Dependency Parsing of Indian Languages with DeSR

Giuseppe Attardi
Dipartimento di Informatica
Università di Pisa
attardi@di.unipi.it

Stefano Dei Rossi
Dipartimento di Informatica
Università di Pisa
deirossi@di.unipi.it

Maria Simi
Dipartimento di Informatica
Università di Pisa
simi@di.unipi.it

Abstract

DeSR is a statistical transition-based dependency parser which learns from annotated corpora which actions to perform for building parse trees while scanning a sentence. We describe the experiments performed for the ICON 2010 Tools Contest on Indian Dependency Parsing. DesR was configured to exploit specific features from the Indian treebanks. The submitted run used a stacked combination of four configurations of the DeSR parser and achieved the best unlabeled accuracy scores in all languages. The contribution to the result of various choices is analyzed.

1 Introduction

Dependency-based syntactic parsing is the task of uncovering the dependency tree of a sentence, which consists of labeled links representing dependency relationships between words. The task can be cast into a series of classification problems, picking a pair of tokens and deciding whether to create a dependency link between them.

A statistical *transition-based parser* uses training data to learn which actions to perform for building a dependency graph while scanning a sentence. The state of the art in the field is well represented by the results in the CoNLL Shared tasks.

The DeSR parser was among the most accurate transition-based parsers at the CoNLL 2008 Shared Task, Surdeanu, et al. (2008), and presents distinctive features like a deterministic incremental algorithm of linear complexity and the ability to handle non-projective dependencies, which often arise in languages like Hindi.

The best accuracy though was obtained using an SVM classifier, or rather multiple binary SVM classifiers, which are quite expensive to train and require a lot of memory. More recently we revised the DeSR architecture incorporating the following features:

- a Multilayer Perceptron (MLP) classifier to improve speed and reduce the memory footprint, while maintaining a similar level of accuracy
- a beam search strategy to achieve better accuracy

The MLP classifier was used successfully in the Evalita 2009 contest on dependency parsing for Italian, Attardi, et al. (2009), in combination with the SVM classifier. For the ICON competition we explored the combination of different versions of the parser, all of them using an MLP classifier.

2 Parsing Indian Languages

The ICON 2010 Tools Contest on Indian Language Dependency Parsing, Husain et al. (2010), provided data for three languages: Hindi, Bangla and Telugu. The collections are annotated according to the Shakti Standard Format described in Bharati, Sangal and Sharma (2007).

The annotations include POS tagging, morphology and chunking. A chunk is a set of adjacent words which are in a dependency relation with each other, and are connected to the rest of the words by a single incoming arc to the head of the chunk. The Hindi treebank has complete dependency trees, while Telugu and Bangla treebanks only provide chunk heads and links between them.

Chunk information provides useful indications for the parser, as shown in our previous experiments with other languages, reported in Attardi and Dell'Orletta (2008). Hence we devised a specific feature model that exploits chunk information. The chunk information for Hindi is provided in the FEATS field of the CoNLL-X format, described in Buchholz, et al. (2006), by means of the tags ChunkId and ChunkType. The ChunkId is present only for tokens which are chunk heads and includes the type of the chunk and a chunk sequence number (e.g. NP2), while the ChunkType denotes whether the token is a head or a child of the chunk and also includes the ChunkId (e.g. head:NP2, child:NP2).

In the CoNLL-X versions of the Telugu and Bangla treebanks, only chunk heads are present and the sub-tree representing the chunk itself has been removed from the corpus. So only the ChunkId information is present and it can be found in the CPOSTAG field.

In all three treebanks, the field FEATS contains redundant information, including for instance the values of the token's lemma and form, which are present in separate fields. Hence we preprocessed the Hindi data files removing the redundant information and moving the chunk information from the FEATS field into two separate fields (named CHUNKID and CHUNKTYPE). We also dropped the chunks sequence number after some experiments showed that this provided a small accuracy improvement.

For Bangla and Telugu we used both the ChunkIds with sequence number (CHUNKID field) and without (CPOSTAG field). Instead of the missing information about the ChunkType we used the coreference information that was moved from the FEATS field into a new field called COREF.

DeSR, Attardi (2006) and Attardi et al. (2007), can be configured to use a specific feature model.

The following table shows a baseline feature model that we used as a starting point for our experiments:

Field	*Tokens*
FORM	-1 0 1
LEMMA	-1 0 1
POSTAG	-2 -1 0 1 2 3 leftChild(-1) leftChild(0)
CPOSTAG	-1 0
FEATS	-1 0
DEPREL	-1 0 leftChild(-1) leftChild(0) rightChild(-1)

Table **1**. Basic feature model

In the above notation, the first column indicates which feature is used and the second column indicates from which tokens the feature is extracted. Tokens are numbered relative to the current parser position in the input queue, indicated as 0; positive numbers represent successive tokens in the input queue, while negative numbers represent tokens in the parser stack. Tokens are reached from those numbered following a certain path, which is composed by operators for moving along the constructed parse tree (parent, leftChild, rightChild) or along the linear sentence order (prev, next). For overall details on the operation of the parser and its use of features during learning and parsing, we refer to Attardi (2006), while for the details of the Multilayer Perceptron Classifier and the stacking approach, we refer to Attardi, et al. (2009).

The features were chosen after a series of experiments on the development set in an effort to improve on the baseline results. After an extensive feature selection process we achieved the best results with the following feature models for the three Indian languages:

Field	*Tokens*
FORM	-1 0 1 prev(-1) parent(-1) leftChild(0)
LEMMA	-1 0 1
POSTAG	-1 0 1 2 3 parent(-1) prev(-1) next(-1) rightChild(-1) leftChild(0) prev(0) next(0)
CPOSTAG	-1 0
FEATS	-1 0
DEPREL	-1 0 rightChild(-1) leftChild(0)
CHUNKID	-1 0 1
CHUNKTYPE	-1 0 1

Table 2. Feature model for Hindi

Field	*Tokens*
FORM	parent(-1) leftChild(0) prev(-1)
LEMMA	-1 0
POSTAG	-1 0 1 2 3 parent(-1) rightChild(-1) leftChild(0) prev(-1) next(-1) prev(0) next(0)
CPOSTAG	-1 0 1
FEATS	-1 0
DEPREL	-1 0 rightChild(-1) leftChild(0)
CHUNKID	-3 -2 -1 0 1 2 3
COREF	-3 -2 -1 0 1 2 3

Table 3. Feature model for Bangla

Field	*Tokens*
FORM	-1 0 parent(-1) rightChild(-1) leftChild(-1) leftChild(0) prev(-1)
LEMMA	-1 0 1
POSTAG	-1 0 1 2 3 parent(-1) leftChild(-1) rightChild(-1) leftChild(0) prev(-1) next(-1) prev(0) next(0)
CPOSTAG	-1 0
FEATS	-1 0 1
DEPREL	-1 0 rightChild(-1) leftChild(0)
CHUNKID	-2 -1 0 1 2
COREF	-2 -1 0 1 2

Table 4. Feature model for Telugu

The main difference with respect to the baseline feature model is the addition of the chunk and coreference fields as features.

In the experiments in Attardi and Dell'Orletta (2008) the best accuracy was obtained using as a chunk feature the combination EOC/TYPE that represents the distance of a token from its end of chunk in combination with the chunk type. Experiments with this feature on the Hindi language corpus, the only one which supplies the necessary information for chunks, did not prove to be effective.

DeSR can also exploit morphological information in terms of morphological agreement between head and dependent. The parser has been customized to extract such information from the morphological information provided in the training corpus. Morphological agreement is controlled by the configuration parameter of the parser MorphoAgreement.

Additionally, DeSR can learn about words that are typically used in dependencies for time and locations: this is enabled by a configuration parameter (PrepChildEntityType) that collects during training all words that are used as dependent in a relation of type time (tag k7t) or (location (tag k7p). This information is only available in the fine variant of the treebanks.

3 Experiments

In the experiments we used the DeSR (Dependency Shift Reduce) parser, freely available from Sourceforge.[1]

The issues we tried to address in the experiments were:

- effectiveness of single MLP parsers
- effectiveness of morph agreement and entity types
- effectiveness of stacking and parser combination

3.1 Experimental setup

The parser can be configured using a number of parameters through a configuration file. The feature model can be defined using the notation shown in the previous section.

We describe the configurations of DeSR that were used in our experiments and provide an indication on their relative accuracy.

The parser can be run reading the input in either forward or reverse direction. On the development set the former achieved slightly better LAS accuracy: on coarse Hindi 87.56% vs 87.10%, on Bangla 73.67% vs 72.11%, while on Telugu 68.78% vs 69.62% and similarly on the fine versions of the treebanks.

Another variant of the parser is the stacked version, which we introduced first for our CoNLL Shared Task 2008 participation (Ciaramita et al, 2008) and further developed in Attardi and Dell'Orletta (2009), which resembles the solution proposed at the same time by Nivre and R. McDonald (2008). In this configuration, the sentence is parsed twice: the second parsing stage is given information on the output of the first parser by means of additional features extracted from the parse tree obtained from the first stage.

As shown in our previous work by Attardi and Dell'Orletta (2009), it is best to use a less accurate parser in the first stage, since this provides more examples of mistakes that the second stage can learn how to correct. We use a configuration of DeSR based on a Maximum Entropy classifier with a beam size of one as a “poor” first stage parser.

The *Guided Reverse* parser analyzes the input in the forward direction using hints extracted from a poor parser run in the reverse direction.

The *Reverse Revision* parser analyzes the input in the reverse direction using hints extracted from a poor parser run in the forward direction.

We used the results from the development set to select the best parsers and combine their output using the voted combination algorithm also described in Attardi and Dell'Orletta (2009).

The algorithm avoids the complexity of previous solutions which involve computing the minimal spanning trees of graphs and exploits the fact that its inputs are trees. The algorithm works by combining the trees top down and has linear complexity: it has been shown to produce a combination that is as good as more costly solutions (Surdeanu, 2010).

In the submitted run we performed a combination of four parser outputs: Guided Reverse (*rev*), Reverse Revision (*rev2*), Reverse (*R*) and Refined (*Ref*) (see Figure 1). The *Ref* output was generated using a standard forward version of DeSR with the following additional features:

Field	*Token*
POSTAG	-2
CPOSTAG	1

Table 5. Additional features of the refined model

[1] http://sourceforge.net/projects/desr/

All versions used a Multilayer Perceptron classifier with 180 hidden features and a learning rate of 0.01.

Bangla and Telugu have very small training sets (less than 7000 tokens in the CoNLL-X version) and this might lead to problems of overfitting on the training corpus. To avoid this we chose to use a small number of training iterations (< 20) for all our experiments.

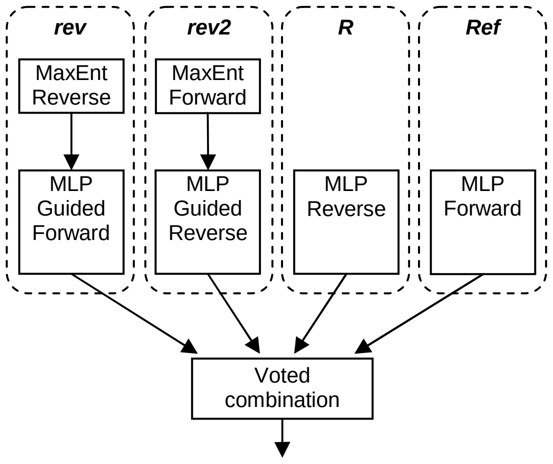

Figure 1. Parser combination schema

3.2 Results

The following table shows the Labeled Attachment Scores on the test set of the individual parsers as well as of their combination:

Corpus	*rev*	*rev2*	*Ref*	*R*	*Comb*
Hindi (coarse)	76.17	68.31	87.17	87.72	88.98
Bangla (coarse)	74.40	74.30	74.40	71.18	74.61
Telugu (coarse)	68.78	67.45	67.45	68,95	67.61
Hindi (fine)	74.45	67.44	86.02	85.50	87.49
Bangla (fine)	69.72	71.07	70.66	67.01	71.07
Telugu (fine)	68.45	65.28	65.61	67.95	65.78

Table 6. LAS of several configurations of DeSR

Quite surprisingly, the scores for the Guided Reverse (*rev*) and Reverse Revision (*rev2*) versions, are quite lower on the Hindi test set than on the development set: indeed they were chosen for the combination because they were the best. Despite this, the combination was still effective, achieving even a higher score on the test set than on the development set.

The following table shows our official results in comparison with the best scores achieved in the ICON 2010 contest. LAS is the Labeled Attachment Score and UAS stands for Unlabeled Attachment Score.

Corpus	*LAS*	*Best LAS*	*UAS*	*Best UAS*
Hindi (coarse)	**88.98**	**88.98**	**94.57**	**94.57**
Bangla (coarse)	74.61	**75.65**	**88.24**	**88.24**
Telugu (coarse)	67.45	**69.45**	**90.15**	**90.15**
Hindi (fine)	87.49	**88.63**	**94.78**	**94.78**
Bangla (fine)	**70.66**	**70.66**	**87.41**	**87.41**
Telugu (fine)	65.61	**70.12**	90.48	**91.82**

Table 7. Official results

After the official submission we performed some further experiments, in particular for Bangla and Telugu. We removed the sequence number also from the CHUNKID field and changed the width of the boundaries for that field in the feature model obtaining some further improvements that are listed in the following table.

Corpus	*LAS*	*Best LAS*	*UAS*	*Best UAS*
Hindi (coarse)	**88.98**	**88.98**	**94.57**	**94.57**
Bangla (coarse)	**75.96**	75.65	**88.76**	88.24
Telugu (coarse)	**69.62**	69.45	**91.15**	90.15
Hindi (fine)	87.49	**88.63**	**94.78**	**94.78**
Bangla (fine)	**71.07**	70.66	**88.14**	87.41
Telugu (fine)	67.78	**70.12**	91.15	**91.82**

Table 8. Improved results.

To appreciate the overall improvements due to our final parser configuration, we can compare these results with those achieved using a simple baseline configuration, consisting of a single pass of DeSR, the baseline feature model in Table 1 and without the use of features MorphoAgreement and PrepChildEntityType. Table 9 shows relevant improvements on both Hindi and Bangla, and minor improvements on Telugu.

	Baseline		*Improvement*	
Corpus	*LAS*	*UAS*	*LAS*	*UAS*
Hindi (coarse)	83.21	90.44	+ 5.77	+ 4.13
Bangla (coarse)	65.45	82.41	+ 10.51	+ 6.36
Telugu (coarse)	67.28	88.65	+ 2.34	+ 2.5
Hindi (fine)	81.44	90.82	+ 6.05	+ 3.96
Bangla (fine)	64.31	83.98	+ 6.76	+ 4.16
Telugu (fine)	65.94	89.32	+ 1.84	+ 1.83

Table 9. Improvements over baseline results.

4 Software Performance

In the experiments we used DeSR with an MLP classifier, which is quite efficient both in training and in parsing.

The following table shows the time to perform training and parsing on the various corpora, using a linux server with a 2.53 GHz Intel Xeon and 12 GB of memory using a single core.

Corpus	*Training*	*Parsing*
Hindi (coarse)	8’42’’	8’’
Bangla (coarse)	20’’	0.8’’
Telugu (coarse)	12’’	0.4’’
Hindi (fine)	13’ 25’’	11’’
Bangla (fine)	29’’	0.9’’
Telugu (fine)	16’’	0.5’’

Table 10. DeSR performance on the Indian languages

5 Conclusion

In this paper we presented the DeSR dependency parser and the use of chunks in the feature model to improve the parser performance on the Indian languages.

The use of specific features such as chunk information, morphological agreement and entity types together with parser combination results on the average in a 5.5 points improvement over our baseline LAS score.

DeSR achieved the best UAS (Unlabeled Attachment Score) results in all the three Indian languages and it was also best on LAS (Labeled Attached Score) on the coarse corpora and on the Bangla fine corpus (unofficial run after the official submission).

These results show that DeSR can adapt quite well to different corpora including quite small ones.

Reference

G. Attardi. 2006. Experiments with a Multilanguage non-projective dependency parser. In *Proc. of the Tenth CoNLL*. 166-170. Association for Computational Linguistics.

G. Attardi, A. Chanev, M. Ciaramita, F. Dell'Orletta and M. Simi. 2007. Multilingual Dependency Parsing and Domain Adaptation using DeSR. *Proceedings the CoNLL Shared Task Session of EMNLP-CoNLL 2007*. 1112-1118.

G. Attardi and F. Dell’Orletta. 2008. Chunking and Dependency Parsing. *Proceedings of LREC 2008 Workshop on Partial Parsing*, Marrakech.

G. Attardi and F. Dell'Orletta. 2009. Reverse Revision and Linear Tree Combination for Dependency Parsing. *Proc. of NAACL HLT 2009*. 261-264.

G. Attardi, F. Dell’Orletta, M. Simi and J. Turian. 2009. Accurate Dependency Parsing with a Stacked Multilayer Perceptron. *Proc. of Workshop Evalita 2009*, ISBN 978-88-903581-1-1.

A. Bharati, R. Sangal and D. M. Sharma. 2007. SSF: Shakti Standard Format. Language Technologies Research Centre, International Institute of Information Technology, Kharagpur, India.

S. Buchholz and E. Marsi. 2006. CoNLL-X Shared Task on Multilingual Dependency Parsing. In *Proc. of the Tenth CoNLL*. 149-164.

M. Ciaramita, G. Attardi, F. Dell'Orletta and M. Surdeanu. 2008. DeSRL: A Linear-Time Semantic Role Labeling System. *Proceedings the Twelfth Conference on Natural Language Learning*, Manchester.

S. Husain, P. Mannem, B. Ambati and P. Gadde. 2010. The ICON-2010 tools contest on Indian language dependency parsing. In Proc of *ICON-2010 tools contest on Indian language dependency parsing*. Hyderabad, India.

J. Nivre and R. McDonald. 2008. Integrating Graph-Based and Transition-Based Dependency Parsers. In *Proc. of ACL-08: HLT*, 950–958.

M. Surdeanu, R. Johansson, A. Meyers, L. Màrquez and J. Nivre. 2008. The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies. 2008. In Proc. of CoNLL 2008, 159–157.

M. Surdeanu and Christopher D. Manning. 2010. Ensemble Models for Dependency Parsing: Cheap and Good? In *Proc. of the North American Chapter of the Association for Computational Linguistics Conference* (NAACL-2010).