# Parsing Indian Languages with MaltParser

**Joakim Nivre**
Uppsala University
Department of Linguistics and Philology
E-mail: joakim.nivre@lingfil.uu.se

## Abstract

This paper describes the application of MaltParser, a transition-based dependency parser, to three Indian languages – Bangla, Hindi and Telugu – in the context of the NLP Tools Contest at ICON 2009. In the final evaluation, MaltParser was ranked second among the participating systems and achieved an unlabeled attachment score close to 90% for Bangla and Hindi, and over 85% for Telugu, while the labeled attachment score was 15–25 percentage points lower. It is likely that the high unlabeled accuracy is achieved thanks to a relatively low syntactic complexity in the data sets, while the low labeled accuracy is due to the limited amounts of training data.

## 1 Introduction

The NLP Tools Contest at ICON 2009 consisted in training and evaluating dependency parsers for three Indian languages: Bangla, Hindi and Telugu. I participated using the freely available MaltParser system (Nivre et al., 2006a), which implements a transition-based approach to dependency parsing, and which has previously been applied to over twenty languages (Nivre et al., 2006b; Nivre et al., 2007a; Nivre et al., 2007b; Nivre et al., 2008). In this paper, I give a brief introduction to MaltParser (Section 2), describe the process of optimizing the system for the three different languages (Section 3), report the experimental results from the final evaluation (Section 4), and conclude with some ideas for future work (Section 5).

## 2 MaltParser

MaltParser (Nivre et al., 2006a) implements the transition-based approach to dependency parsing, which has two essential components:

- A transition system for mapping sentences to dependency trees
- A classifier for predicting the next transition for every possible system configuration

Given these two components, dependency parsing can be realized as deterministic search through the transition system, guided by the classifier. With this technique, parsing can be performed in linear time for projective dependency trees and quadratic time for arbitrary (possibly non-projective) trees (Nivre, 2008).

### 2.1 Transition Systems

MaltParser comes with a number of built-in transition systems, but we limit our attention to the two systems that have been used in the parsing experiments: the arc-eager projective system first described in Nivre (2003) and the non-projective transition system based on the method described by Covington (2001). For a more detailed analysis of this and other transition systems for dependency parsing, see Nivre (2008).

A configuration in the arc-eager projective system contains a stack holding partially processed tokens, an input buffer containing the remaining tokens, and a set of arcs representing the partially built dependency tree. There are four possible transitions (where *top* is the token on top of the stack and *next* is the next token in the input buffer):

- LEFT-ARC($r$): Add an arc labeled $r$ from *next* to *top*; pop the stack.

| | | FORM | LEMMA | CPOS | POS | FEATS | DEPREL |
|---|---|---|---|---|---|---|---|
| Stack: | *top* | **1** | 2 | 3 | **4** | 5 | **6** |
| Stack: | *top*−1 | | | | **7** | | |
| Input: | *next* | **8** | 9 | 10 | **11** | 12 | **13** |
| Input: | *next*+1 | **14** | 15 | | **16** | 17 | |
| Input: | *next*+2 | | | | **18** | | |
| Input: | *next*+3 | | | | **19** | | |
| Tree: | head of *top* | **20** | | | 21 | | |
| Tree: | leftmost dep of *top* | | | | 22 | | **23** |
| Tree: | rightmost dep of *top* | 24 | | | 25 | | **26** |
| Tree: | leftmost dep of *next* | 27 | | | 28 | | **29** |
| String: | predecessor of *top* | 30 | | | 31 | | |
| String: | successor of *top* | | | | 32 | | |
| String: | predecessor of *next* | | | | 33 | | |
| String: | successor of *next* | | | | 34 | | |
| Second stack: | top word | | | | **35** | | |
| Second stack: | bottom word | | | | **36** | | |

Table 1: Feature pool for optimization with columns representing data fields and rows representing words defined relative to the stack, input buffer, partially built tree, input string, and second temporary stack. Features with boldface numbers belong to the baseline model; features 13, 35 and 36 are relevant only for the non-projective transition system.

- RIGHT-ARC($r$): Add an arc labeled $r$ from *top* to *next*; push *next* onto the stack.
- REDUCE: Pop the stack.
- SHIFT: Push *next* onto the stack.

Although this system can only derive projective dependency trees, the fact that the trees are labeled allows non-projective dependencies to be captured using the pseudo-projective parsing technique proposed in Nivre and Nilsson (2005).

The non-projective system uses a similar type of configuration but adds a second temporary stack. There are again four possible transitions:

- LEFT-ARC($r$): Add an arc labeled $r$ from *next* to *top*; push *top* onto the second stack.
- RIGHT-ARC($r$): Add an arc labeled $r$ from *top* to *next*; push *top* onto the second stack.
- NO-ARC: Push *top* onto the second stack.
- SHIFT: Empty the second stack by pushing every word back onto the stack; then push *next* onto the stack.

Unlike the first system, this allows the derivation of arbitrary non-projective dependency trees.

### 2.2 Classifiers

Classifiers can be induced from treebank data using a wide variety of different machine learning methods, but all experiments reported below use support vector machines with a polynomial kernel, as implemented in the LIBSVM package (Chang and Lin, 2001) included in MaltParser. The task of the classifier is to map a high-dimensional feature vector representation of a parser configuration to the optimal transition out of that configuration.

The features used in our system all represent attributes of tokens and have been extracted from the following fields of the CoNLL data representation (Buchholz and Marsi, 2006): FORM, LEMMA, CPOSTAG, POSTAG, FEATS, and DEPREL. The pool of features used in the experiments are shown in Table 1, where rows denote tokens in a parser configuration – defined relative to the stack, the input buffer, the partially built dependency tree, the input string, and the second temporary stack – and columns correspond to data fields. Each non-empty cell represents a feature, and features are numbered for future reference. Features with boldface numbers are present in the baseline model, while the other features have been explored in feature selection experiments. Note that features 15, 34 and 35 are relevant only for the non-projective transition system. (Features 34 and 35

refer to the second temporary stack, and feature 15 refers to the dependency label of *next*, which will always be undefined in the projective system.)

## 3 Optimization

Although MaltParser is a language-independent system in the sense that it can be trained on data from any language, it is usually possible to improve parsing accuracy by optimizing parameters of the transition system and the classifier. In this section, we describe the different stages of optimization performed for Bangla, Hindi and Telugu.

### 3.1 Data and Methodology

For all three languages, data were provided in the form of one training set and one validation set, which I merged into a single training set and used for ten-fold cross-validation using a pseudo-randomized split. This resulted in a training set of 1130 sentences (7260 tokens) for Bangla, 1651 sentences (15029 tokens) for Hindi, and 1615 sentences (6207 tokens) for Telugu. It is worth noting that the average sentence length in all data sets is small, with 6.4 tokens for Bangla, 9.1 for Hindi, and 3.8 for Telugu.

The baseline model for all three languages was MaltParser trained with default settings, including the default feature model illustrated in Table 1. In order to incorporate a change to the model, I required an improvement in both labeled attachment score (LAS) and unlabeled attachment score (UAS).

The development period was divided into two phases, with a preliminary evaluation after the first phase. After this evaluation, new versions of the data sets were distributed for all three languages, where the number of distinct dependency labels had been reduced to twelve coarse-grained labels that were the same across all three languages (although all labels did not occur in all languages). In the final evaluation, participants had to submit results both for the original fine-grained labels and for the new coarse-grained labels.

It would have been possible to optimize MaltParser separately for the two levels of granularity, but I decided to use only the coarse-grained labels during the second development phase, starting from the models that had been optimized for the fine-grained labels during the first phase. All results in the final evaluation are therefore based on models optimized for the coarse-grained labels, which in fact led to a small drop in performance for Bangla and Hindi with fine-grained labels, as compared to the preliminary evaluation.

### 3.2 Transition Systems

I evaluated three different transition systems. In addition to the arc-eager projective system and the non-projective system described in Section 2.1, I used the arc-standard projective system described in Nivre (2008). I found that the non-projective system gave the highest accuracy for Bangla and Telugu, while the arc-eager projective system worked best for Hindi. I also tried combining the two projective systems with pseudo-projective parsing (Nivre and Nilsson, 2005), which gave a marginal improvement with the arc-eager system for Bangla – but results were still inferior to those obtained with the non-projective system – and no improvement for any of the other combinations. Based on these experiments, I decided to use the non-projective system for Bangla and Telugu and the strictly projective arc-eager system for Hindi in all the following experiments.

### 3.3 Classifiers

Classifier optimization essentially consists in optimizing two different aspects:

- Feature model
- Learning algorithm parameters

Ideally, the feature model and the parameters of the learning algorithm should be optimized jointly, but in practice we are forced to use some kind of sequential or interleaved optimization. In this case, I started by optimizing the feature model and the learning parameters for the fine-grained dependency labels used during the first development phase. During the second phase, I performed additional feature selection experiments with the coarse-grained labels but without changing any parameters of the learning algorithm.

#### 3.3.1 Feature Models

In order to tune the feature models, I first performed backward feature selection to find out whether any of the features in the default model could be omitted. This resulted in the exclusion of feature number 23 for all three languages: the dependency label of the leftmost dependent of *top*. This is a feature that is useful for some languages but apparently not for Bangla, Hindi and Telugu.

I then performed several rounds of forward feature selection, first with the fine-grained data sets and later with the coarse-grained data sets, which resulted in the following addition of features:

- Features 2 and 9, the lemma of *top* and *next*, respectively, were added for all three languages. For Bangla, feature 15, the lemma of *next*+1, was added as well.
- Features 3 and 10, the coarse-grained part of speech of *top* and *next*, respectively, were added for all three languages.
- Features 5 and 12, the morphological features associated with *top* and *next*, respectively, were added for all three languages. For Bangla, feature 17, the morphological features of *next*+1, was added as well.
- Additional part-of-speech features were added for Bangla (22, 25) and Hindi (21, 25, 28, 31).
- Additional lexical (word form) features were added for Bangla (24), Hindi (27, 30) and Telugu (24, 27).
- Conjoined features were added for Bangla (1&4, 4&11, 8&11) and Hindi (1&4, 1&8, 4&11).[1]

The greatest improvement for all languages came from the addition of features 5 and 14, that is, the morphological features associated with *top* and *next*, which improved the LAS by almost 10 percentage points for Hindi.

#### 3.3.2 Learning Algorithm Parameters

The learning parameter optimization was done in two steps. The first was to check whether accuracy could be improved by splitting the classification problem into two steps, first choosing the basic transition and then (in the case of LEFT-ARC and RIGHT-ARC) choosing the arc label. This turned out to improve accuracy for all three languages. There was a small additional improvement from having separate label classifiers for the LEFT-ARC and RIGHT-ARC case.

[1] It should be noted that the use of support vector machines with a degree-2 polynomial kernel implies that all explicit features are conjoined implicitly by the kernel. This includes the explicitly conjoined features, which are thus combined with both simple and conjoined features.

The second step was to optimize the SVM parameters, in particular the cost parameter $C$, which controls the tradeoff between minimizing training error and maximizing margin. This parameter was set to 0.75 for Bangla, 1.0 for Hindi, and 0.875 for Telugu.

### 3.4 Final Models

Table 2 shows the cross-validation results for the final optimized models on the data with fine-grained dependency labels, compared to the baseline model. For all three languages, we find the largest improvement in LAS, with 7.4 percentage points for Bangla, 12.9 for Hindi, and 5.0 for Telugu, while the corresponding improvement in UAS is only 3.2 percentage points for Bangla, 7.0 for Hindi, and 1.9 for Telugu.

Table 3 shows the corresponding results with coarse-grained labels. The overall pattern is very similar, but with slightly larger improvements, in particular for Bangla and Hindi (LAS: 10.6, 14.3, 5.3; UAS: 6.3, 9.1, 2.3), and the labeled attachment scores of the optimized models are generally about 4 percentage points higher than for the fine-grained label set.

## 4 Evaluation

Tables 2 and 3 also show the evaluation results on the final test sets. For Bangla and Hindi, the results are slightly better than the cross-validation results, especially with coarse-grained labels (over 2 percentage points for Bangla and over 4 percentage points for Hindi), but for Telugu they are in fact lower than the baseline results. On the other hand, all test sets are small, consisting only of 150 sentences, so it is natural to expect considerable variance in the results.

Of the systems that submitted test results for all languages, MaltParser was ranked second with respect to the average score over all languages (for LAS, UAS and LA with both fine-grained and coarse-grained labels), but for both Bangla and Telugu there were in fact two other systems that achieved better results. The closest margin to the best performing system was for Hindi, where the MaltParser results were about 1 percentage point below the best results with both fine-grained and coarse-grained labels. It may be significant that Hindi had by far the largest training set, more than twice the size of the two others.

For all three languages, there is a wide gap be-

| | Cross-Validation | | | | | | Final Test | | |
|---|---|---|---|---|---|---|---|---|---|
| | Baseline | | | Optimized | | | | | |
| **Language** | **LAS** | **UAS** | **LA** | **LAS** | **UAS** | **LA** | **LAS** | **UAS** | **LA** |
| Bangla | 61.7 | 83.5 | 63.8 | 69.7 | 86.5 | 72.6 | 70.5 | 88.7 | 72.9 |
| Hindi | 56.5 | 78.9 | 58.7 | 69.6 | 86.0 | 72.2 | 73.4 | 89.8 | 75.3 |
| Telugu | 62.9 | 87.4 | 65.8 | 67.9 | 89.3 | 70.9 | 57.6 | 84.7 | 58.5 |

Table 2: Parsing accuracy for Bangla, Hindi and Telugu with fine-grained dependency labels; cross-validation results with baseline and optimized models on training sets (including development sets); test set results for optimized models trained on entire training set. LAS = labeled attachment score; UAS = unlabeled attachment score; LA = label accuracy.

| | Cross-Validation | | | | | | Final Test | | |
|---|---|---|---|---|---|---|---|---|---|
| | Baseline | | | Optimized | | | | | |
| **Language** | **LAS** | **UAS** | **LA** | **LAS** | **UAS** | **LA** | **LAS** | **UAS** | **LA** |
| Bangla | 63.2 | 80.3 | 66.5 | 73.8 | 86.6 | 76.7 | 76.1 | 89.0 | 79.6 |
| Hindi | 59.8 | 79.1 | 62.3 | 74.1 | 86.2 | 77.2 | 78.2 | 89.4 | 81.1 |
| Telugu | 66.8 | 86.7 | 70.4 | 72.1 | 89.0 | 75.7 | 62.4 | 86.3 | 63.0 |

Table 3: Parsing accuracy for Bangla, Hindi and Telugu with coarse-grained dependency labels; cross-validation results with baseline and optimized models on training sets (including development sets); test set results for optimized models trained on entire training set. LAS = labeled attachment score; UAS = unlabeled attachment score; LA = label accuracy.

tween the labeled and unlabeled attachment score. For Telugu with fine-grained labels (worst case) it is over 25 percentage points, but even for Hindi with coarse-grained labels (best case) it is more than 10 points, which is more than we typically find for other languages in the dependency parsing literature. It is possible that the more semantically oriented nature of the Paninian annotation scheme makes labeled parsing more difficult than in schemes based on more surface-oriented grammatical functions. However, I believe that stronger explanatory factors can be found in properties of the data sets.

As noted above, the average sentence length is low in all data sets, which means that most sentences will not have a very complex syntactic structure, which in turn favors unlabeled parsing accuracy. On the other hand, the training sets are all very small, which means that many labels are sparsely represented, which in turn has a negative impact on labeled parsing accuracy. If this hypothesis is correct, then we should expect to see an improvement of labeled accuracy as the data sets grow larger, but it is also possible that unlabeled accuracy will in fact drop as longer and more complex sentences are added to the data sets.

Table 4 provides a more detailed analysis of the results by reporting labeled precision and recall for the twelve dependency types in the coarse-grained label set (with unlabeled precision and recall in parentheses), based on cross-validation over the entire training sets. For Bangla and Telugu, the highest accuracy is achieved for the label *main*, assigned to the root of each dependency tree, which has both precision and recall over 90%. The second highest accuracy is reported for the label *ccof*, assigned to conjuncts in a coordination, with both precision and recall over 80%. For both these relations, labeled precision/recall is also very close to unlabeled precision/recall. For Hindi, the picture is rather different, which may be partly due to the fact that a different transition system was used for this language. First of all, precision for *main* is much lower, which is due to fragmented parses with too many root nodes. Secondly, the *r6* label, assigned to possessives, has a precision and recall that is higher than the *ccof* label and also higher than for the *r6* label in the other two languages.

Looking at the *karaka* relations *k1-ext*, *k2-ext* and *k7-ext*, finally, we see that they have very high unlabeled precision and recall (85–90%) in all languages but substantially lower labeled precision and recall. The same kind of pattern is found also for the *vmod-rest* label. Whether these patterns are due primarily to an intrinsic difficulty of the Paninian scheme of categorization or to the lim-

| | Bangla | | | | Hindi | | | | Telugu | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Precision** | | **Recall** | | **Precision** | | **Recall** | | **Precision** | | **Recall** | |
| ccof | 83.4 | (83.2) | 83.2 | (83.5) | 78.6 | (80.3) | 77.2 | (77.7) | 80.7 | (84.1) | 80.1 | (83.8) |
| jjmod | – | (–) | – | (–) | 50.0 | (66.7) | 9.1 | (15.2) | – | (–) | – | (–) |
| fragof | – | (–) | 0.0 | (0.0) | – | (–) | – | (–) | – | (–) | – | (–) |
| k1-ext | 68.8 | (89.5) | 70.7 | (90.3) | 72.4 | (90.6) | 71.1 | (92.3) | 60.1 | (87.4) | 64.4 | (90.1) |
| k2-ext | 68.7 | (89.4) | 76.7 | (92.9) | 69.1 | (92.0) | 73.8 | (91.8) | 63.0 | (91.7) | 62.9 | (92.7) |
| k7-ext | 65.8 | (84.5) | 64.2 | (86.8) | 79.3 | (90.4) | 78.3 | (92.0) | 66.0 | (85.4) | 62.2 | (88.7) |
| main | 92.3 | (92.3) | 93.4 | (93.4) | 77.2 | (77.2) | 89.7 | (89.7) | 95.9 | (96.0) | 96.0 | (96.2) |
| nmod | 45.9 | (57.1) | 26.4 | (39.0) | 58.0 | (64.9) | 32.4 | (38.8) | 49.6 | (68.1) | 34.9 | (44.8) |
| r6 | 75.8 | (79.6) | 77.4 | (79.2) | 86.0 | (86.6) | 87.1 | (88.8) | 67.7 | (78.5) | 50.6 | (62.1) |
| rbmod | – | (–) | – | (–) | – | (–) | 0.0 | (0.0) | – | (–) | 0.0 | (25.0) |
| relc | 48.6 | (56.8) | 35.3 | (35.3) | 56.0 | (60.3) | 25.8 | (30.2) | 40.0 | (60.0) | 12.5 | (50.0) |
| vmod-rest | 67.9 | (84.9) | 64.0 | (86.4) | 62.9 | (85.0) | 61.3 | (84.4) | 65.8 | (85.4) | 67.6 | (86.9) |

Table 4: Labeled precision and recall for coarse-grained dependency types in Bangla, Hindi and Telugu; unlabeled precision/recall in parentheses. (For precision, – means that the parser did not assign this label; for recall, – means that the label did not occur in the data set.)

ited amount of training data for labeled parsing, as hypothesized above, must remain a topic for future research.

## 5 Conclusion

I have presented the work done to optimize MaltParser for Bangla, Hindi and Telugu in the NLP Tools Contest at ICON 2009. Using MaltParser with default settings as the baseline, optimization improved the labeled attachment scores by 7–13 percentage points and the unlabeled attachment scores by 2–5 percentage points on development data. This led to the second best results for Hindi and on average, and the third best results for Bangla and Telugu..

It is not impossible that parsing accuracy can be improved further by performing more extensive feature selection experiments. However, a more substantial increase in accuracy can only come from a substantial increase in the amount of training data, since the data sets used for training and validation in the NLP Tools Contest are really too small for data-driven approaches to syntactic parsing. Having more data available will also make it possible to test the hypothesis that unlabeled parsing accuracy is currently quite high thanks to a relatively low syntactic complexity, while labeled accuracy is very low because of data sparseness.

Another important source of information for further improvement is the growing body of work focusing specifically on parsing Indian languages, exemplified by Bharati et al. (2009a) and Bharati et al. (2009b). By taking into account the special features of these languages, it should be possible to attain higher accuracy than with a completely agnostic language-independent system.

## Acknowledgments

I want to thank the organizers of the NLP Tools Contest for their sterling effort from start to finish.

## References

Akshar Bharati, Mridul Gupta, Vineet Yadav, Karthik Gali, and Dipti Misra Sharma. 2009a. Simple parser for indian languages in a dependency framework. In *Proceedings of the Third Linguistic Annotation Workshop*, pages 162–165.

Akshar Bharati, Samar Husain, Dipti Misra, and Rajeev Sangal. 2009b. Two stage constraint based hybrid approach to free word order language dependency parsing. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT)*, pages 77–80.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 149–164.

Chih-Chung Chang and Chih-Jen Lin, 2001. *LIBSVM: A Library for Support Vector Machines*. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

Michael A. Covington. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the*

*39th Annual ACM Southeast Conference*, pages 95–102.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 99–106.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2006a. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*, pages 2216–2219.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülsen Eryiğit, and Svetoslav Marinov. 2006b. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 221–225.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007a. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 915–932.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülşen Eryiğit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007b. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13:95–135.

Joakim Nivre, Igor M. Boguslavsky, and Leonid L. Iomdin. 2008. Parsing the SynTagRus treebank of russian. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 641–648.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34:513–553.