association for Computational Linguistics.

Nivre, Joakim, Hall, Johan, and Nilsson, Jens. (2006). Maltparser: a data-driven parser-generator for dependency parsing. In *Proceedings of LREC-2006*.

Simov, Kiril, Ivanova, Ginka, Mateva, Maria, and Osenova, Petya. (2013). Integration of dependency parsers for bulgarian. In *The Twelfth Workshop on Treebanks and Linguistic Theories*, pages 145–156, Sofia, Bulgaria.

[4] storm-project.net