# Integration of Dependency Parsers for Bulgarian

Kiril Simov[†], Ginka Ivanova[†], Maria Mateva[‡], Petya Osenova[†]

[†]Linguistic Modelling Laboratory, IICT-BAS, Sofia, Bulgaria
[‡]Sofia University "St. Kl. Ohridski", Sofia, Bulgaria
kivs@bultreebank, givanova@bultreebank.org,
mateva.maria@gmail.com, petya@bultreebank.org

**Abstract**

Various parsing models — based on different parsing algorithms or different sets of features — produce errors in different places in the dependency trees — [6]. This observation initiated a wide range of research devoted to combining the outputs of various parsing models with the hope to achieve a better parsing result. In this paper we present our work on combining results from 14 parsing models for Bulgarian. First, we construct an extension of the Bulgarian treebank with the parses from each of the models. Then, we evaluate different combining approaches including three voting models and two machine learning approaches. Each of the weighting mechanisms is used by two different tree construction algorithms to find an optimal solution.

## 1 Introduction

The combination of several dependency parsers is motivated by the need of having an accurate parser even at the price of slower performance. Considering the outcomes of [10], we have performed several experiments to assemble the results of several parsing models using different types of voting and also exploiting machine learning techniques. We show that, contrary to the conclusion of [10], machine learning techniques can outperform voting techniques.

First, we tried fourteen different parsing models using two of the most popular parsing systems: MaltParser (see [7]) and MSTParser (see [5]). They were trained using different parsing algorithms and different sets of features. Our goal was to minimize the number of nodes for which there was no parsing model to infer the correct arc. For each of the fourteen models at our disposal, we constructed a version of the original treebank where we stored the arcs suggested by the parsing model. Thus, at the end we had fourteen additional treebanks — one per model. We used them for performing the voting experiments and for training a machine learning component to select the most appropriate arc for each node. We experimented with three voting approaches: (1) majority of the parsers that selected a

given arc; (2) maximum of the sum of parsers' accuracy values of the parsers that selected a given arc; and (3) average of the accuracy of the parsers that selected a given arc. The best result was achieved by the second approach. For machine learning approaches we performed two experiments: (1) selecting the correct arc among all arcs suggested by the fourteen models; and (2) ranking the arcs by comparing pairs of arcs. Here also the second approach was the best. In both cases we used two algorithms for constructing dependency trees from the trees suggested by the fourteen models: the incremental algorithm of [1] and the global algorithm for non-projective parsing of [5]. The most evident difference between the two models is shown by the second machine learning approach.

The structure of the paper is as follows: first, we discuss the related work; in the next section, the linguistic phenomena, encoded in the treebank, are presented; in Section 4 we introduce the parsing models that we have trained as well as the construction of an extended version of BulTreeBank to be used in the experiments; Section 5 describes the models that we exploit for the voting combination of parses; Section 6 presents the machine learning approaches relevant to the task; the last section concludes the paper.

## 2 Related work

Our work is inspired by the analysis of the two most influential dependency parsing models: transition-based and graph-based frameworks — [6]. They showed that these two frameworks made different errors on the same training and test datasets. The authors conclude the paper by proposing three approaches of using the advantages of both frameworks: (1) Ensemble systems — weighted combinations of both systems output; (2) Hybrid systems — design of a single system integrating the strengths of each framework; and (3) Novel approaches — based on combination of new training and inference methods. In their further work (see [8]) the authors present a hybrid system that combines the two models. In our work we consider option one — ensemble systems, since it is easier to implement and does not require further knowledge of the models and the design and implementation details of the parsing algorithms.

Another work devoted to ensemble systems is [10]. They tested different approaches to parser integration — different types of voting and meta-classification. The voting determines the correct arcs on the basis of majority of parsers that select given arcs. The weighted voting is using the accuracy of each parser in order to choose between them for each arc. Authors' conclusion is that meta-classification does not help for improving the result in comparison to voting. The authors divided the dependencies into two ways: majority dependencies and minority dependencies. Their conclusion is that meta-classification cannot provide a better selection on minority dependencies, and in this way it is comparable to voting. In our work we show that depending on the feature selections for the meta-classification we might outperform the voting approach. The results reported in [10] do not use a

special algorithm for the selection of dependencies. They do not require the result to be a tree. In our work we use two algorithms to ensure the construction of trees. We show that the improvement also depends on the algorithm for constructing the complete tree.

As mentioned above, we use two algorithms for construction of the dependency tree. The first algorithm is reported in [1]. 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 decides for the best arc locally. We will denote this algorithm below as `LocTr` . The second algorithm is Chu-Liu-Edmonds algorithm for maximal spanning tree implemented in the MSTParser of [5]. This algorithm starts with a complete dependency graph including all possible dependency arcs. Then the algorithm selects the maximal spanning tree on the basis of the weights assigned to the potential arcs. In our work we use weights for the arcs assigned in two ways: (1) by voting; and (2) by metaclassifiers. The arcs that are not proposed by any of the parsers are deleted. This algorithm is global with respect to the selection of arcs. We will denote this algorithm below as `GloTr` .

## 3 The Linguistic Annotation of the Bulgarian Treebank (BulTreeBank)

BulTreeBank (see [9]) is a treebank that provides rich linguistic information going beyond the syntactic information. It has part-of-speech (POS) and full grammatical tags; lemmas; syntactic relations, based on Head-driven Phrase Structure Grammar (HPSG); named entities and co-references within a sentence.

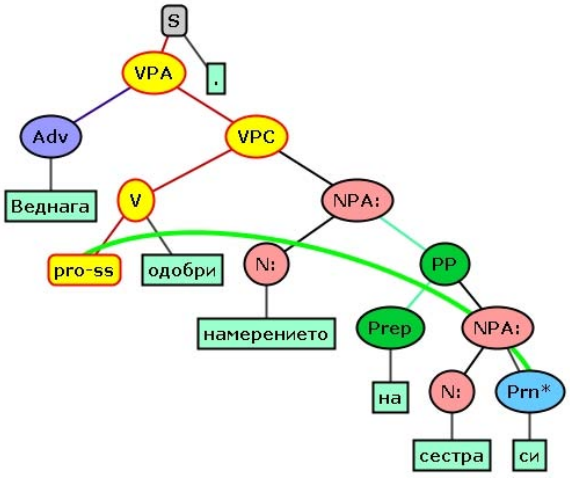


Figure 1: HPSG-based tree for the sentence “Веднага одобри намерението на сестра си” (‘Immediately approved intention of sister his’, He approved his sister’s intention immediately).

Figure 1 presents the HPSG-based tree of a Bulgarian sentence. Here two things are worth mentioning: the presence of dependency information in the HPSG-based version (NPA means a nominal phrase of type head-adjunct) and the presence of a co-reference link between the unexpressed subject and the reflexive possessive pronoun. In the HPSG-based version of the treebank the unexpressed subject is represented explicitly only in the cases when it participates in a coreference chain as in this example. It is considered a property of the verb node, not a part of the constituent structure.

In Figure 2 the same sentence can be observed in a dependency format. The head-adjunct relation within the NPA nodes in the HPSG-based tree is projected into a head-modifier relation in the dependency tree. The arc labels are represented as ovals between word nodes. The coreference is represented as a secondary edge between the corresponding word nodes.

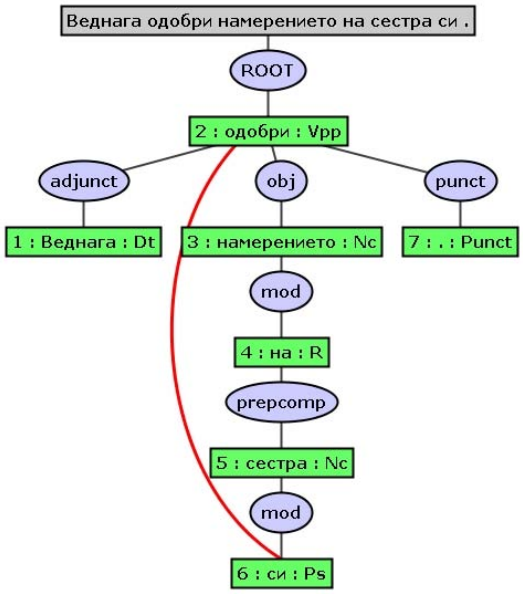


Figure 2: Dependency tree representation of the sentence from Figure 1.

In Table 1 the dependency tagset related to the Dependency part of the BulTreeBank is presented. In addition to the dependency tags we also have morphosyntactic tags attached to each word. For each lexical node the lemma is assigned. The number under the name of each relation indicates how many times the relation appears in the dependency version of BulTreeBank. Many parsers have been trained on data from BulTreeBank during the CoNLL 2006 shared task and after[1]. BulTreeBank appeared to be one of the best parsed treebanks — [3].

[1] See papers in the proceedings at http://aclweb.org/anthology/W/W06/#2900

| Relation | Occurrences | Description |
|---|---|---|
| adjunct | 12009 | Adjunct (optional verbal argument) |
| clitic | 2263 | Short forms of the possessive pronouns |
| comp | 18043 | Complement (argument of non-finite verbs, copula, auxiliaries) |
| conj | 6342 | Conjunction in coordination |
| conjarg | 7005 | Argument (second, third, ...) of coordination |
| indobj | 4232 | Indirect Object |
| marked | 2650 | Marked (clause, introduced by a subordinator) |
| mod | 42706 | Modifier (dependants which modify nouns, adjectives, adverbs, ...) |
| obj | 7248 | Object (direct argument of a non-auxiliary verbal head) |
| pragadjunct | 1612 | Pragmatic adjunct |
| punct | 28134 | Punctuation |
| subj | 14064 | Subject |
| xadjunct | 1826 | Clausal adjunct |
| xcomp | 4651 | Clausal complement |
| xmod | 2219 | Clausal modifier |
| xprepcomp | 168 | Clausal complement of preposition |
| xsubj | 504 | Clausal subject |

Table 1: Dependency relations and numbers of their occurrences in BulTreeBank.

# 4 Parser Models and Extension of the Treebank

For our experiments we have trained 12 models of MaltParser [7] using different parsing algorithms and different features, and MSTParser of [5] with two different sets of features. The models are: MaltParser and MSTParser.

Initially, we conducted a series of experiments with both parsers and BulTreeBank using 10-fold cross-validation. The original treebank on which the models were trained contains 11988 sentences. They were randomly divided in ten training and test sets in such a way that each sentence appeared in one test set. Each training set contains 10790 sentences and each test set — 1198. The number of tokens is 183649. For each of the models we performed ten training sessions. The trained model was applied on the corresponding test set for evaluation, but also the predicted parses for each sentence were stored. In this way, for each model we constructed a treebank of the suggested parses. Thus, at the end we had the original treebank plus fourteen treebanks of parses. We used all these treebanks for performing different combinations of parses from the different models.

For MSTParser we preselected the two best performing parsers on average with major difference in the scope of features (order). The first parser used features over a single edge, while the second one used features over pairs of adjacent edges. The rest of the parameters were chosen for the parser's optimal labeled and unlabeled accuracy, on average, of the experiments on BulTreeBank: MST01 model used the following features: loss-type: *punc*; second-order: *false*; training-iterations: *15*;

trainingk: *1*; decode-type: *non-proj*; create-forest: *true*; MST02 model used the following features: loss-type: *punc*; second-order: *true*; training-iterations: *15*; trainingk: *1*; decode-type: *non-proj*; create-forest: *true*.

For MaltParser we applied three Malt algorithms: Covington non-projective, Stack Eager and Stack Lazy. The predefined flow chart is set to learn. The actual configuration type of MaltParser is “singlemalt". The input data type is CoNLL-X shared task data format — see [3]. 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 (otherwise all tokens in Left must be analyzed before treating the next token). 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 late as possible respectively for the second one). The execution of algorithms with the LIBLINEAR method is faster than algorithms with the LIBSVM method and the results are better. On the basis of these six models we constructed additional six ones by extending the set of node features including more detailed description of grammatical features and lemma.

The results in terms of Labeled Attachment Scores (LAS) and Unlabeled Attachment Scores (UAS) for the fourteen models are given in Table 2.

| | **MLT01** | **MLT02** | **MLT03** | **MLT04** | **MLT05** | **MLT06** | **MLT07** |
|---|---|---|---|---|---|---|---|
| **LAS** | 0.842 | 0.788 | 0.843 | 0.809 | 0.825 | 0.820 | 0,863 |
| **UAS** | 0.886 | 0.835 | 0.887 | 0.860 | 0.869 | 0.869 | 0,900 |
| | **MLT08** | **MLT09** | **MLT10** | **MLT11** | **MLT12** | **MST01** | **MST02** |
| **LAS** | 0.810 | 0.872 | 0.844 | 0.851 | 0.847 | 0.852 | 0.828 |
| **UAS** | 0.849 | 0.909 | 0.886 | 0.887 | 0.888 | 0.898 | 0.872 |

Table 2: LAS and UAS results for the fourteen models.

The models do not predict the correct unlabeled arc for 1.99% of the words in the treebank. For the labeled arcs the percentage is 3.49%. Thus, the upper bound for the UAS measure is 98.01% and for the LAS measure is 96.51%. The results reported within this paper are still far from the upper bounds for both measures.

## 5 Combining Parses by Voting

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

of the UAS measures of the parsers that predicted the arcs. Let us consider an artificial example for arcs suggested by different models. They are given in Table 3

| | **MLT01** | **MLT02** | **MLT03** | **MLT04** | **MLT05** | **MLT06** | **MLT07** |
|---|---|---|---|---|---|---|---|
| **Node** | Arc01 | Arc02 | Arc01 | Arc01 | Arc03 | Arc02 | Arc02 |
| **UAS** | 0.835 | 0.887 | 0.860 | 0.869 | 0.869 | 0.886 | 0,899 |
| | **MLT08** | **MLT09** | **MLT10** | **MLT11** | **MLT12** | **MST01** | **MST02** |
| **Node** | Arc01 | Arc04 | Arc03 | Arc03 | Arc03 | Arc02 | Arc02 |
| **UAS** | 0.848 | 0.908 | 0.886 | 0.887 | 0.888 | 0.898 | 0.872 |

Table 3: Different arcs for a node in a tree. For orientation we also give UAS values for each model.

On the basis of these arc suggestions and UAS values for the different models we can calculate the three ranks. These ranks for voting are given in Table 4. As it can be seen, the different ranking models define a different ordering on the arcs. Different values also play significant role when the arcs are combined by the algorithms to form a dependency tree.

| **Arcs** | **UASes** | **Rank01** | **Rank02** | **Rank03** |
|---|---|---|---|---|
| **Arc01** | 0.835, 0.860, 0.869, 0,848 | 4 | 3.412 | 0.853 |
| **Arc02** | 0.887, 0.886, 0.899, 0.898, 0.872 | 5 | 4.442 | 0.888 |
| **Arc03** | 0.869, 0.886, 0.887, 0.888 | 4 | 3.530 | 0.883 |
| **Arc04** | 0.908 | 1 | 0.908 | 0.908 |

Table 4: Different ranking for voting.

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. Then following the suggestions of [10] we performed combinations by using the results from different models. Although our results are not drastically different, they show that combining only a few of the models could give better results than combining all the models. It is interesting that combining several parser models with best scores does not give the best result, but it is relevant to include at least one parser model with low score. Table 5 shows the results from combining all fourteen models and then the best combinations for 3, 4 and 5 models.

The results clearly show that ranking with average accuracy influences performance very negatively. The experiments demonstrate slight preference for the rank, based on the sum of accuracies for the various models. The good news is that several combinations achieved substantial improvement over the accuracy of the individual models.

# 6 Combining Parses by Machine Learning

Although [10] claimed that they could not implement a combination model that involves machine learning methods for selection of arcs and which improves sub-

| Models | Algor. | 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 | 0.883 | 0.919 | 0.885 | 0.921 | 0.804 | 0.836 |
| MLT08, | LocTr | 0.876 | 0.920 | 0.878 | 0.922 | 0.844 | 0.885 |
| MLT09, MLT11 | GloTr | 0.869 | 0.903 | 0.875 | 0.909 | 0.858 | 0.893 |
| MLT07, MLT08, | LocTr | 0.872 | 0.918 | 0.877 | 0.922 | 0.830 | 0.871 |
| MLT09, MLT11 | GloTr | 0.873 | 0.907 | 0.882 | 0.916 | 0.852 | 0.885 |
| MLT07, MLT08, | LocTr | 0.875 | 0.923 | 0.875 | **0.924** | 0.828 | 0.872 |
| MLT09, MLT11, MST01 | GloTr | 0.886 | 0.918 | 0.888 | **0.921** | 0.850 | 0.881 |
| MLT07, MLT09, | LocTr | 0.874 | 0.923 | 0.875 | **0.924** | 0.828 | 0.872 |
| MLT12, MST01, MST02 | GloTr | 0.888 | 0.923 | 0.891 | **0.925** | 0.840 | 0.872 |
| MLT01, MLT07, | LocTr | 0.872 | **0.925** | 0.873 | **0.925** | 0.825 | 0.872 |
| MLT09, MLT11, | GloTr | 0.891 | **0.925** | 0.892 | **0.925** | 0.838 | 0.869 |
| MLT12, MST01, MST02 | | | | | | | |

Table 5: Voting using algorithms LocTr and GloTr for tree construction.

stantially over the voting approaches, we performed several experiments in order to test different feature sets. Our goal was to test this claim on our data. Our intuition was that independently from the task: direct dependency parsing or combining existing parses, we need machine learning mechanisms that are able to learn also hard-to-predict dependencies.

Here we conducted two experiments using the extended treebank for training and testing of machine learning techniques for ranking suggested arcs. Again, as in the case of voting, we tested both algorithms for the construction of the dependency trees.

The experiments were conducted using the package RandomForest [2] of the system R[3]. Random Forest (see [2]) constructs randomly several decision trees over the data and then assembles their predictions. We chose this package because of the following characteristics of RandomForest : (1) it supports classification and regression methods; and (2) it does not overfit. The data for RandomForest is prepared as a vector of feature values where one of the features determines what the package predicts.

The treebank was divided again into training and test parts in the same proportion: 90% for training and 10% for test. The division was made orthogonal to the division used for the creation of the fourteen treebanks used to train the models. In this way, we reduced the bias from the training of the parsing models.

Our goal was to evaluate each arc suggested by a parsing model as a correct arc for a given context or as an incorrect arc for the context. Each arc was modeled by three features: relation (Rel ), distance to the parent node measured as a number

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

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

of words in the sentence between the two nodes (Dist ) and direction of the parent node (Dir ). Direction could be Left , which means the parent node is on the left side of the node, and Right — the parent node is on the right side of the node. In cases when these features are not relevant, we simply use Arc or ArcN for the arc features. The features for each word node include: the word form (WF ), lemma (Lemma ), morphosyntactic tag (POS ). Again, when the features are not important in isolation, we denote them as Word with optional modification.

For the first experiments we have evaluate the appropriateness of one arc between dependant node Word and a head (or parent) node. In this experiment we have used all the arcs for a given node as a context together with the trigrams around the node, and the parent node. The node has features Word , the parent ParentWord . A representation of this data as a value vector for RandomForest is presented in Table 6.

| **Feature** | **Value** |
|---|---|
| Word | the current node |
| WordBefore | the word before the current node |
| WordAfter | the word after the current node |
| Arc01 , . . . , ARC14 | arcs suggested by each of the fourteen models |
| ParentWord | the parent word |
| PWordBefore | the word before the parent word |
| PWordAfter | the word after the parent word |
| EvaluationArc | one of the arcs suggested by one of the models for the node |
| CorrectIncorrect | true or false depending on whether the EvaluationArc arc is the correct one for the node |

Table 6: Feature vector used with RandomForest for the first experiment.

All tuples for all arcs in the training and the test parts of the extended treebank were generated. The tuples generated from the training part of the treebank were used to train the RandomForest , then the model was applied on test set to classify each of the tuples as correct or incorrect.

The first experiments were done using the classification method of RandomForest . The results were very disappointing. Table 7 presents the results for this classification method. These are the results for the tuples, not the dependencies. The results obtained for tree construction were very bad. Our explanation of the results is the fact that we had only two classes. Thus the classifier was sensitive to values close to 50%. Thus, dependencies that were hard to predict correctly very often received weights near to this threshold, and small fluctuations above or under it caused wrong classification. The first attempt to avoid this problem was to divide the training and the test sets according to the POS information for the wordform. Unfortunately, this did not improve the situation substantially. The second attempt we performed was to switch from this classification method to regression.

In order to use the regression method, we first encoded all the symbolic data as numerical data. The results from regression are between 8 and 17 mean of

| | Correct | Incorrect |
|---|---|---|
| **True** | 65.2% | 73.4% |
| **False** | 34.8% | 26.6% |

Table 7: Results from the classification of tuples.

| Model | Algorithm | LAS | UAS |
|---|---|---|---|
| **MLearn14** | `LocTr` | 0.859 | 0.920 |
| **MLearn14** | `GloTr` | 0.896 | **0.925** |

Table 8: Results for the algorithms `LocTr` and `GloTr` when the correctness of an arc is evaluated with respect to all 14 models.

squared residuals which are much better than the results for classification. Again, the training and test sets were divided according to the POS tag of Word. Each element of the test set received a weight returned from the regression model for the corresponding part of speech. These weights were used by the algorithms `LocTr` and `GloTr` to construct the dependency trees. The results from the two algorithms are presented in Table 8.

These results confirm the conclusions of [10] that machine learning hardly improves over the voting methods. Our explanation of the results is that the machine learning component is learning to do voting selecting the best suggestion of the fourteen parsing models. Thus, using all the fourteen arcs as a context is not a good idea. In order to avoid such cases, we designed a second experiment in which the candidate arc was evaluated in the context of one alternative arc for a given node. In this way, we were trying to implement a model which learns which arc from two candidate arcs is better for a node. Thus, for the second experiment we have evaluated the appropriateness of one arc between a dependant node `Word` and a head (or parent) node in context of one alternative arc for the same node with the trigrams around the node, and the parent node. The node has features `Word` , the parent has features `ParentWord` . For the alternative arc we also have used as a context the grammatical features of its parent node — `AltParentPOS` . A representation of this data as a value vector for `RandomForest` is presented in Table 9.

The training and test sets were divided further according to the POS tag. The mean of squared residuals for the different POS are between 4 and 12. Each element of the test set received a weight returned from the regression model. But this time some of the arcs in the treebank could receive more than one weight because each arc could have more than one alternative arc. We used these weights to define three models: (1) prefer as a weight for the arc the maximum of the weights for the tuples for the arc; (2) prefer as a weight for the arc the minimum of the weights for the tuples for the arc; and (3) assign as a weight for the arc the multiplication of the weights for the tuples for the arc. For all three models we ran the algorithms `LocTr` and `GloTr` to construct the dependency trees. The results from this experiment are presented in Table 10. They show that machine learning could improve substantially the voting models. In our case the improvement is near half percent. These

| Feature | Value |
|---|---|
| Word | the current node |
| WordBefore | the word before the current node |
| WordAfter | the word after the current node |
| ArcAlt | alternative arc |
| AltParentPOS | grammatical features for alternative parent node |
| ParentWord | the parent word |
| PWordBefore | the word before the parent word |
| PWordAfter | the word after the parent word |
| EvaluationArc | the arc which we compare with the alternative arc ArcAlt |
| CorrectIncorrect | true or false depending on whether the EvaluationArc arc is the correct one for the node |

Table 9: Feature vector used for the second experiment with RandomForest.

results show better the difference between the two combining algorithms (LocTr and GloTr ) — the results from the global optimization are better.

| 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 10: Results for the algorithms LocTr and GloTr when the correctness of an arc is evaluated with respect to one alternative arc.

# 7 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. The better ranking of the suggested arcs also demonstrates the better performance of the global algorithm.

Although we achieved some substantial improvement over the individual models (near 3% improvement), the results are far from the potential maximum UAS measure of 98.01% and for LAS measure — of 96.51%. In spite of that, our best result is near to the state-of-the-art performance for Bulgarian — 93.5 % — see [4]. Thus, our first goal to have access to a better parser for Bulgarian is achieved.

In future, we would like to combine better individual parsing models. In these experiments we executed the very basic parser configurations without extensive optimization. We hope that starting from better models, the combined result will be also better. Also, we would like to include more sentences in the original treebank in order to have better coverage of the problematic cases. Last, but not least, it will be interesting to see the impact of adding semantic features to the model.

## Acknowledgements

This research has received partial funding from the EC's FP7 (FP7/2007-2013) under grant agreement number 611760: "EUropean and National CASE Law and Legislation Linked in Open Data Stack" and EC's FP7 (FP7/2007-2013) under grant agreement number 610516: "QTLeap: Quality Translation by Deep Language Engineering Approaches". We would like to thank Laura Tolo̧si for helping us with R system.

## References

[1] Giuseppe Attardi and Felice Dell'Orletta. 2009. Reverse revision and linear tree combination for dependency parsing. In *Proceedings of NAACL HLT 2009, Companion Volume: Short Papers*, pages 261–264.

[2] Leo Breiman. Random forests. 2001. *Machine Learning*, 45(1):5–32, 2001.

[3] Sabine Buchholz and Erwin Marsi. 2006. Conll-x shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164.

[4] Andre Martins, Noah Smith, Mario Figueiredo, and Pedro Aguiar. 2011. Dual decomposition with many overlapping components. In *Proceedings of EMNLP 2011*, pages 238–249.

[5] Ryan McDonald. 2006. *Discriminative Training and Spanning Tree Algorithms for Dependency Parsing*. PhD thesis.

[6] Ryan McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of EMNLP-CoNLL 2007*, pages 122–131.

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

[8] Joakim Nivre and Ryan McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proceedings of ACL-08: HLT*, pages 950–958.

[9] Kiril Simov, Petya Osenova, Alexander Simov, and Milen Kouylekov. 2004. Design and implementation of the Bulgarian HPSG-based treebank. *Research on Language & Computation*, 2(4):495–522, 2004.

[10] Mihai Surdeanu and Christopher D. Manning. 2010. Ensemble models for dependency parsing: Cheap and good? In *Proceedings of NAACL-2010*, pages 649–652.