Proceedings of
ICON10 NLP TOOLS CONTEST: INDIAN LANGUAGE DEPENDENCY PARSING

Kharagpur, India

1st August – 1st December 2010

Acknowledgement

We'd like to thank all the current dependency treebank annotators/validators; without their effort the contest would not have been possible.

Hindi: Ms. Preeti Pradhan, Ms. Nandini Upasani, Mr. Samar Husain, Ms. Rafiya Begum, Dr. Dipti Misra Sharma,
Telugu: Mr. Sudheer Kolachina, Mr. Phani Gadde, Mr. Meher Vijay,
Bangla: Dr. Soma Paul.

Thanks are also due to Anita Chaturvedi, Sarita Shrivastav, Alpana Agarwal and Kiran Kapoor who have worked on other layers of the treebank. Preeti Shukla, Susanta Kishore, Viswanath Naidu, Bharat Ambati, and others have previously worked on the treebank development. Many thanks to Dr. Anil Kumar Singh for providing the 'Sanchay' syntactic annotation platform. We would also like to thank Mr. Abhijeet Gupta for providing the necessary web support.

Many thanks to Dr. Dipti Misra Sharma, Dr. Lakshmi Bai and Dr. Rajeev Sangal, who were instrumental in taking various crucial decisions. Thanks to all the reviewers for spending their valuable time on the reviews.

Finally, thanks to all the participants for their wonderful effort.

The Hindi treebank annotation is supported by the NSF grant (Award Number: CNS 0751202; CFDA Number: 47.070). Please note that the Hindi treebank is part of a developing treebank and is not yet publicly released.

The final release version of all the treebanks might have some differences from the ones released for the contest.

Samar Husain,
Prashanth Mannem,
Bharat Ambati,
Phani Gadde.

(ICON10 tools contest organizing committee.)

Review Committee

Bharat Ambati, IIIT-Hyderabad
Michael Elhadad, Ben-Gurion University of the Negev
Jennifer Foster, Dublin City University
Phani Gadde, IIIT-Hyderabad
Samar Husain, IIIT-Hyderabad
Sandra Kuebler, Indiana University
Prashanth Mannem, IIIT-Hyderabad
Ryan McDonald, Google Research, NY
Joakim Nivre, Uppsala University
Owen Rambow, Columbia University
Rajeev Sangal, IIIT-Hyderabad
Djamé Seddah, Alpage & Université Paris-Sorbonne
Dipti Misra Sharma, IIIT-Hyderabad
Khalil Sima'an, University of Amsterdam

Table of Contents

The ICON-2010 tools contest on Indian language dependency parsing

Samar Husain, Prashanth Mannem, Bharat Ambati and Phani Gadde
Language Technologies Research Centre, IIIT-Hyderabad, India.
{samar, prashanth, ambati, phani.gadde}@research.iiit.ac.in

Abstract

The ICON10 tools contest was dedicated to the task of dependency parsing for Indian languages (IL). Three languages namely, Hindi, Telugu and Bangla, were explored. The motivation behind the task was to investigate and solve the challenges in IL parsing by making annotated data available to the larger community.

1 Introduction

The tools contest at International Conference on Natural Language Processing (ICON) is a regular event that aims at building/improving Indian language (IL) NLP tools. Following the enthusiastic response of ICON09 contest on IL dependency parsing (Husain, 2009), a follow-up contest on the same topic was organized. Husain (2009) describes the participating systems. Crucial parsing issues, many IL specific, came to light and were discussed. However, efficient Indian Language (IL) parsing still remains a challenging task. Most Indian languages are morphologically rich and free word order (MoR-FWO). It is known that MoR-FWO languages pose various challenges for the task of parsing because of their non-configurationality. Also, the syntactic cues necessary to identify various relations in such languages are complex and distributed. This problem worsens in the context of data-driven dependency parsing due to non-availability of large annotated corpus. Past experiments on parser evaluation and parser adaptation for MoR-FWO languages (like Turkish, Basque, Czech, Arabic, Hebrew, etc.) have shown that there are a number of factors which contribute to the performance of a parser (Nivre et al., 2007b; Hall et al. 2007; McDonald and Nivre, 2007). For Hindi, (a) *difficulty in extracting relevant linguistic cues*, (b) *non-projectivity*, (c) *lack of explicit cues*, (d) *long distance dependencies*, (e) *complex linguistic phenomena*, and (f) *small corpus size*, have been suggested as possible reasons for low performance (Bharati et al., 2008, Ambati et al., 2010a). There has been a recent surge in addressing parsing for MoR-FOW languages (Nivre and McDonald, 2008; Nivre, 2009; Tsarfaty and Sima'an, 2008; Seddah et al., 2009; Gadde et al., 2010; Husain et al., 2009, Eryigit et al., 2008; Goldberg and Elhedad, 2009, Tsafarty et al., 2010; Mannem et al., 2009). It is our hope that the ICON10 tools contest will add to this knowledge.

2 Annotated Data

The data for all the three languages was annotated using the Computational Paninian Grammar (Bharati et al., 1995). The annotation scheme based on this grammar has been described in Begum et al. (2008) and Bharati et al. (2009a). Table 1 shows the training, development and the testing data sizes for the Hindi, Telugu and Bangla Treebanks.

Type	Lang.	Sent Count	Word Count	Avg. sent_length
	Hindi	2,972	64632	22.69
Train	Telugu	1,400	7602	5.43
	Bangla	980	10305	10.52
	Hindi	543	12617	23.28
Devel	Telugu	150	839	5.59
	Bangla	150	1196	7.97
	Hindi	320	6589	20.59
Test	Telugu	150	836	5.57
	Bangla	150	1350	9.0

Table 1. Treebank Statistics

2.1 Dependency Tagset

The tagset used in the dependency framework is syntactic-semantic in nature and has 59 labels. Keeping in mind the small size of the current treebanks an additional coarse-grained tagset containing 37 labels was derived from the original tagset. The coarse-grained tagset data was automatically created from the original treebank by mapping the original tagset to the coarse-grained tagset. Hence, two sets of data for each language were released. APPENDIX-I shows

the original tagset (henceforth referred as the fine-grained tagset). The mapping from fine to coarse tags is shown in APPENDIX-I.

2.2 Information in the released data

The released annotated data for the three languages has the following information:

(1) Morphological information
(2) Part of Speech (POS) tag
(3) Chunk boundary and chunk tag
(4) Chunk head information
(5) *Vibhakti*[1] of the head
(6) Dependency relation

Morph output has the following information

a) *Root*: Root form of the word
b) *Category*: Course grained POS·
c) *Gender*: Masculine/Feminine/Neuter
d) *Number*: Singular/Plural
e) *Person*: First/Second/Third person
f) *Case*: Oblique/Direct case
g) *Vibhakti*: Vibhakti of the word

POS and chunk annotation follows the scheme proposed by Bharati et al. (2006b). The dependency annotation for all the three languages was done on chunks. A chunk groups local dependencies that have no effect on the global dependency structure. In general, all the nominal inflections, nominal modifications (adjective modifying a noun, etc.) are treated as part of a noun chunk, similarly, verbal inflections, auxiliaries are treated as part of the verb chunk (Bharati et al., 2006b). For each sentence in the corpus, the POS tags and the chunks along with their head words are marked first and in the next step, dependencies are annotated between these chunk heads. The dependency relations between words within a chunk are not marked and they can be derived automatically.

Due to this scheme of annotation, the treebanks contain only inter-chunk relations. For Hindi, an automatic tool was used to expand the chunks to get the intra-chunk relations (Bharati etl al, 2009b). The performance of this tool is close to 96%. Due to unavailability of such a tool for Telugu and Bangla, the dependency relations in these treebanks are between chunk heads only.

Table 2 shows how all the above information has been marked in the treebanks. For Hindi, the morph features, POS tags, chunk tags and boundaries along with inter-chunk dependencies have been manually annotated. The head word of the chunk, its vibhkati and the intra-chunk dependencies were automatically marked. For Telugu and Bangla, only POS, chunk and inter-chunk dependency information are manually annotated. This holds true for training, development and the test data.

Lang	POS	Ch	Dep	Mo	Head	Vib.
Hin	Man	Man	Man/ Auto	Man	Auto	Auto
Tel	Man	Man	Man	Auto	Auto	Auto
Ban	Man	Man	Man	Auto	Auto	Auto

Table 2. Lang: Language, Hin: Hindi, Tel: Telugu, Ban: Bangla POS: POS tags, Ch: Chunk boundaries and tags, Dep: Dependency relations, Mo: Morphological features, Head: Chunk head information, Vib: *vibhakti* of head. Man: Manual, Auto: Automatic

3 Contest

3.1 Data format

The annotation for all the three treebanks were done in Shakti Standard Format (SSF) (Bharati et al., 2006a). SSF allows for a multi-layered representation in a single data structure. Each sentence is wrapped in a XML tag to indicate the start and end of a sentence. For each sentence, the word id, word form/chunk boundary, POS/chunk tag and features are listed in four columns respectively. The features column contains the morphological and dependency information.

Since CoNLL-X is a widely used representation for dependency parsers, hence the treebanks for all the three langauges were released in SSF as well as in CoNLL-X format.

As Telugu and Bangla treebanks do not have the intra-chunk relations, the sentences in the CoNLL-X data for these languages contain just the chunk heads. The non-head of a chunk are absent in the CoNLL-X format. However, the post-positions and auxiliary verbs (crucial for parsing ILs) which normally are non-heads in a chunk are listed as chunk head features.. The SSF data, on the other hand, retains the full sentences.

3.2 Evaluation

The standard dependency evaluation metrics like Unlabeled Attachment Score (UAS), Label Accuracy (LA), and Labeled Attachment Score (LAS) have been used to evaluate the submis-

[1] *Vibhakti* is a generic term for nominal case-marking, post-positions and verbal inflections, tense, aspect , modality.

sions (Nivre et al., 2007a). UAS is the percentage of words in the sentences across the entire test data that have correct parents. LA is the percentage of words with correct dependency label, while LAS is the percentage of words with correct parent and correct dependency label.

For evaluation on the test data, the teams were asked to submit two separate outputs with coarse grained and fine grained dependency labels. The three scores were computed for both the outputs.

4 Submissions

All the participating teams were expected to submit their system's results for both fine-grained and coarse-grained tagset data. In all, around 15 teams registered for the contest. 6 teams eventually submitted the results. Out of these, 4 teams submitted outputs for all the three languages and 2 teams did it only for one language.

4.1 Approaches

Attardi et al. (2010) use a transition based dependency shift reduce parser (DeSR parser) that uses a Multilayer Perceptron (MLP) classifier with a beam search strategy. They explore the effectiveness of morph agreement and chunk features by stacking MLP parses obtained from different configurations.

Ghosh et al. (2010) describe a dependency parsing system for Bengali in which MaltParser is used as a baseline system and its output is corrected using a set of post-processing rules.

Kosaraju et al. (2010) use MaltParser and explored the effectiveness of local morpho-syntactic features, chunk features and automatic semantic information. Parser settings in terms of different algorithms and features were also explored.

Abhilash and Mannem (2010) use a bidirectional parser with perceptron learning for all the three languages with rich context as features.

Kesedi et al. (2010) use a hybrid constraint based parser for Telugu. The scoring function for ranking the base parses is inspired by a graph-based parsing model and labeling.

Kolachina et al. (2010) use MaltParser to explore the effect of valency and arc-directionality on the parser performance. They employ feature propagation in order to incorporate such features during the parsing process. They also explore various blended systems to get the best accuracy.

Team	Approach	Learning Algorithm
Attardi	DeSR	Multilayer Perceptron
Abhilash	Bidirectional Parsing	Perceptron
Ghosh	MaltParser + Post processing	SVM
Kesedi	Constraint based hybrid Parsing	MaxEnt
Kolachina	MaltParser	SVM
Kosaraju	MaltParser	SVM

Table 3. Systems summary

4.1.1 Labeling

Labeling dependencies is a hard problem in IL dependency parsing due to the non-configurational nature of these languages. Ambati et al. (2010b) attribute this to, among others, the absence of postpositions and ambiguous postpositions.

Abhilash and Mannem (2010) used the bidirectional parser for unlabeled parsing and assigned labels in the next stage using a maximum entropy classifier. The constraint system of Kesedi et al. (2010) uses manually created grammar to get the labels. A maximum entropy labeler is then used to select the best parse. Teams which used MaltParser did unlabeled and labeled parsing together (Kolachina et al., 2010; Kosaraju et al., 2010). Ghosh et al. (2010) too used labels from MaltParser, but these labels were corrected in some cases using rules in a post-processing step. Attardi et al. (2010) too performed labeling along with unlabeled dependency parsing.

4.1.2 Features

Apart from the telugu constraint based parser (Kesedi et al, 2010), all the other teams used state of the art data driven dependency parsers which have been proposed earlier. And therefore, the features used in these systems become the crucial differentiating factor. In what follows we point out the novel linguistically motivated features used by the teams.

Attardi et al. (2010) use morph agreement as feature along with chunk features. For time and location dependency labels, word lists of dependents are collected during training to be used while parsing.

Kolachina et al. (2010) experiment with dependency arc direction and valency through fea-

ture propagation. Valency information is captured using the dependency labels of the outgoing arcs which are deemed mandatory for the parent node.

Kosaraju et al. (2010) successfully use features derived from a shallow parser's output for Hindi. Semantic features such as *human*, *nonhuman*, *inanimate* and *rest* are used for parsing without much success, mainly due to the difficulty in correctly predicting the semantic features for a new sentence.

It would be interesting to see the effect of all these novel features (morph agreement, valency information, semantic labels) in a single parser.

5 Results

Evaluation is done on system's output for both fine-grained and coarse-grained tagset data of all the three languages. Average of all the six (3 languages * 2 tagsets) outputs is taken for final evaluation. Out of 6 teams, 4 teams submitted all the six outputs. Gosh et al. (2010) submitted output for fine-grained Bangla data and Kesedi et al. (2010) submitted output for coarse-grained Telugu data. Best average UAS of 90.94% is achieved by Attardi et al. (2010). Whereas, best average LAS of 76.83% and LS of 79.14 is achieved by Kosaraju et al. (2010). Table 4 (a-c) shows the consolidated results for all the system.

6 Conclusion

Many new parsing approaches were successfully adapted for ILs in ICON10 tools contest. Previously unexplored features were also investigated. The state-of-the-art parsing accuracies for Hindi and Telugu were improved.

The large jump in the Hindi accuracies were mainly due to the high precision in the identification of intra-chunk relations. This is because, intra-chunk relations are mostly local syntactic dependencies that can be easily identified (Ambati et al., 2010b). The chunk level dependency issues pointed out during the ICON 2009 tools contest on IL dependency parsing (Husain, 2009), as well as in Bharati et al. (2008) and Ambati et al. (2010a) that lead to the low LAS in IL parsing still remain pertinent. This means that there remains a lot of scope for improvement, especially, in bridging the gap between UAS and LAS.

References

A. Abhilash and P. Mannem. 2010. Bidirectional Dependency Parser for Indian Languages. In Proc of *ICON-2010 tools contest on Indian language dependency parsing*. Kharagpur, India.

B. Ambati, S. Husain, J. Nivre and R. Sangal. 2010a. On the Role of Morphosyntactic Features in Hindi Dependency Parsing. *In Proceedings of NAACL-HLT 2010 workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2010), Los Angeles, CA.*

B. Ambati, S. Husain, S. Jain, D. M. Sharma and R. Sangal. 2010b. Two methods to incorporate 'local morphosyntactic' features in Hindi dependency parsing. *In Proceedings of NAACL-HLT2010 workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2010) LosAngeles, CA.*

G. Attardi, S. D. Rossi and M. Simi. 2010. Dependency Parsing of Indian Languages with DeSR. In Proc of *ICON-2010 tools contest on Indian language dependency parsing*. Kharagpur, India.

R. Begum, S. Husain, A. Dhwaj, D. M. Sharma, L. Bai and R. Sangal. 2008. Dependency Annotation Scheme for Indian Languages. *In Proceedings of The Third International Joint Conference on Natural Language Processing (IJCNLP). Hyderabad, India.*

A. Bharati, D. M. Sharma, S. Husain, L. Bai, R. Begam and R. Sangal. 2009a. *AnnCorra:TreeBanks for Indian Languages, Guidelines for Annotating Hindi Treebank.* http://ltrc.iiit.ac.in/MachineTrans/research/tb/DS-guidelines/DS-guidelines-ver2-28-05-09.pdf

A. Bharati, S. Husain, D. M. Sharma, L. Bai, and R. Sangal. 2009b. *AnnCorra:TreeBanks for Indian Languages, Guidelines for Annotating Intra-chunk dependency.* http://ltrc.iiit.ac.in/MachineTrans/research/tb/intra-chunk%20guidelines.pdf

A. Bharati, S. Husain, B. Ambati, S. Jain, D. M. Sharma and R. Sangal. 2008. Two semantic features make all the difference in Parsing accuracy. *In Proceedings of the 6th International Conference on Natural Language Processing (ICON-08), DAC Pune, India.*

A. Bharati, R. Sangal and D. M. Sharma. 2006a. SSF: Shakti Standard Format Guide. *LTRC-TR33*. http://ltrc.iiit.ac.in/MachineTrans/publications/technicalReports/tr033/SSF.pdf

A. Bharati, D. M. Sharma, L. Bai and R. Sangal. 2006b. AnnCorra: Annotating Corpora Guidelines for POS and Chunk Annotation for Indian Languages. *LTRC-TR31*. http://ltrc.iiit.ac.in/MachineTrans/publications/technicalReports/tr031/posguidelines.pdf

A. Bharati, V. Chaitanya, R. Sangal. 1995. *Natural Language Processing: A Paninian Perspective.* Prentice-Hall of India, New Delhi.

P. Gadde, K. Jindal, S. Husain, D. M Sharma, and R. Sangal. 2010. Improving Data Driven Dependency Parsing using Clausal Information. *In Proceedings of NAACL-HLT 2010, Los Angeles, CA. 2010.*

A. Ghosh, A. Das, P. Bhaskar and S. Bandyopadhyay. 2010. Bengali Parsing System at ICON NLP Tool Contest 2010. In Proc of *ICON-2010 tools contest on Indian language dependency parsing*. Kharagpur, India.

Y. Goldberg and M. Elhadad. 2009. Hebrew Dependency Parsing: Initial Results. *In Proceedings of the 11th IWPT09. Paris. 2009.*

J. Hall, J. Nilsson, J. Nivre, G. Eryigit, B. Megyesi, M. Nilsson and M. Saers. 2007. Single Malt or Blended? A Study in Multilingual Parser Optimization. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, 933—939

S. Husain, P. Gadde, B. Ambati, D. M. Sharma and R. Sangal. 2009. A modular cascaded approach to complete parsing. *In Proceedings of the COLIPS International Conference on Asian Language Processing 2009 (IALP). Singapore.*

S. Husain. 2009. Dependency Parsers for Indian Languages. *In Proceedings of ICON09 NLP Tools Contest: Indian Language Dependency Parsing. Hyderabad, India. 2009.*

S. R. Kesidi, P. Kosaraju, M. Vijay and S. Husain. 2010. A Two Stage Constraint Based Hybrid Dependency Parser for Telugu. In Proc of *ICON-2010 tools contest on Indian language dependency parsing*. Kharagpur, India.

S. Kolachina, P. Kolachina M. Agarwal, and S. Husain. 2010. Experiments with MaltParser for parsing Indian Languages. In Proc of *ICON-2010 tools contest on Indian language dependency parsing*. Kharagpur, India.

P. Kosaraju, S. R. Kesidi, V. B. R. Ainavolu and P. Kukkadapu. 2010. Experiments on Indian Language Dependency Parsing. In Proc of *ICON-2010 tools contest on Indian language dependency parsing*. Kharagpur, India.

P. Mannem, A. Abhilash and A. Bharati. 2009. LTAGspinal Treebank and Parser for Hindi. *Proceedings of International Conference on NLP, Hyderabad. 2009.*

R. McDonald and J. Nivre. 2007. Characterizing the Errors of Data-Driven Dependency Parsing Models. In *Proc of Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*

J. Nivre. 2009. Non-Projective Dependency Parsing in Expected Linear Time. In *Proc. of ACL IJCNLP*.

J. Nivre and R. McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proc. of ACL-HLT.*

J. Nivre, J. Hall, S. Kubler, R. McDonald, J. Nilsson, S. Riedel and D. Yuret. 2007a. The CoNLL 2007 Shared Task on Dependency Parsing. In *Proc of EMNLP/CoNLL-2007.*

J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kübler, S. Marinov and E Marsi. 2007b. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2), 95-135.

D. Seddah, M. Candito and B. Crabbé. 2009. Cross parser evaluation : a French Treebanks study. *In Proceedings of the 11th IWPT09. Paris. 2009.*

R. Tsarfaty, D. Seddah, Y. Goldberg, S. Kuebler, Y. Versley, M. Candito, J. Foster, I. Rehbein and L. Tounsi. 2010. Statistical Parsing of Morphologically Ricj Languages (SPMRL) What, How and Wither. In Proc of *NAACL-HLT 2010 workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2010), Los Angeles, CA.*

R. Tsarfaty and K. Sima'an. 2008. Relational-Realizational Parsing. *Proceedings of the 22nd CoLing. Manchester, UK.*

Team	*Average*	*Fine-grained*			*Coarse-grained*		
		Hindi	*Bangla*	*Telugu*	*Hindi*	*Bangla*	*Telugu*
Kosaraju et al.	76.83	88.63	70.55	70.12	88.87	73.67	69.12
Kolachina et al.	76.42	86.22	70.14	68.11	88.96	75.65	69.45
Attardi et al.	75.80	87.49	70.66	65.61	88.98	74.61	67.45
Abhilash and Mannem	72.59	82.18	65.97	66.94	83.41	69.09	67.95
Ghosh et al.	--		64.31				
Kesedi et al.	--						48.08

Table 4a: Labeled Attachment Score (LAS) of all the systems in the ICON 2010 Tools Contest.

Team	*Average*	*Fine-grained*			*Coarse-grained*		
		Hindi	*Bangla*	*Telugu*	*Hindi*	*Bangla*	*Telugu*
Attardi et al.	90.94	94.78	87.41	90.48	94.57	88.24	90.15
Kolachina et al.	90.46	93.25	87.10	90.15	94.13	88.14	89.98
Kosaraju et al.	90.30	94.54	86.16	91.82	93.62	86.16	89.48
Abhilash and Mannem	86.63	89.62	83.45	86.81	89.62	83.45	86.81
Ghosh et al.	--		83.87				
Kesedi et al.	--						76.29

Table 4b: Unlabeled Attachment Score (UAS) of all the systems in the ICON 2010 Tools Contest.

Team	*Average*	*Fine-grained*			*Coarse-grained*		
		Hindi	*Bangla*	*Telugu*	*Hindi*	*Bangla*	*Telugu*
Kosaraju et al.	79.14	90.00	73.36	71.95	90.79	77.42	71.29
Kolachina et al.	78.70	87.95	73.26	70.12	90.91	78.67	71.29
Attardi et al.	77.80	88.96	73.47	66.94	90.73	77.73	68.95
Abhilash and Mannem	75.37	84.02	69.51	69.62	85.29	73.15	70.62
Ghosh et al.	--		69.30				
Kesedi et al.	--						50.25

Table 4c: Label Score (LS) of all the systems in the ICON 2010 Tools Contest.

Appendix – I: Dependency labels

Tag Name	Tag description	Coarse-grain tag
k1	karta (doer/agent/subject)	k1
pk1	prayojaka karta (Causer)	k1
jk1	prayojya karta (causee)	vmod
mk1	madhyastha karta (mediator-causer)	vmod
k1g	gauna karta (secondary karta)	vmod
k1s	vidheya karta (karta samanadhikarana)	k1s
k2	karma (object/patient)	k2
k2p	Goal, Destination	k2p
k2g	gauna karma (secondary karma)	vmod
k2s	karma samanadhikarana (object complement)	k2s
k3	karana (instrument)	k3
k4	sampradaana (recipient)	k4
k4a	anubhava karta (Experiencer)	k4a
k5	apaadaana (source)	k5
k5prk	prakruti apadana (‘source material' in verbs denoting change of state)	vmod
k7t	kaalaadhikarana (location in time)	k7
k7p	deshadhikarana (location in space)	k7
k7	vishayaadhikarana (location abstract)	k7
k*u	saadrishya (similarity)	vmod
k*s	samanadhikarana (complement)	vmod
r6	shashthi (possessive)	r6
r6-k1, r6-k2	*karta* or *karma* of a conjunct verb (complex predicate)	r6-k1, r6-k2
r6v	('kA' relation between a noun and a verb)	vmod
adv	kriyaavisheshana ('manner adverbs' only)	vmod
sent-adv	Sentential Adverbs	vmod
rd	prati (direction)	vmod
rh	hetu (cause-effect)	rh
rt	taadarthya (purpose)	rt

ras-k*	upapada__ sahakaarakatwa (associative)	vmod
ras-neg	Negation in Associatives	vmod
rs	relation samanadhikaran (noun elaboration)	rs
rsp	relation for duratives	nmod
rad	Address words	vmod
nmod__relc, jjmod__relc, rbmod__relc	Relative clauses, jo-vo constructions	relc
nmod__*inv		nmod
nmod	Noun modifier (including participles)	nmod
vmod	Verb modifier	vmod
jjmod	Modifiers of the adjectives	jjmod
rbmod	Modiiers of adverbs	rbmod
pof	Part of relation	pof
ccof	Conjunct of relation	ccof
fragof	Fragment of	fragof
enm	Enumerator	vmod
nmod__adj	adjectival modifications	nmod__adj
lwg__psp	noun and post-position/suffix modification	lwg__psp
lwg__neg	NEG and verb/noun modification	lwg__neg
lwg__vaux	Auxiliary verb modification	lwg__vaux
lwg__rp	particle modification	lwg__rp
lwg_cont	lwg continuation relation	lwg_cont
lwg__*	Other modifications in lwg	lwg__rest
jjmod__intf	intensifier adjectival modifica-tions.	jjmod__intf
pof__redup	reduplication	pof__redup
pof__cn	compound noun	pof__cn
pof__cv	compound verb	pof__cv
rsym	Punctuations and symbols	rsym
mod	Modifier	mod

Bidirectional Dependency Parser for Indian Languages

Aswarth Abhilash and Prashanth Mannem
Language Technologies Research Center
International Institute of Information Technology
Hyderabad, AP 500032, India
{abhilash.d,prashanth}@research.iiit.ac.in

Abstract

In this paper, we apply bidirectional dependency parsing algorithm for parsing Indian languages such as Hindi, Bangla and Telugu as part of NLP Tools Contest, ICON 2010. The parser builds the dependency tree incrementally with the two operations namely proj and non-proj. The complete dependency tree given by the unlabeled parser is used by SVM (Support Vector Machines) classifier for labeling. The system achieved Labeled Attachment Score (LAS) of 84.79%, 69.09%, 68.95% for Hindi, Bangla and Telugu. While using fine-grained dependency labels, it achieved LAS of 83.12%, 65.97%, 67.45% respectively.

1 Introduction

The task of NLP tools contest in ICON 2009 (Husian, 2009) is to do dependency parsing for Indian languages Hindi, Bangla and Hindi given the training, development and testing datasets. Various teams had participated in it. Ambati et al (2009), Nivre (2009a) used the transition-based Malt Parser (2007b) for their experiments. They explored various features with various configuration settings in it. Zeman (2009) combined the output of the various well-known dependency parsers into a superparser by using a simple voting method to perform the task. Mannem (2009b) used the bi-directional dependency parsing algorithm proposed by Shen and Joshi (2008). It does a best first search over the sentence and picks the most confident dependency relation between a pair of words every time. It builds the tree incrementally without following a specific direction (either from left-to-right or right-to-left). The search can start at any position and expands the partial analyses in any direction. A slight variant of this approach has been proposed by Goldberg and Elhadad (2010). It can be viewed as a special case of Shen and Joshi (2008) with beam search one. This year also, the same task has been put up by the organizers with the larger annotated data than in 2009.

In this work, we apply this bidirectional parsing algorithm by using two actions to build the dependency tree for a sentence. The two actions distinguish nodes that have an outgoing non-projective arc and those that don't. Once the complete dependency tree is generated by the unlabeled parser, we used LIBSVM tools (Wu et al., 2004) for labeling. Our system achieved Labeled Attachment Score of 83.12%, 65.97% and 67.45% for Hindi, Telugu and Bangla respectively with fine-grained dependency labels.

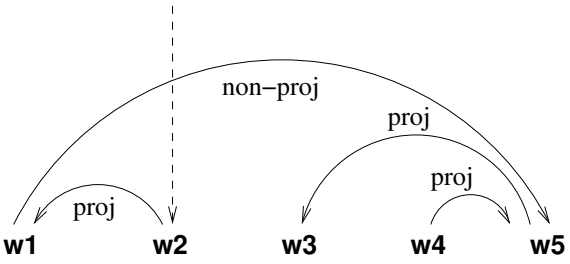


Figure 1: Example of dependency tree built using the two actions *proj* and *non-proj*

2 Our approach

The relatively free word order property in Indian languages indicate the presence of non-projective dependencies in them. A relation is non-projective if the yield/projection of the parent in a relation

is not contiguous. 18.75% of Hindi sentences in the released treebank are non-projective. Some of the sentences have multiple non-projective arcs within a sentence. For Bangla, it is 6.02%. Telugu treebank has very few non-projective structures (0.82%) due to the kind of corpus chosen for treebank annotation. The average sentence lengths in the Hindi, Bangla and Telugu treebanks are 21.92 words, 6.42 words and 3.94 words respectively. The smaller sentence length in Telugu explains the low occurrence of non-projectivity in Telugu treebank.

Non-projectivity has been handled in incremental parsers earlier. Nivre and Nilsson (2005) used a pseudo-projective parsing technique to retrieve non-projective dependencies as a post processing step in a transition based parser. Nivre (2009b) proposed a transition system which can parse arbitrary non-projective trees by swapping the order of words in the input while parsing.

In our approach to dependency parsing, we use two different operations to connect nodes that have outgoing non-projective arcs and those that don't. A dependency tree is built by connecting nodes with two operations *proj* and *non-proj*. The operation *non-proj* is used to connect the head in a non-projective relation to its parent. *proj* is used to connect all the other nodes in a dependency tree to their parents. In the example dependency tree given in Figure 1, there is a non-projective arc from w1 to w5 connected through *non-proj* operation. Nodes connected through *non-proj* indicate scope for an outgoing non-projective arc from them. During parsing, the set of candidate parents for a node will include all the nodes which satisfy projectivity and also those nodes which have been connected through a *non-proj* operation.

In this work, we apply the bidirectional parsing approach described in Shen and Joshi (2008) for Indian Languages such as Hindi, Bangla and Telugu using the two operations defined above. The original implementation of the parsing algorithm by Libin Shen performed unlabeled LTAG dependency parsing by training on LTAG-spinal treebank. Later is has been extended to do labeled dependency parsing by Mannem (2009b). Since the size of the annotated treebank is considerably not large and due to the presence of more dependency labels (>30), the labeled bidirectional dependency parsing algorithm extended by Mannem (2009b) doesnot perform well. So, we used LIBSVM tools (Wu et al., 2004) for labeling.

Mannem et al. (2009a) extracted LTAG-spinal treebank for Hindi from a dependency treebank by using the *adjunction* operation to handle non-projective structures and *attachment* to handle projective relations other than coordination structures. They trained the bidirectional LTAG dependency parser (Shen and Joshi, 2008) on the extracted Hindi LTAG-spinal treebank with good results.

3 Unlabeled Bidirectional Dependency Parser (UBDP)

Shen and Joshi (2008) proposed a bidirectional incremental parsing algorithm which searches the sentence for the best score dependency hypothesis in both the directions (left-to-right & right-to-left). The search can begin at any position and can expand the intermediate results in any direction. The order of search is learned automatically.

The implementation of this parsing framework by Libin Shen could only do unlabeled LTAG dependency parsing and the training data was LTAG-spinal treebank in LTAG-spinal format. We use the same algorithm by doing minor changes to it.

In the rest of the section, we give an overview of the parsing algorithm along with the training and inference processes presented in (Shen and Joshi, 2008). We mention our extensions to the original parsing framework whereever appropriate.

3.1 Parsing Algorithm

We are given a linear graph $G = (V, E)$ with vertices $V = \{v_i\}$ and $E(v_{i-1}, v_i)$ with a hidden structure $U = \{u_k\}$. The hidden structure $u_k = (v_{s_k}, v_{e_k}, l_k)$, where vertex v_{e_k} depends on vertex v_{s_k} with label $l_k \, \epsilon \, L$ is what we want to find (the parse tree). L is the full list of dependency labels occurring in the corpus. A sentence is a linear graph with an edge between the adjacent words. A **fragment** is a connected sub-graph of $G(V, E)$. Each fragment x is associated with a set of hypothesized hidden structures, or **fragment hypotheses** for short: $Y^x = \{y_1^x, \cdots, y_2^x\}$. Each y^x is a possible fragment hypothesis of x. A fragment hypothesis represents a possible parse analysis for a fragment. Initially, each word with its POS tag comprises a **fragment**.

Let x_i and x_j be two fragments, where $x_i \cap x_j = \emptyset$ and are directly connected via an edge in E. Let y^{x_i} be a fragment hypothesis of x_i , and y^{x_j} a fragment hypothesis of x_j. We can combine the hy-

potheses for two nearby fragments with one of the labels from L. Suppose we choose to combine y^{x_i} and y^{x_j} with an operation $R_{type,label,dir}$ to build a fragment hypothesis for $x_k = x_i \cup x_j$. The output of the operation is

$$y^{x_k} = R_{type,label,dir}(x_i, x_j, y^{x_i}, y^{x_j}) \supseteq y^{x_i} \cup y^{x_j}$$

where $type \in$ {*proj*, *non-proj*} is the type of operation, $label$ is the dependency label from L and $dir \in \{left, right\}$, representing whether the left or the right fragment is the parent. y^{x_i} and y^{x_j} stand for the fragment hypotheses of the left and right fragments x_i and x_j.

An operation R on fragment hypotheses $R.y^{x_i}$ and $R.y^{x_j}$ generates a new hypotheses $y(R)$ for the new fragment which contains both the fragments $R.x_i$ and $R.x_j$. The score of an operation is defined as

$$s(R) = W.\phi(R)$$

where $s(R)$ is the score of the operation R, which is calculated as the dot product of a weight vector W and $\phi(R)$, the feature vector of R. The score of the new hypothesis is the sum of scores of the operation and the involving fragment hypotheses.

$$score(y(R)) = s(R) + score(R.y^{x_i}) + score(R.y^{x_j})$$

The feature vector $\phi(R)$ is defined on $R.y^{x_i}$ and $R.y^{x_j}$, as well as the context hypotheses. If $\phi(R)$ only contains information in $R.y^{x_i}$ and $R.y^{x_j}$, its called **level-0 feature dependency**. If features contain information of the hypotheses of nearby fragments, its called **level-1 feature dependency**. A **chain**, is used to represent a set of fragments, such that hypotheses of each fragment always have feature dependency relations with some other fragments within the same chain. Furthermore, each fragment can only belong to one chain. A set of related fragment hypotheses is called a **chain hypothesis**. For a given chain, each fragment contributes a fragment hypothesis to build a chain hypothesis. Beam search is used with a predefined beam width to keep the top k chain hypotheses for each chain. The score of a chain hypothesis is the sum of the scores of the fragment hypotheses in this chain hypothesis. For chain hypothesis c,

$$score(c) = \sum_{fragment\, hypothesis\, y^x\, of\, c} score(y^x)$$

A **cut** T of a given sentence, $T = \{c_1, c_2, \cdots, c_m\}$, is a set of chains satisfying

- exclusiveness: $\cup c_i \cap \cup c_j = \emptyset, \forall i, j$, and
- completeness:$\cup(\cap T) = V$.

Furthermore, $H^T = \{H^{c_i} | c_i \in T\}$ is used to represent the sets of fragment hypotheses for all the fragments in cut T. At every point in the parsing process, a priority queue of candidate operations Q is maintained. Q contains all the possible operations for the fragments and their hypotheses in cut T. $s(R)$ is used to order the operations in Q.

With the above formal notations, we now list the inference and learning algorithms in Algorithm 1 and Algorithm 2.

Algorithm 1 UBDP: Inference Algorithm

INPUT: graph $G(V, E)$ and weight vector **W**;
INITIATE cut T, hypotheses H^T, queue of candidate operations Q;
while Q is not empty **do**
  operation $y \leftarrow \arg_{op \in Q}$ max score (op, **W**);
  UPDATE T, H^T, Q with y ;
end while

3.2 Decoding

Algorithm 1 describes the procedure of building hypotheses incrementally on a given linear graph $G = (V, E)$. Parameter k is used to set the beam width of search. Weight vector w is used to compute the score of an operation. First, the cut T is initiated by treating each vertex in V as a fragment and a chain. Then the initial hypotheses for each vertex/fragment/chain are set with the POS tag for each word. The priority queue Q is used to collect all the possible operations over the initial cut T and hypotheses H^T. Whenever Q is not empty, the chain hypothesis with highest score on operation according to a given weight vector w is searched for and the cut along with its hypotheses are updated with that chain hypothesis. The candidate queue Q is then updated by removing operations depending on the chain hypotheses that have been removed from H^T , and adding new operations depending on those chain hypotheses.

3.3 Training

For each given training sample (G_r, H_r), where H_r is the gold standard hidden structure of graph G_r, cut T , its hypotheses H^T and candidate queue Q are initialized. The gold standard H_r is used to guide the searchi. The candidate (x, y) with the highest operation score in Q is selected. If y' is

Algorithm 2 UBDP: Training Algorithm

```
W ← 0;
for round = 1..T, i = 1..n do
    LOAD graph G_r(V, E), hidden structure H_r;
    Initiate cut T, hypotheses H^T, queue Q;
    while Q is not empty do
        operation y ← arg_{op ϵ Q} max score (op, W);
        if compatible(H_r,y) then
            Update T, H^T, Q with y ;
        else
            y* ← searchCompatible(Q,y);
            promote(y*)
            demote(y)
            UPDATE Q with W;
        end if
    end while
end for
```

compatible with H_r, the cut T , hypotheses H^T and Q are updated. If y' is incompatible with H_r , y' is treated as a negative sample, and a positive sample $y*$ compatible with H_r is searched for in Q. If such compatible hypothesis doesn't exist, the hypothesis with highest score in Q and compatible with H_r is searched. Then, the weight vector w is updated with with y and $y*$. At the end, the candidate queue Q is updated with the new weights w to compute the score of operation. Perceptron learning with margin is used in the training and averaged Perceptron for inference. Algorithm 2 lists the training procedure.

3.4 UBDP + SVM classifier

The unlabeled dependency tree given by UBDP is used by SVM (Support Vector Machines) classifier for labeling. We have performed experiments using various features with different options provided by SVM tool.

The best accuracy for Telugu and Bangla are obtained using the feature set described in Table 1 $.cnk$ denotes the chunk label. $.litem$ denotes the word/root form. $.afx$ denotes the affix information. $.pa$ denotes the head word and $.ch$ dentoes the dependent. Apart from the features mentioned above, the features like number of children, number of siblings, difference in positions of the child and parent and pos list from the root of a sentence to the dependent are also considered. For Hindi, we have also added structural features involving part of speech tags of all the words involved in the partial trees while building the tree incrementally. Due to the availability of smaller treebanks for both Telugu and Bangla, adding these structural features didn't help in improving the accuracy.

(a) Unigram features involving pa and ch	
$pa.litem$	$ch.litem$
$pa.pos$	$ch.pos$
$pa.cnk$	$ch.cnk$
$pa.afx$	$ch.afx$
($pa.litem$ + $pa.pos$)	($ch.litem$ + $ch.pos$)
($pa.litem$ + $pa.cnk$)	($ch.litem$ + $ch.cnk$)
($pa.litem$ + $pa.afx$)	($ch.litem$ + $ch.afx$)

(b) Bigram features involving pa and ch
($pa.litem$ + $ch.litem$)
($pa.litem$ + $pa.pos$ + $ch.litem$ + $ch.pos$)
($pa.litem$ + $pa.afx$ + $ch.litem$ + $ch.afx$)
($pa.pos$ + $pa.afx$ + $ch.pos$ + $ch.afx$)
($pa.cnk$ + $pa.afx$ + $ch.cnk$ + $ch.afx$)
($pa.litem$+$pa.afx$+$pa.pos$+$ch.litem$+$ch.afx$+$ch.pos$)
($pa.litem$+$pa.afx$+$pa.cnk$+$ch.litem$+$ch.afx$+$ch.cnk$)

(c) Features involving all the siblings sb of ch
($pa.pos$ + $ch.pos$ + $sb.pos$)
($pa.litem$ + $pa.pos$ + $ch.pos$ + $sb.pos$)
($pa.litem$ + $pa.afx$ + $ch.afx$ + $sb.pos$)
($pa.pos$ + $pa.afx$ + $ch.pos$ + $ch.afx$ + $sb.pos$)

(d) Features involving the context words $b.i$ **i could range from -2 to +2**
($pa.pos$ + $ch.pos$ + $b.i.pos$)
($pa.pos$ + $ch.pos$ + $b.i.cnk$)
($pa.pos$ + $ch.pos$ + $b.i.afx$)
($pa.litem$ + $pa.pos$ + $ch.pos$ + $b.i.pos$)
($pa.litem$ + $pa.pos$ + $ch.pos$ + $b.i.cnk$)
($pa.litem$ + $pa.pos$ + $ch.pos$ + $b.i.afx$)

Table 1: Features used by the parser for all the three languages

4 Data

Annotated training and development data for Hindi, Telugu and Bangla were released as part of the contest (ICON, 2010). Some sentences have been discarded due to the presence of errors in the treebank. The final data (training+development) contained 3515 Hindi, 1450 Telugu and 1129 Bangla sentences. 68 Bangla (6.02%), 659 (18.75%) Hindi and 12 (0.82%) Telugu sentences were non-projective in the entire corpus.

For the testing phase of the contest, the parser was trained on the entire released data with the best performing feature set and the unannotated test data was parsed with the model obtained.

5 Experiments and Results

The Unlabeled Bidirectional Dependency Parser (UBDP) with the features listed in Table 1 was used for developing models for the three languages. This feature set was arrived at by doing a 5-fold cross validation on the released data (training+development) for Hindi. Sentence context (SC) and local context (LC) were varied over a range of 0 to +/-3. The best accuracy over the 5-fold cross validation was reported with a SC of 2 and LC of 1. This feature set was used for Telugu and Bangla for the accuracies reported in this section. For Hindi, combining more structural features as mentioned in the previous section with this feature set performed well. The results shown here are different from the ones submitted. This is due to the selection of incorrect models for testing while submitting the outputs.

The annotated data for all the three languages was released with fine-grained and coarse-grained dependency labels separately by the organizers. The unannotated test data released by the organizers had 321 sentences for Hindi and 150 sentences for Bangla and Telugu. The accuracies achieved by our system on the data with fine-grained dependency labels are in Table 2. UAS is the Unlabeled Attachment Score, LAS is the Labeled Attachment Score and LA is the Labeled Accuracy. UAS, LAS and LA are standard evaluation metrics in the literature for dependency parsing (Nivre et al., 2007a).

On fine-grained tagset			
Language	*UAS*	*LAS*	*LA*
Hindi	90.97%	83.12%	84.99%
Bangla	83.45%	65.97%	69.51%
Telugu	89.65%	67.45%	69.78%
Average	88.02%	72.18%	74.76%

Table 2: Accuracies on data annotated with fine-grained dependency labels

Though the available annotated data for all the three languages is small, the average UAS (88.02%) is high. One reason is because of the use of gold-standard chunks and part of speech tags during training and testing. The other reason is due to the presence of large number of intra chunk relations when compared to inter chunk relations. The average LAS is however considerably low at 72.18% when compared to the avg. UAS (88.02%). This is because of the small size of the available treebanks and also due to the large number of dependency labels used in the annotation (>30). The dependency labels used in the treebanks are syntactico-semantic in nature, marking labels is even more difficult than the one with pure syntactic labels .

The organizers also released treebanks for the three languages with coarse-grained dependency labels. Table 3 shows the performance of the parser on this data. The avg. LAS has increased to 77.06% because of the reduction in the number of dependency labels.

On coarse-grained tagset			
Language	*UAS*	*LAS*	*LA*
Hindi	90.97%	84.79%	86.60%
Bangla	83.45%	69.09%	73.15%
Telugu	89.65%	68.95%	71.45%
Average	88.02%	74.28%	77.06%

Table 3: Accuracies on data annotated with coarse-grained dependency labels

6 Error Analysis

For all languages, in case of labeled accuracy, the classifier is not able to predict the correct label among the labels k1, k2 and pof. It is due to the wrong dependencies produced by the unlabeled parser and also due to the absence of specific case markers which makes hard to disambiguate k1 and k2. While in case of unlabeled parsing, the frequent errors are the inter chunk relations involving nouns and conjuncts. For Bangla and Telugu, lower labeled accuracy is due to the availability of smaller treebank and the classifier doesn't have enough samples to learn properly. Since the number of dependency labels are large, the number of hypotheses generated are even larger which allows the classifier to take an improper decision.

7 Conclusion and Future Work

In this work, we used the approach to dependency parsing which builds the tree using two actions *proj* and *non-proj*. *non-proj* is used to connect nodes that lead to non-projective arcs. A node connected with this operation is available/visible

for combination beyond its projective scope. *proj* is used to connect the rest of the nodes. A bidirectional dependency parsing algorithm is used with these two operations for our system. The unlabeled dependency tree generated is used by SVM (Support Vector Machines) classifier for labeling. The parser was trained on the entire released data and we got LAS of 84.79%, 69.09% and 68.95% for Hindi, Bangla and Telugu respectively for the coarse-grained dependency tagset. While using fine-grained dependency labels, the LAS for the three languages are 83.12%, 65.97% and 67.45% respectively. Our immediate future work consists of experiments for proper selection of features and also extending bidirectional parser to provide features easily.

References

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 742–750, Morristown, NJ, USA. Association for Computational Linguistics.

S. Husian. 2009. Dependency parsers for indian languages. In *ICON09 NLP Tools Contest*.

ICON. 2010. Nlp tools contest 2010.

K Jindal, P. Gadde, and B.R Ambati. 2009. Experiments in indian language dependency parsing. In *ICON09 NLP Tools Contest*, pages 32–37.

Prashanth Mannem, Aswarth Abhilash, and Akshar Bharati. 2009a. Ltag-spinal treebank and parser for hindi. In *Proceedings of the International Conference on NLP (ICON)*, Hyderabad, India.

Prashanth Mannem, Himani Chaudhry, and Akshar Bharati. 2009b. Insights into non-projectivity in hindi. In *Proceedings of the ACL-IJCNLP 2009 Student Research Workshop*, pages 10–17, Suntec, Singapore, August. Association for Computational Linguistics.

J. Nivre and J. 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, 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 Session of EMNLP-CoNLL 2007*, pages 915–932, Prague, Czech Republic. Association for Computational Linguistics.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Glsen Eryigit, Sandra Kbler, Svetoslav Marinov, and Erwin Marsi. 2007b. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

J. Nivre. 2009a. Parsing indian languages with maltparser. In *ICON09 NLP Tools Contest*, pages 11–17.

Joakim Nivre. 2009b. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, Suntec, Singapore.

Libin Shen and Aravind Joshi. 2008. LTAG dependency parsing with bidirectional incremental construction. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 495–504, Honolulu, Hawaii, October. Association for Computational Linguistics.

Ting-Fan Wu, Chih-Jen Lin, and Ruby C. Weng. 2004. Probability estimates for multi-class classification by pairwise coupling. *J. Mach. Learn. Res.*, 5:975–1005, December.

D. Zeman. 2009. Maximum spanning malt: Hiring world's leading dependency parsers to plant indian trees. In *ICON09 NLP Tools Contest*, pages 18–23.

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 infor-

mation. 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.

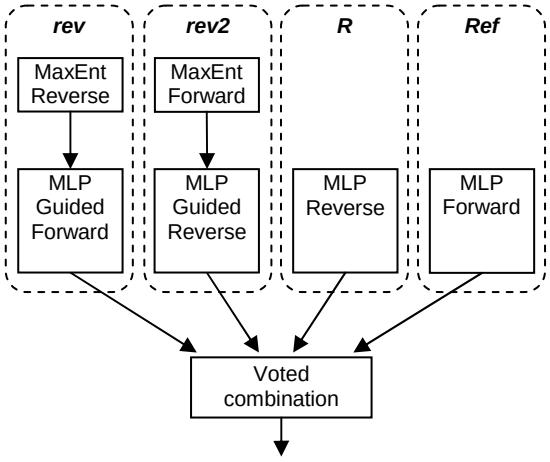


Figure 1. Parser combination schema

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).

Bengali Parsing System at ICON NLP Tool Contest 2010

Aniruddha Ghosh[†] **Amitava Das**[*] **Pinaki Bhaskar**[+] **Sivaji Bandyopadhyay**[¥]
Department of Computer Science and Engineering
Jadavpur University
Jadavpur, Kolkata 700032, India
arghyaonline@gmail.com[†] amitava.santu@gmail.com[*] pinaki.bhaskar@gmail.com[+] sivaji_cse_ju@yahoo.com[¥]

Abstract

This paper reports about the development of a Bengali Dependency Parser as a part in the ICON 2010 NLP TOOLS CONTEST: IL Dependency Parsing. Due to syntactic richness of Bengali language, hybrid architecture has been proposed for the problem domain. A statistical data driven parsing system (maltparser) has been used followed by a rule-based post-processing technique. The system has been trained on the ICON 2010 NLP TOOLS CONTEST: IL Dependency Parsing at ICON 2010 datasets. The system demonstrated an accuracy of 83.87% 64.31% 69.3% respectively over fine-grained tagset.

1 Introduction

Bengali language is characterized by a rich system of inflections (VIBHAKTI), derivation, and compound formation (Saha et al., 2004; Chakroborty, 2003) and ***karakas***, which is why the Natural Language Engineering (NLE) for Bengali is a very challenging task. Naturally these language specific peculiarities involve parsing natural language sentences considerable problems. Therefore, developing a computational grammar for a natural language can be a complicated endeavor. Grammar development, also known as grammar engineering, is a formidable challenging task.

Previous research proposed two different approaches context of parsing of a sentence. These techniques are known as grammar driven parsing and data driven parsing. Most of the previous grammar driven research attempts was for detection and formation of the proper rule set to identify characteristics of relations among inter-chunk relations.

Natural languages are very diverse in nature, no matter how many sentences a person has heard and taking in consideration during the rule making, new ones can always be produced. Even when people speak natural language incorrectly i.e., not strictly in accordance with rules of grammar and syntax, anyone can still make sense out of it. Hence developing a parsing rule book always will remain inadequate. Most of the modern grammar-driven dependency parsers (Karlsson et al.; 1995, Bharati et al., 2008) parse by eliminating the parses which do not satisfy the given set of constraints. They require rules to be developed for each layer. It is the reason to start with a data driven parsing system (Maltparser[1] ver.1.3.1) as a baseline. The data driven parser requires a large set of manually annotated corpus. But the available dataset is not large in size and thus only data driven cannot outperform in this experimental setup. We propose a hybrid technique and the output of the baseline system then filtered by a rule-based post-processing system.

2 Dataset

The ICON 2010 NLP TOOLS CONTEST: IL Dependency Parsing at ICON 2010 datasets was provided with fine-grain and coarse-grain tagset. The corpus statistics is reported in the Table 1. A few detailed statistics about the distribution of sentence type is reported in table 2.

Corpus	s#	t#	t#/s#
Training	960	7269	7.57
Development	150	812	5.41
Testing	150	962	6.4

Table 1: Corpus statistics; s# = number of sentence; t# = number of tokens; t#/s# = number of tokens per sentence.

Corpus	simple	compound	complex
Training	223	188	589
Development	31	11	108
Testing	26	7	117

Table 2: Corpus sentence statistics

[1] http://maltparser.org/download.html

3 The Dependency parsing system

3.1 The Maltparser

Malt is a classifier based Shift/Reduce parsing methodology. It uses arc-eager, arc-standard, covington projective and convington non-projective algorithms for parsing (Nivre, 2006). History-based feature models are used for predicting the next parser action (Black et al., 1992). Support vector machines are used for mapping histories to parser actions (Kudo and Matsumoto, 2002). It uses graph transformation to handle non-projective trees (Nivre and Nilsson, 2005).

Maltparser accepts CoNLL format as the input. The FEATS column in the training and developing corpus set, there are 10 attribute fields in a node. Among the attributes, six morphological features namely lexicon, category, gender, number, person, vibhakti or Tense-Aspect-modality (TAM) markers of the node are considered in the experiment. After experimentation with set of different combination of these features, the vibhakti, TAM and the morphological category produce better results as these morphological informations contains most crucial information to identify dependency relations for Indian languages. The Bengali datasets consists of 7% non-projective sentences. Among the four parsing algorithm provided with maltparser, we found that nivreeager works best for the Bengali corpus with fine-grained tagset. Analyzing output of parsing with default setting, we found the complex and compound sentences are the most error prone. We tried to separate compound sentences and compound sentences into simple sentences. We calculated the average number of tokens per sentence is around 6. Thus the max sentence length was set to 6. Malt parser provides root handling which specifies how dependents of the special root node are handled. After experimentation with root handling, the relaxed root handling yields better results. As the learning method maltparser uses SVM with polynomial kernel to map feature vector-representation of a parser configuration. While tuning the learning method parameter, we changed the cost parameter from default value 1 to .65, which controls the tradeoff between minimizing training error and maximizing margin. From corpus statistics we revealed that the average number of instances i.e. tokens per sentence is near about 6 and the number of attribute per node i.e. chunk is 9. Thus, we also experimented with the liblinear classifier. Due to small set of dataset, many of the dependency relations are sparsely distributed, which leads to low LAS value. The comparative study of accuracies of different maltparser configurations are shown below.

Algorithm	UAS[2]	LAS[3]	LS[4]
Nivreea-ger+Liblinear	**81.64%**	**54.58%**	**50.62%**
Convington non-projec-tive+LIBSVM	78.22%	51.02%	48.43%
Convington non-projective+ Liblinear	79.33%	52.47%	50.52%

Table 3: comparison of maltparser output with different settings

During the development process confusion matrix helped to detect errors. Most prominent errors on the development set are shown in Table 4.

	k1	k2	k7p	k7t	pof	vmod
k1	0	29	3	1	6	1
k2	7	0	0	0	6	9
k7p	12	6	0	3	1	2
k7t	3	2	0	0	1	4
pof	2	3	0	0	0	0
vmod	4	7	1	2	0	0

Table 4: Confusion Matrix on Development Set1

3.2 Post-Processing

With the help of confusion matrix, depending on the nature of errors we have devised a set of parsing rules. A few rules have been devised to produce more dependency relation variations.

Vibhakti plays a crucial role in dependency relation identification. In corpus, as the vibhakti information is missing in some cases, a suffix analyzer is applied to the word.

r6: According to dependency tagset, r6 denotes genitive relation. It takes 'ra', 'era', 'xera' genetive markers. For the marker 'ra', it can appear at the end of many words, e.g. 'AbiskAra'. We have used a dictionary based approach to exclude these words. When chunks with these genitive markers have any indirect relation with the main verb, then it is marked with r6 relation

[2] UAS – Unlabeled Attachment Score
[3] LAS – Labeled Attachment Score
[4] LS - Labeled Score

and following NP chunks is marked the related chunks.

k7t: Chunks with suffix 'kAle' denotes a time temporal. We have also, from the training corpus, developed list of time temporal. After a manual check of the generated list, we used the list to extract time temporal and marked with k7t.

k7p: Chunks with suffix 'pur' denotes a space temporal. We have also, from the training corpus, developed list of space temporal. After a manual check of the generated list, we used the list to identify space temporal and marked with k7p.

After an in depth study of the errors made by maltparser, a rule based system has been developed with the help of linguistic knowledge. Depending of specific attributes of the chunk like vibhakti/case markers and/or word information, the rule based system derives the dependency relation of the chunk. For each dependency relation tag depending on specific linguistic features, syntactic cues are derived identify the dependency relations. As example:

1. A NP chunk with 0 vibhakti and NNP or PRP postag will be marked with k1 relation with the nearest verb chunk.
2. A chunk with "era" vibhakti will be marked with 'r6' relation with next noun chunk.
3. A NP chunk with 0 vibhakti and NN postag will be marked with k2 nearest verb chunk.
4. In co-ordinate type sentences, the verb chunk will be marked with 'ccof' relation with the nearest CCP chunk. If CCP chunk is surrounded by two NP chunk then NP chunk will be marked 'ccof' with the CCP.
5. Sub-ordinate sentences are identified using the presence of keywords like 'ye' etc. In a sub-ordinate type sentences, the verb chunk of sub-ordinate clause will be marked with "nmod__relc" with that chunk of main clause, which the clause is modifying.
6. If a chunk with "0_weke" vibhakti, k5 relation will be indentified. Both of these relation is pre-dependent i.e. a dependent that precedes its head.
7. If a chunk has "0_prawi" vibhakti, dependency relation will be 'rd'.
8. After carefully analyzing training corpus, we found certain vibhakti with semantic meanings like '0_pakRe', '0_hisAbe' etc. can be treated as cue to mark vmod relation.
9. Verb like 'kar' or 'ha' often takes another argument to form compound verbs. The argument is marked with part-of relation (pof). The preceding noun or verb chunk, if it has no suffix, is marked with 'pof' relation.
10. If a chunk marked with "ke", 'k2' relation will be indentified.
11. Noun chunks with rootwords like "ami" with "NN" postag or "wumi" with "PRP" postag will be marked with 'k1'.
12. If the rootword is 'ye' and the word is 'ya'. 'wa', 'ye', then the chunk will be marked with 'k2' relation.

The ambiguity comes when for a certain vibhakti, multiple possible relations are identified. As example, when a chunk with "0" vibhakti two possible dependency output relation is 'k1' & 'k2'. Then this ambiguity is resolved using postag. If postag is 'NNP' then dependency relation will be 'k1' and if 'NN' then it will be 'k2'. If ambiguity is not resolved with this rule then position of the chunk in the sentence is considered. If there are two chunks with "0" vibhakti, the distant chunk from the verb chunk will be marked with 'k1' relation and nearer ones will be marked with 'k2' relation.

After a study of co-occurrence of 'k1' & 'k2', we found that single occurrence of noun chunk with '0' vibhakti is marked with 'k1' relation.

The output of the maltparser and the output of the rule based system are compared. The rule based system is given the higher priority as it is based on sytacto-semantic cues. If there is any mismatch between the two results then if rule based system derive an output then output of rule based system has been taken.

4 Error Analysis

During the development stage of the system we have gone through the error prone areas of the system. The system works best with simple sentences.

In the example sentence in Figure 1, CCP chunk is identified as root and it split the sentence and treats ((rAwa jege AsA)) and ((xedIte snAnaKAoyZAra animeRera Alasemi boXa hachila)) as two individual sentences. But semantically it is not correct. Here both the preceding and succeeding VGNF chunk of CCP should be connected with ccof relation. Among the dependency relation tags of the VGNF chunks, one of them will be attached with the CCP chunk. System faces ambiguity to choose proper dependency relation for CCP among the 'rh' and 'vmod'

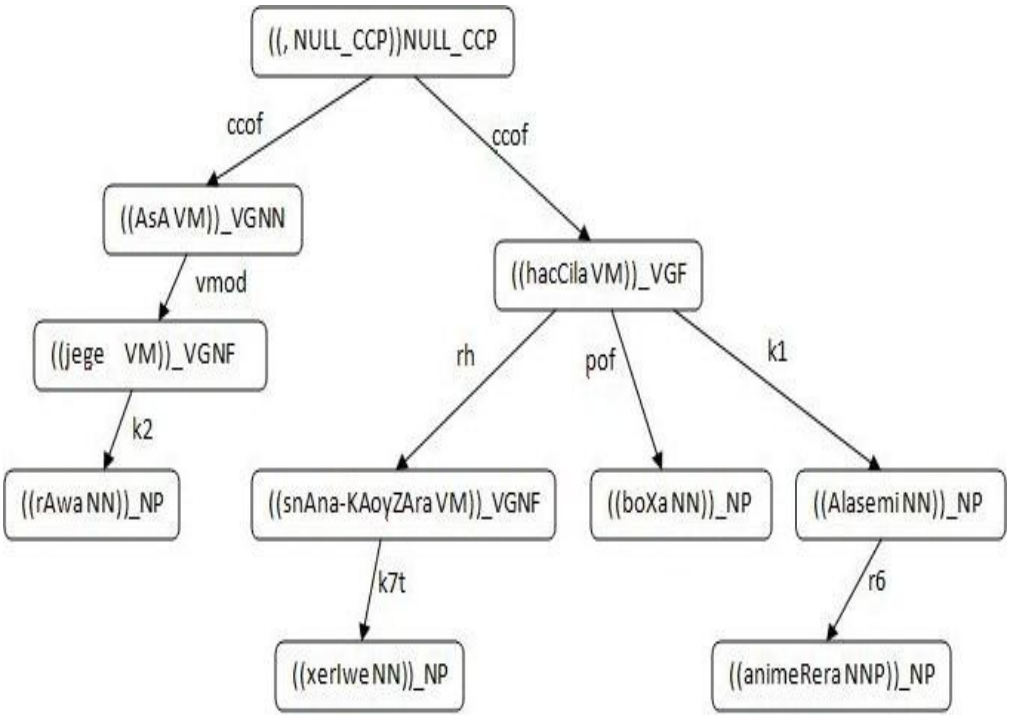


Figure 1: Example Parsing

1 ((rAwa NN))_NP	2	k2	
2 ((jege VM))_VGNF	3	vmod	
3 ((AsA VM))_VGNN	4	ccof	
4 ((, NULL_CCP))NULL_CCP	0	main	
5 ((xerIwe NN))_NP	6	k7t	
6 ((snAna-KAoyZAra VM))_VGNF	10	rh	
7 ((animeRera NNP))_NP	8	r6	
8 ((Alasemi NN))_NP	10	k1	
9 ((boXa NN))_NP	10	pof	
10 ((hacCila VM))_VGF	4	ccof	

4.1 Experimental Results

We have trained maltparser with the training set data with fine-grained tagset only. A brief statistics of the datasets are presented in Table 1 & 2. The maltparser with nivreeager parsing algorithm, yielded unlabelled attachment score of 81.64%, Labeled attachment score of 54.58% and Labeled score of 50.62% on the development set. After the suffix analyzer, we achieved 3% improvement of UAS and 9% improvement of LAS. The LS score yields 69%. Depending upon the nature of errors involved in the results, we have devised a set of rules for different dependency tags. The rule based system yields UAS, LAS, LS 84.02%, 66.63% & 70.82% respectively on the development set.

In the final evaluation, the system demonstrated UAS (Unlabelled Accuracy Score) is 83.87%, LAS (Labeled Accuracy Score) is 64.31% and LS (Labeled Score) is 69.3% respectively.

5 Conclusion

This paper reports on our works as part of the NLP Tool Contest at ICON 2010. We have used the data driven maltparser along with rule based post-processing. Using maltpaser, we have obtained UAS 80%, LAS 54% and LS 59% with the nivreeager algorithm with relaxed root handling. We have used suffix analyzer and a rule-based post-processing and produce a better accuracy value. The rule based system is given the higher priority as it is based on sytacto-semantic cues and using bootstrapping method, performance of data driven parser can be accuracy can be further enhanced.

A properly designed NLP platform for Indian languages must come with an efficient morpho-syntactic unit for parsing words into their constituent morphemes where lexical projections of words can be obtained from projections of individual morphemes. Phrasal order could be vary depending on the corpus. In future task our aim is to develop a more proficient statistical system, can produce more than one possible parsed output. A more concise rule set should be generated with morpho-syntactic and lexico-syntactic variations.

References

Chakroborty, B., 2003. Uchchotoro Bangla Byakaron. AkshayMalancha.

Saha, G.K., Saha, A.B., Debnath, S., 2004. Computer Assisted Bangla POS Tagging. iSTRAN, Tata McGraw-Hill, NewDelhi.

Karlsson, F., A. Voutilainen, J. Heikkilä and A. Anttila,(eds). Constraint Grammar: A languageindependent system for parsing unrestricted text. Mouton de Gruyter. 1995.

Bharati A., Husain S., Ambati B., Jain S., Sharma D.M. and Sangal R. Two semantic features make all the difference in Parsing accuracy. In Proceedings of the 6th International Conference on Natural Language Processing (ICON-08), CDAC Pune, India. 2008.

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

Nivre J., Hall J. and Nilsson J. MaltParser: A Data-Driven Parser-Generator for Dependency Parsing. In Proceedings of the fifth international conference on Language Resources and Evaluation (LREC2006), May 24-26, 2006, Genoa, Italy, pp. 2216-2219

E. Black, F. Jelinek, J. D. Lafferty, D.M. Magerman, R. L. Mercer, and S. Roukos. 1992. Towards history-based grammars: Using richer models for probabilistic parsing. In Proc. of the 5th DARPA Speech and Natural Language Workshop, Pages 31–37.

T. Kudo and Y. Matsumoto. 2002. Japanese dependency analysis using cascaded chunking. In CoNLL-2002. Pages 63–69.

A Two Stage Constraint Based Hybrid Dependency Parser for Telugu

Sruthilaya Reddy Kesidi, Prudhvi Kosaraju, Meher Vijay and Samar Husain
Language Technologies Research Centre, IIIT-Hyderabad, India.
{sruthilaya, prudhvi, meher_vijay}@students.iiit.ac.in,
samar@research.iiit.ac.in

Abstract

In this paper we present a two stage constraint Telugu dependency parsing. We define the two stages and show how different Telugu constructions are parsed at appropriate stages. The division leads to selective identification and resolution of specific dependency relations at the two stages. We then discuss the ranking strategy that makes use of the S-constraints to give us the best parse. A detailed error analysis of the output of core parser and the ranker helps us to flesh out the current issues and future directions.

1 Introduction

There has been a recent surge in addressing parsing for morphologically rich free word order languages such as Czech, Turkish, Hindi, etc. These languages pose various challenges for the task of parsing mainly because the syntactic cues necessary to identify various relations are complex and distributed (Tsarfaty et al., 2010; Ambati et al., 2010; McDonald and Nivre, 2007; Nivre, 2009; Tsafarty and Sima'an, 2008; Seddah et al., 2009; Goldberg and Elhadad, 2009; Husain et al., 2009; Eryigit et al., 2008). Data driven parser performance (Hall et al., 2007) show that many of these complex phenomenons are not being captured by the present parsers. Constraint based parsers have been known to have the power to account for difficult constructions and are well suited for handling complex phenomenons. Some of the recent constraint based systems have been (Karlsson et al., 1995; Maruyama, 1990; Bharati et al., 2002, 1993; Tapanainen, Järvinen, 1997; Schröder, 2002; Martins et al., 2009; and Debusmann et al., 2004).

In this paper we present a two stage constraint based hybrid approach to Telugu dependency parsing based on the parser proposed by Bharati et al., (2008, 2009a, 2009b). We begin by describing how different Telugu constructions are parsed at appropriate stages. We will see that this division leads to selective identification and resolution of specific dependency relations at the two stages. We then discuss the ranking strategy that makes use of the S-constraints to give us the best parse. A detailed error analysis of the output of core parser and the ranker helps us to flesh out the current issues and future directions.

This paper is arranged as follows: section 2 explains in details how we handle different Telugu constructs in two stages using CBHP, section 3 describes the results of the core parser and makes some observations on these results. In section 4 we describe the ranking strategies, followed with results in Section 5. We conclude the paper with future directions in section 6.

2 CBHP for Telugu

Bharati et al., (2009a, 2009b) have proposed a constraint based hybrid parser (CBHP) for Indian languages. It has however been tested only for Hindi. We use the same machinery that was used by them to handle Telugu. We first describe the different types of relations the parser currently handles in the two stages, following which we briefly mention H-constraints for Telugu CBHP.

2.1 Relations handled in Stage 1

Stage 1 handles mainly intra-clausal relations. These relations represent:

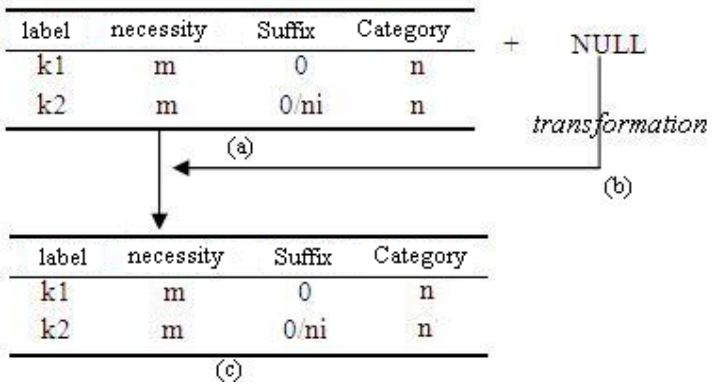
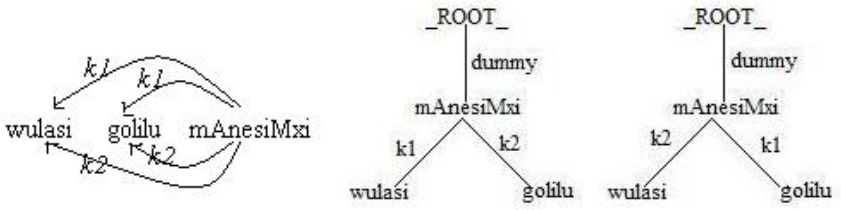


Figure 1. Final demand frame for *mAnesiMxi* shown in (c) is obtained from the basic demand frame of *manu* (a) and the transformation (b) which is NULL here.



(a) Constraint graph for (1). (b) Final parses obtained from CG

Figure 2.

i. Argument structure of the finite verb
ii. Argument structure of the non-finite verb
iii. Adjuncts of finite and the non-finite verb
iv. Noun modifications
v. Adjectival modifications

Using example (1) below we describe how the parser handles a simple declarative sentence.

(1) *wulasi golilu mAnesiMxi*
'Tulasi' 'tablets' 'stopped using'
'Tulasi stopped using the tablets'

CBHP begins by constructing the constraint graph (CG) for the above sentence. A constraint graph shows all the potential arcs that can exist between the heads and their corresponding dependents. This information is obtained from the demand frame of the head. According to the frame the verb can take k1[1] and k2 as mandatory children with null postpositions. Figure 1 shows that the demand frame for *mAnesiMxi* is obtained from basic demand frame of root verb *mAnu* and the null frame of suffix *–esiMxi* (which represent the tense, aspect and modality (TAM)). The basic demand frame for a verb or a class of verbs specifies the suffix, category, etc. permitted for the allowed relations for a verb when the verb has the basic TAM label. In Figure 2(a) we show the constraint graph that is constructed using the demand frame. The final parses are obtained by converting the CG into integer programming equations and using bi-partite graph matching (Bharati et al., 1993, 2002). Figure 2(b) shows the final parses obtained from the CG. Notice that we get two parses. This indicates the ambiguity in the sentence if only the limited knowledge base is considered. Stated another way, H-constraints (in the form of demand frames) are insufficient to restrict multiple analysis of a given sentence and that more knowledge (semantics, other preferences, etc.) is required to curtail the ambiguities. We will see in Section 5 how we can obtain the best parse from these multiple parses. Notice also the presence of the _ROOT_ node. By introducing _ROOT_ we are able to attach all un-

[1]k1 can be roughly translated to 'agent', k2 can be roughly translated as 'theme'. CBHP follows the syntactic-semantic dependency labels proposed in (Begum et al., 2008).

processed nodes to it. _ROOT_ ensures that the output we get after each stage is a tree.

Example (2) is a slightly complex construction containing a non-finite verb. Such sentences are also handled in the 1st stage.

(2) *wulasi rogaM waggiMxani golilu*
'Tulasi' 'disease' 'having reduced-so' 'tablets'
mAnesiMxi.
'stopped using'.
'As the disease reduced, Tulasi stopped using the tablets'

We have already seen through (1) how a sentence with a finite verb like *mAnesiMxi* can be parsed. In (2) in addition to a finite verb we have a non-finite verb *waggiMxani*. Like last time we first get the basic demand frame of the non-finite verb *waggiMxani*. Basic demand frames always represent the argument structure of a verb with its default TAM (present, indefinite). When a new TAM appears with the verb, we transform the original frame to get the final frame that is accessed during the construction of the CG.

2.2 Relations handled in Stage 2

Stage 2 handles more complex inter-clausal relations such as those involved in constructions of coordination and subordination between clauses. Stage 2 handles the following relations:

i. Clausal coordination
ii. Clausal subordination
iii. Non-clausal coordination
iv. Clausal complement

Basically, 2nd stage functions like the 1st stage, except that in the 2nd stage the CG has very few nodes. This is because the 1st stage parse subtrees are mostly not modified in the 2nd stage and only those nodes that are relevant for 2nd stage relations are considered. Take (3) as a case in point.

(3) *wulasi golilu mAnesiMxi mariyu paniki*
'Tulasi' 'tablets' 'stopped using' 'and' 'work'
velYliMxi
'went'
'Tulasi stopped using tablets and went to work'

Figure 3 shows the first stage parse for this sentence. We can see that the analysis of the two clauses is complete. The root of these subtrees and the conjunct *mariyu* are seen attached to the _ROOT_. Figure 4 shows the 2nd stage CG for (3). Notice that only the two finite verbs and the conjunct are seen here. The relations identified in the 1st stage remain untouched. Figure 5 also shows the final parse for this sentence; notice that the outputs of the two stages are combined to get the final parse. Merging the 1st stage and 2nd stage is not problematic as the two stages are mutually exclusive.

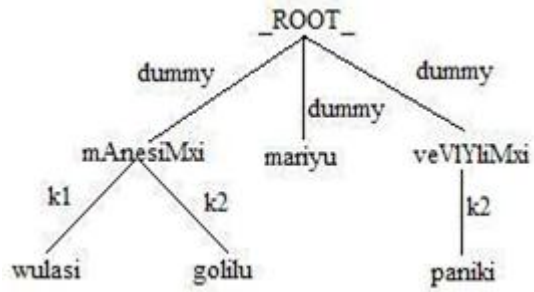


Figure 3. 1st stage parse output of (3)

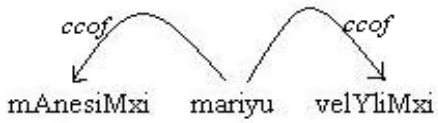


Figure 4. 2nd stage constraint graph for (3)

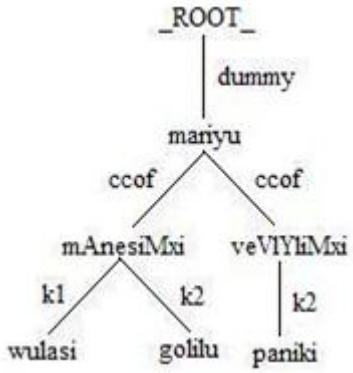


Figure 5. Final parse output of (3)

We must note here that for few cases such as (4) below, the 1st stage output is revised. We follow the same principles as described in (Bharati et al., 2008). This mainly concerns case dropping in nominal coordinations.

(4) *wulasi mariyu rama golilu*
'Tulasi 'and' 'Rama' 'tablets'
mAnesAru.
'stopped using'
'Tulasi and Rama stopped using tablets.'

2.3 H-constraints

As mentioned earlier H-constraints in CBHP mainly comprises of the lexical constraints. This as we saw in section 3.1 corresponds to the basic demand frame and the transformation frame. For Telugu CBHP we manually prepared around 460 verb frames and 95 transformation frames. The transformation frames handles various alternations that are brought in by non-finite and passive TAMs. High frequency verbs and tense, aspect and modality markers were extracted from the training data to prepare the frames. Similarly, other heads such as conjuncts were also extracted. Preparation of these frames took around 30 days.

3 Results

In this section we describe the data that was used for evaluation. We then give the oracle result of the core parser on this data, following which we discuss the results and the error analysis. The oracle parse for a sentence is the best available parse amongst all the parses of that sentence. It is obtained by selecting the parse closest to the gold parse. The oracle accuracy gives the upper-bound of the parser accuracy and gives some idea about its coverage.

3.1 Data

All the results in this paper are reported for Telugu. We use the Telugu data set that was released as part of the ICON Tools Contest 2010 (Husain et al., 2010). The training data had 1300 sentences, development and test set had 150 and 150 sentences respectively. The current parser does not handle ellipses and therefore all the sentences with NULL nodes have been removed to report the results. This data has 1119 training, 133 development and 127 test sentences.

3.2 Results

Table 1 below shows the oracle accuracies of the parser for the development and testing data. We see that the UAS (unlabeled attachment score) for both test and development is very good; the accuracies for LAS (labeled attachment score) however are not. In Table 2 we show the breakup of the results into intra-clausal and inter-clausal relations. We see that on an average the inter-clausal relations are being identified successfully, and the low LAS of Table 1 can be attributed mainly to the intra-clausal relations.

	LAS	UAS	LS
Development	68.06	84.41	70.34
Testing	65.33	84.14	66.60

Table 1. Overall parser oracle accuracy

		LAS	UAS	LS
Devel	**Intra**	59.83	82.82	63.15
	Inter	85.89	87.73	85.89
Test	**Intra**	57.92	85.11	59.87
	Inter	79.63	82.09	79.63

Table 2. Intra-clausal (Intra) and Inter-clausal (Inter) relation accuracy

4 S-constraints and Prioritization

It was clear from section 3 that the core parser that uses H-constraints produces multiple parses. S-constraints are those constraints that are used in a language as preferences and hence can be used to rank the multiple parses. These S-constraints can be used for ranking by penalizing a parse for the constraint that it violates and finally choosing the parse that gets least penalized. This strategy is similar to the one used in Weighted Constraint Grammar (Schröder, 2002). The other way is to use these S-constraints as features, associate weight with them, use them to score the output parses and select the parse with the best score. This is similar to the work on ranking in phrase structure parsing (Coolins and Koo, 2005; Shen et al., 2003) and and graph-based parsing (McDonald et al., 2005). We use the latter strategy. The score of a dependency parse tree t=(V, A) in most graph-based parsing system (Kubler et al., 2009) is

$$\text{Score}(t) = \text{Score}(V, A) \in \mathbf{R} \qquad \text{(I)}$$

where, V and A are the set of vertices and arcs. This score signifies how likely it is that a particular tree is the correct analysis of a sentence S. Many systems assume the above score to factor through

the scores of subgraph of t. Thus, the above score becomes

$$\text{Score}(t) = \Sigma_{\alpha \in \alpha t}\ \lambda_{\alpha} \quad \text{(II)}$$

where, α is the subgraph, α_t is the relevant set of subgraph in t and λ is a real valued parameter. If one follows the arc-factored model for scoring a dependency tree (Kubler et al., 2009) like we do, the above score become

$$\text{Score}(t) = \Sigma_{(i, r, j) \in A}\ \lambda_{(i, r, j)} \quad \text{(III)}$$

In (III) the score is parameterized over the arcs of the dependency tree. Since we are interested in using this scoring function for ranking, our ranking function (R) should therefore select the parse that has the maximum score amongst all the parses (Φ) produced by the core parser.

$$R(\Phi, \lambda) = \text{argmax}_{(t=V,A) \in T}\ \text{Score}(t) = \text{argmax}_{(t=V,A) \in T} \Sigma_{(i, r, j) \in A}\ \lambda_{(i, r, j)} \quad \text{(IV)}$$

Since, in our case $\lambda_{(i, r, j)}$ represent probabilities therefore it is more natural to multiply the arc parameters instead of summing them.

$$R(\Phi, \lambda) = \text{argmax}_{(t=V,A) \in T}\ \text{Score}(t) = \text{argmax}_{(t=V,A) \in T} \Pi_{(i, r, j) \in A}\ \lambda_{(i, r, j)} \quad \text{(V)}$$

For us $\lambda_{(i, r, j)}$ is simply the probability of relation *r* on arc i → j given some S-constraints (Sc). This probability is obtained using the MaxEnt model (Ratnaparkhi, 1998). So,

$$\lambda_{(i, r, j)} = p(r_{i,j} \mid \text{Sc}) \quad \text{(VI)}$$

If A denotes the set of all dependency labels and B denotes the set of all S-constraints then MaxEnt ensures that p maximizes the entropy

$$H(p) = - \Sigma_{x \in E}\ p(x) \log p(x) \quad \text{(VII)}$$

where $x = (a,b)$, $a \in A$, $b \in B$ and $E = A \times B$. Note that, since we are not parsing but prioritizing, unlike the arc-factored model where the feature function associated with the arc parameter consists only of the features associated with that specific arc, our features can have wider context. Some of the S-constraints that have been tried out are: (1) *Order of the arguments*, (2) *Relative position of arguments with respect to the verb*, (3) *Agreement*, (4) *General graph properties*.

These S-constraints get reflected as features that are used in MaxEnt. The features for which the model gave the best performance are given below. Note that the actual feature pool was much larger, and some features like that for agreement did not get selected.

(1) Root, POS tag, Chunk tag, suffix of the current node and its parent
(2) Suffix of the grandparent, Conjoined suffix of current node and head
(3) Root, Chunk Tag, Suffix, Morph category of the 1st right sibling
(4) Suffix, Morph category of the 1st left sibling
(5) Dependency relations between the first two, right and left sibling and the head
(6) Dependency relation between the grandparent and head
(7) Dependency relation between the current node and its child
(8) A binary feature to signify if a k1 already exist for this head
(9) A binary feature to signify if a k2 already exist for this head
(10) Distance from a non-finite head

5 Prioritization Results and observations

Table 3 shows the result of the MaxEnt model[2] on the development and test data. The features used for training were mentioned in the previous section.

	Accuracy
Development	76.62
Test	76.78

Table 3. Accuracy of the MaxEnt labeler

The ranked parse score (Rank-P) for both development and testing data is shown in Table 4. We can see that the best UAS is very close to the oracle UAS. The difference however is wider for LAS.

[2]http://maxent.sourceforge.net/

		LAS	UAS	LA
Devel.	**Rank-P**	59.51	81.56	63.12
Test	**Rank-P**	58.99	82.45	61.10

Table 4. Parser accuracy after Ranking

The average number of output parses for each sentence is around 10. It was noticed that the differences between these parses were very minimal and this makes ranking them a non-trivial task. The closeness between parses is quite expected from a constraint based parser whose output parses are only those that do not violate any of the H-constraints. In other words most of the output parses are linguistically very sound. Of course, linguistic soundness is only restricted to morpho-syntax and does not consider any semantics. This is because the H-constraints do not incorporate any semantics in the parser as of now. Considering this, the error analysis doesn't throw up any big surprises. The main reasons why the LAS suffers can be attributed to:

i. *Lack of explicit post-positions or presence of ambiguous one*: Errors because of this, manifest themselves at different places. This can lead to attachment error. Few common cases are finite and non-finite argument sharing, confusion between finite and non-finite argument, adjectival participle, appositions, etc. Also, it was noted that the most frequent errors are for those arguments of the verb, that have no post-position. Consequently, relations such as 'k1', 'k2', 'k7' and 'vmod' have very high confusion. The other major error caused by lack of postposition is the selection of parses with argument ordering errors.

ii. *Multiple parses with the same score*: It is possible that more than one parse finally gets the same score. This is partly caused due the above reason but it also reflects the accuracy of the labeler. As the accuracy of the labeler increases this problem will lessen. Currently, we select only the first parse amongst all the parses with equal score.

Our system's official result is much less than the results in Table 4. As noted earlier this is primarily because the parser does not handle ellipses and in those sentences fails to identify the correct heads.

	LAS	UAS	LA
Test	48.08	76.29	50.25

Table 5. Official Score

6 Conclusion and Future directions

In this paper we successfully adapted a constraint based hybrid parser for Telugu. We showed that the parser is broad coverage and handles various syntactic phenomenons. We motivated the analysis in two stages and showed that a finite clause can be a basis of such a division. The oracle accuracies of the parser on the development and the test data set shows that the parser performs well, however there is lot of room for improvement in LAS. Apart from incorporating more H-constraints, handling more constructions, we also plan to try and induce the H-constraints automatically from a treebank. For Hindi and Telugu, this has recently been successfully shown by (Kolachina et al., 2009). Along with the base parser, we also discussed the ranking strategy to get the best parse. We noticed that the best selected parse comes very close to the oracle UAS but lags behind in LAS. The error analysis shows that this is mainly because of lack of any explicit cues in the sentence.

References

B. Ambati, S. Husain, J. Nivre and R. Sangal. 2010. On the Role of Morphosyntactic Features in Hindi Dependency Parsing. In Proc of *NAACL-HLT 2010 workshop on Statistical Parsing of Morphologically Rich Language, Los Angeles, CA.*

R. Begum, S. Husain, A. Dhwaj, D. Sharma,L. Bai and R. Sangal. 2008. Dependency annotation scheme for Indian languages. In Proc of *IJCNLP08*

A. Bharati, S. Husain, D. Misra and R. Sangal. 2009a. Two stage constraint based hybrid approach to free word order language dependency parsing. In Proc of *the 11th IWPT09. Paris*

A. Bharati, S. Husain, M. Vijay, K. Deepak, D. Misra and R. Sangal. 2009b. Constraint Based Hybrid Approach to Parsing Indian Languages. In Proc of *The 23rd Pacific Asia Conference on Language, Information and Computation. Hong Kong*

A. Bharati, S. Husain, D. Sharma and R. Sangal. 2008. A Two-Stage Constraint Based Dependency Parser for Free Word Order Languages. In Proc of *the COLIPS IALP*.

A. Bharati, D. Sharma, L. Bai and R. Sangal. 2006. AnnCorra: Annotating Corpora Guidelines for POS and Chunk Annotation for Indian Languages. *LTRC-Technical Report TR31*.

A. Bharati and R. Sangal, T.P. Reddy. 2002. A Constraint Based Parser Using Integer Programming, In Proc of *ICON.*

A. Bharati and R. Sangal. 1993. Parsing Free Word Order Languages in the Paninian Framework. In Proc of *ACL: 93*

M. Collins, and T. Koo. 2005. Discriminative reranking for natural language parsing. In *CL p.25*-70 *March05*.

R. Debusmann, D. Duchier and G. Kruijff. 2004. Extensible dependency grammar: A new methodology. In Proc of *Workshop on Recent Advances in Dependency Grammar, pp. 78–85*.

G. Eryigit, J. Nivre and K. Oflazer. 2008. Dependency Parsing of Turkish. *Computational Linguistics 34(3), 357-389.*

Y. Goldberg and M. Elhadad. 2009. Hebrew Dependency Parsing: Initial Results. In Proc *of the 11th IWPT09.*

J. Hall, J. Nilsson, J. Nivre, G. Eryigit, B. Megyesi, M. Nilsson and M. Saers. 2007. Single Malt or Blended? A Study in Multilingual Parser Optimization. In Proc of *EMNLP-CoNLL shared task*

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*. Kharagpur, India *to appear*

S. Husain, P. Gadde, B. Ambati, D. Sharma and R. Sangal. 2009. A modular cascaded approach to complete parsing. In Proc of *The COLIPS International Conference on Asian Language Processing (IALP) Singapore*

F. Karlsson, A. Voutilainen, J. Heikkilä and A. Anttila. (eds). 1995. *Constraint Grammar: A language-independent system for parsing unrestricted text.* Mouton de Gruyter.

P. Kolachina, S. Kolachina, A.K. Singh, V. Naidu, S. Husain, R. Sangal and A. Bharati. 2009. Grammar Extraction from Treebanks for Hindi and Telugu. In Proceedings of *The 7th LREC. Valleta. Malta.*

S. Kubler, R. McDonald and J. Nivre. 2009. *Dependency parsing.* Morgan and Claypool.

A. Martins, N. Smith and E. Xing. 2009. Concise Integer Linear Programming Formulations for Dependency Parsing. In Proc of the *ACL-IJCNLP09.*

H. Maruyama. 1990. Structural disambiguation with constraint propagation. In Proc of *ACL:90*

R. McDonald, F. Pereira, K. Ribarov, and J. Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. *Proceedings of HLT/EMNLP*, pp. 523–530.

R. McDonald and J. Nivre. 2007. Characterizing the Errors of Data-Driven Dependency Parsing Models. In Proc of *Joint Conference on EMNLP and CoNLL*.

J. Nivre. 2009. Non-Projective Dependency Parsing in Expected Linear Time. In Proc of *ACL-IJCNLP*.

A. Ratnaparkhi. 1998. *Maximum entropy models for natural language ambiguity resolution.* Ph.D. Dissertation, University of Pennsylvania. IRCS Tech Report IRCS-98-15.

I. Schröder. 2002 *Natural Language Parsing with Graded Constraints*. PhD thesis, Hamburg Univ

D. Seddah, M. Candito and B. Crabbé. 2009. Cross parser evaluation : a French Treebanks study. In proc of *the 11th IWPT09.* Paris

L. Shen, A. Sarkar and A.K. Joshi. 2003. Using LTAG Based Features in Parse Reranking. In Proc of *EMNLP*.

P. Tapanainen and T. Järvinen. 1997. A non-projective dependency parser. In Proc of *the 5th Conference on Applied Natural Language Processing, pp. 64–71.*

R. Tsarfaty, D. Seddah, Y. Goldberg, S. Kuebler, Y. Versley, M. Candito, J. Foster, I. Rehbein and L. Tounsi. 2010. Statistical Parsing of Morphologically Ricj Languages (SPMRL) What, How and Wither. In Proc of *NAACL-HLT 2010 workshop on SPMRL, Los Angeles, CA.*

R. Tsarfaty and K. Sima'an. 2008. Relational-Realizational Parsing. In Proc of *the 22nd CoLing*

Experiments with MaltParser for parsing Indian Languages

Sudheer Kolachina
LTRC, IIIT-Hyderabad
sudheer.kpg08@research.iiit.ac.in

Prasanth Kolachina
LTRC, IIIT-Hyderabad
prasanth_k@students.iiit.ac.in

Manish Agarwal
LTRC, IIIT-Hyderabad
manish_agarwal@students.iiit.ac.in

Samar Husain
LTRC, IIIT-Hyderabad
samar@research.iiit.ac.in

Abstract

In this paper, we describe our experiments on applying the MaltParser for parsing Indian languages. We explore two new kinds of features hitherto unexplored in the parsing of Indian languages, namely, valency and arc-directionality. We motivate the use of arc-directionality features based on empirical evidence in the form of statistics collected over the treebanks. Valency information is made available during parsing through feature propagation. Finally, we blend the outputs of different single Malt parsers built in our experiments to create two blended systems (one coarse-grained and one fine-grained) each for Hindi, Telugu and Bangla. This is also the first attempt at exploring ensemble approaches such as blending for parsing Indian languages. Our submission consisting of these blended systems was ranked second overall at the ICON-2010 tools' contest on IL parsing.

1 Introduction

The earliest attempt at building NL parsing systems for Indian languages was made using a constraint-based parsing approach in the rule-based paradigm (Bharati and Sangal, 1993). This approach worked with a dependency-based representation of syntactic structure as Indian languages have relatively free word order. The main intuition behind this approach was that, given the rich morphology of these languages, morphological markers serve as strong cues for identifying the syntactic relations between the words in a sentence. In recent times, the development of syntactically annotated treebank corpora has paved the way for an alternative paradigm for parsing Indian languages: data-driven (or statistical) parsing. Statistical approaches require less effort to build parsers, although their performance is directly related to the quality and size of the treebank used to build parsers. Although portability to new languages is one of the widely claimed advantages of the data-driven approach, porting statistical parsers to new languages is not always straightforward and has been shown to throw up challenging research issues (Eryigit et al., 2008). Much of the recent work on application of data-driven approaches for parsing Indian languages which are morphologically rich and free word order (MoR-FWO) has focused on incorporating language-specific properties as features that are crucial during parsing. We briefly overview some of this work in section 2. In this paper, we continue this search for new ways to incorporate linguistic knowledge in the form of features that might prove to be beneficial for parsing Indian languages. We experiment with the latest version of the publicly available MaltParser (Nivre et al., 2007). The primary novelty of the experiments reported in this paper is in the exploration of two new kinds of linguistic information: valency information using the new *propagation* feature in MaltParser-1.4 and arc-directionality features which we argue, based on the statistics presented in section 3, are very pertinent for Indian languages. The results of our experiments show that incorporating even a small number of such features based on linguistic intuition can improve the performance of the parser. In the second stage of parser optimization, we use an ensemble technique to combine the outputs of different single Malt parsers to build blended parsers for each language.

The rest of the paper is divided into four sec-

tions. In section 2, we briefly discuss previous work on data-driven parsing for Indian languages. In section 3 we give a very brief description of the treebanks accompanied by a few interesting statistics from the treebanks based on which we motivate feature selection in our experiments. Section 4 contains the details of our experiments with the MaltParser (Nivre et al., 2007) on the multilingual datasets along with the parsing accuracies obtained in each experiment. Finally, we conclude the paper in section 5 with a concise summary of the insights gained from our experiments.

2 Related Work

Bharati et al. (2008) are the first to apply data-driven parsing approaches to Indian languages. In this work, they apply two data-driven parsers-MaltParser and MSTParser (McDonald et al., 2005) to the Hindi dependency treebank. In this work, it is shown that incorporating just two light semantic features can improve the parsing accuracy considerably.

During the ICON-09 tools' contest on dependency parsing of Indian languages[1], a variety of different approaches were tried out for parsing Indian languages. Husain (2009) presents a succinct summary of all these approaches. Nivre (2009) gives a detailed account of a complete set of experiments on applying the MaltParser which implements a deterministic transition-based parsing approach for parsing Indian languages, focusing on three different parsing algorithms (two projective and one non-projective), feature models and classifier settings. Feature optimization is automated using both backward deletion and forward selection techniques. The system described in this work was ranked second overall in the shared task. Ambati et al. (2009) also experiment with the MaltParser by trying out various learner settings and feature models reported by the MaltParser at the CoNLL 2007 shared task on dependency parsing (Hall et al., 2007). Additionally, they also report details of their experiments with the MSTParser. Their best results, however, are reported using MaltParser and are slightly higher than Nivre (2009). Mannem (2009) uses a bidirectional parser with two different operations to connect nodes, a non-projective operation to connect the head in a non-projective relation to its parent, and a projective operation to connect all other nodes with their parents in the dependency tree. This approach uses a bidirectional algorithm searching the sentence for the most confident dependency hypothesis in both directions (Shen and Joshi, 2008). This approach produces better results on the coarse-grained datasets compared to the fine-grained ones. It must be mentioned that this shared task dealt with shallow parsing wherein syntactic dependencies are predicted among the chunks in the sentence.

Ambati et al. (2010b) explore further the use of transition-based parsing approach for dependency parsing of Hindi. They build on previous work such as Nivre (2009) and Ambati et al. (2009) by exploring a common pool of features used by these systems. The results reported in this work are the state of the art for Hindi chunk parsing on the ICON-09 tools' contest dataset. Ambati et al. (2010a) describe two methods to use local morpho-syntactic information such as chunk type, head/non-head information, chunk boundary information, etc. during dependency parsing of Hindi sentences. This work which uses both Malt and MST parsers is also the first attempt at word level parsing for Hindi. Ambati et al. (2010) empirically demonstrate the importance of linguistic constraints in statistical dependency parsing for Hindi. Improvements in the performance of Malt and MST parsers can be achieved by imposing the simple constraint that an argument label of a verb must be assigned uniquely to only one child node.

3 Datasets and Treebank statistics

A dependency annotation scheme inspired by Paṇinian theory was proposed for syntactic treebanking of Indian languages (ILs) which are both morphologically rich and relatively free word-order (Begum et al., 2008). Currently, treebanks of three Indian languages[2], Hindi, Bangla and Telugu are being developed using this annotation scheme. The first versions of all three treebanks were released for the shared task on IL parsing held as part of ICON-2009. The latest versions of these treebanks were released for the ICON 2010 tools' contest on IL parsing earlier this year. In the latest version of the Hindi treebank, the parse trees are augmented with intra-chunk relations and the size of the treebank itself was increased considerably. The Bangla and Telugu treebanks, however, still consist of sentences annotated only for inter-

[1] http://ltrc.iiit.ac.in/nlptools2009/

[2] HyDT-Hindi, HyDT-Bangla and HyDT-Telugu

	Bangla	Hindi	Telugu
Total #edges	7252	77066	5722
Total #edges with head to right	5278	41988	4041
Total #edges with head to left	1974	35078	1681
Ratio of right-headed to left-headed edges	2.6737	1.1969	2.4039
#edges with head to right excluding *main,rsym,lwg**	5278	40637	4041
#edges with head to left excluding *main,rsym,lwg**	844	5151	228
Ratio of right-headed to left-headed edges	6.2536	7.8891	17.7236

Table 1: Arc directionality statistics extracted from the treebanks

chunk relations. The previous version of the Telugu treebank was revised keeping in mind specific issues which were encountered at the ICON-2009 tools contest on IL parsing. Husain et al. (2010) gives a detailed description of the treebank versions released this year as well as the statistics for the different datasets.

At the very outset of our work, we extracted a few statistics from the treebanks (training and validation sets combined) which have proved to be helpful in our experiments on feature optimization. These statistics were extracted to empirically verify some of the widely claimed typological properties of Indian languages. For instance, Indian languages are often described as being head-final or in other words, having a basic SOV word order. In terms of dependency-based representation, what this implies is that the direction of the dependencies would be predominantly to the right. However, at the same time, Indian languages are sometimes also characterized as having relatively free word order. In order to gain some corpus-based insight into the word order aspect of these languages, we extracted the frequencies of dependency edges with respect to the direction of their heads shown in Table 1 for all three languages. The total number of dependency edges in each of the three treebanks is shown in row 1. The number of edges with respect to the direction of the head of that dependency are shown for all the three languages in rows 2 and 3. The values of the ratios presented in row 4, although greater than 1, are not high enough to empirically support the claim that Indian languages, the direction of the dependencies is predominantly rightward. In fact, the values can be interpreted as indicating a high degree of deviation from the basic SOV order. However, a closer look at the statistics shows that this is not the case. The high numbers of leftward dependencies include linguistically uninformative (perhaps calling them less informative would be more apt) relations such as 'main', 'rsym', etc. In the case of Hindi, the very high number of leftward dependencies is also owing to intra-chunk relations such as the ones between nominal post-positions and the nouns whose inflections they mark, and also the ones between verbal auxiliaries and the main verbs they follow. The frequencies of the edges excluding these relations are shown in rows 5 and 6. The ratios in row 7 confirm that the distribution of edges with respect to their directionality is in Indian languages is skewed to the right. It may be noticed that the degree of skewing is much higher in the case of Telugu as compared to the other two languages. This is in agreement with linguistically attested facts about Hindi and Bangla in that deviations from the basic SOV order are possible in the case of finite clausal complements (which occur to the right rather than the left of the verb) in these languages. In section 4.5, we describe how we base our choice of feature on insights gained from such statistics.

4 Experiments

In our experiments, we use the freely available MaltParser (Nivre et al., 2007) which implements deterministic, classifier-based parsing with history-based feature models and discriminative learning. The parameters available in the MaltParser can be divided into three groups:

1. Parsing algorithm parameters
2. Learning algorithm parameters
3. Feature model parameters

Although we experiment with different combinations of these three parameters while defining the baseline systems initially, the main focus of our latter experiments is on feature optimization.

In fact, the main novelty of our experiments lies in the exploration of feature propagation as it is implemented in the latest version of the MaltParser[3]. There is an additional parameter that can be varied with the MaltParser- Single Malt versus Blended. Blending the output of different single Malt system proved to be beneficial for multilingual parsing at the CoNLL 2007 shared task (Hall et al., 2007). We follow this precedent of blending for multilingual parsing to combine the output of various parsers obtained in our experiments with different parameters.

4.1 Experiments with Parsing Algorithm

Malt parser implements three different families of parsing algorithms, comprising of a total of nine parsing algorithms (Nivre et al., 2007). In our experiments, we restricted our choices of the parsing algorithm to the Nivre family (Nivre, 2003; Nivre, 2004), a linear-time algorithm for projective structures that can be run in either arc-eager mode or arc-standard mode. We applied the left-to-right versions of this algorithm in both the modes to the multilingual datasets while fixing the feature model and the learner settings. We also used the root-handling option to change the parser's behavior with respect to root tokens by changing the label to *main*. We observed that the arc-eager algorithm performs better in the case of Hindi and Telugu, whereas arc-standard gives a slightly higher accuracy for Bangla.

4.2 Experiments with SVM Learner

MaltParser can be used with different learning algorithms that induce classifiers from training data (Nivre et al., 2007). The current version of MaltParser provides two inbuilt learning algorithms: LIBSVM and LIBLINEAR. In our experiments, we used the LIBSVM learner algorithm following the experiments reported by Hall et al. (Hall et al., 2007) at the CoNLL Shared Task 2007. We try out the learner settings of various languages presented in this work. The aim as well as intuition in this set of experiments was to see if the choice of learner settings can be based on the typological similarities between Indian languages and the languages explored at CoNLL 2007. We observed for all three languages that the effect of varying the learner settings on the parsing accuracies is relatively weaker when compared to the parsing algorithm parameter discussed in the previous experiment.

[3]MaltParser 1.4.1

Language	Algorithm	SVM settings
Bangla	Arc-standard	Czech settings s0t1d2g0.2c0.25r0.3e1.0
Hindi	Arc-eager	Turkish settings s0t1d2g0.12c0.7r0.3e0.5
Telugu	Arc-eager	Italian settings s0t1d2g0.1c0.5r0.6e1.0

Table 2: Best combinations of parsing algorithm and learner settings for different languages

In table 2, we report the combinations of learner settings and the parsing algorithm for different languages for which the best accuracies were obtained in this set of experiments. As can be seen from the table, the values of the *svm_type, kernel type* and *degree* parameters of the learner are the same for all three languages. Hindi and Bangla share the same value for the kernel coefficient parameter (r). On the other hand, the value of the termination criterion (e) is the same for Telugu and Bangla.

4.3 Initial experiments with Feature Models

In the initial set of experiments described above, we used a baseline feature model for Hindi that consisted of features reported in Ambati et al (2010b). In the case of Bangla and Telugu, we treated the feature models used by the best performing system in last year's contest (Ambati et al., 2009) as our baseline feature models. Using these feature models and the best combinations of the other two parameters obtained from the previous experiments, we built a parser for each language which we define as our baseline system for that language for the rest of the experiments in our work. Using these baseline settings, we performed ten-fold cross-validation on the multilingual datasets, and the corresponding accuracies are shown in table 3. Also shown are the accuracies obtained on the development set. Note the absence of accuracies on the coarse-grained[4] datasets from this table. This is because we restricted ourselves to the fine-grained datasets in our experiments on feature optimization.

We also explored the possibility of directly using the best feature models of different languages reported by the MaltParser at the CoNLL 2007

[4]The coarse-grained datasets were created by reducing the granularity of the dependency labels in the fine-grained sets. See Husain et al. (2010) for further details.

Dataset	Cross-Validation			Development set		
	LAS	UAS	LAcc	LAS	UAS	LAcc
Bangla-fine	67.74	84.84	70.01	68.47	88.42	70.44
Hindi-fine	85.73	92.79	87.59	85.91	92.37	87.60
Telugu-fine	67.98	89.78	69.90	70.85	91.12	72.70

Table 3: Baseline parsing accuracies on all datasets; Cross-validation accuracies with baseline model on training sets; baseline results on development set. LAS = labeled attachment score; UAS = unlabeled attachment score; LAcc = label accuracy.

shared task (Hall et al., 2007), the intuition behind this set of experiments being similar to the one discussed in the case of learner settings. We were able to complete this set of experiments for Telugu and Bangla as the datasets of these languages are quite small. But in the case of Hindi, due to the relatively large size of the datasets and the tight constraints on time and resources, we could not train parsers for Hindi using all these different feature models. We used parsers built this way in our final experiment while blending the output of different single Malt parsers to build our final system. In this set of experiments, we observed that the best accuracies for both Telugu and Bangla were obtained using the feature model for Basque which is also a non-configurational language with rich morphology and relatively free word order.

4.4 Experiments with Feature Propagation

After establishing the baseline systems, we began to explore the idea of feature propagation through dependency arcs as a means to use valency information during parsing. Feature propagation is a well-known strategy used in classical unification-based grammars, as a means of propagating linguistic information through the syntax tree. The application of this idea was explored previously for dependency parsing of Basque by Bengoetxea and Gojenola (2009; 2010). Feature propagation in this work was achieved through stacking. In our experiments, we use feature propagation as it is implemented in the recent version of the MaltParser[5]. In this implementation, values of propagated features get copied to temporary data columns and are accessible during parsing. Since the goal of our experiments on feature propagation was to incorporate information about valency into the parsing process, we propagated the labels of the outgoing arcs up to the parent nodes. The idea is to make the valency and hence argument structure of nodes accessible during parsing. Due to the fine-grained nature of the dependency labels in the datasets, we only propagated the labels of mandatory arguments as opposed to adjuncts. The following labels were considered as argument labels: *k1, k1s, k2, k2p, k4 and k4a*. As expected, we observed an improvement, although slight over the baseline accuracies when the argument labels of the top of Stack and the head of the top of Stack were propagated. Further, we observed that the better improvements could be obtained by merging the valency of these elements with the POS-tag, CPOS-tag and lemma features of the next token in the input buffer (Input[0]). The accuracies in Table 4 show that valency information improves only the UAS in the case of Telugu and both UAS and LAS in the for Hindi. In the case of Bangla, the accuracies actually drop with the addition of these new features.

[5]MaltParser 1.4.1

Experiment	Dataset	LAS	UAS	LAcc
V	Bangla-fine	67.61	87.81	69.95
	Hindi-fine	86.07	92.54	87.87
	Telugu-fine	70.69	92.13	72.36
V+ArcDir	Bangla-fine	68.23	88.30	70.57
	Hindi-fine	85.91	92.41	87.81
	Telugu-fine	71.36	92.13	73.37

Table 4: Parser accuracies on development set using valency information and arc-directionality features; V: Valency information, ArcDir: Arc Directionality information

4.5 Additional features

In this last set of experiments on tuning the feature models, we tried to augment the current feature models of different languages with more features. It is at this point that we found the statistics extracted from the treebanks to be useful while handpicking features to augment the feature mod-

	Baseline model			Best Feature model			Blended		
Dataset	**LAS**	**UAS**	**LACC**	**LAS**	**UAS**	**LACC**	**LAS**	**UAS**	**LACC**
Bangla-Fine	68.26	86.16	71.49	68.57	85.54	71.28	70.14	87.1	73.26
Hindi-Fine	86.60	93.37	88.22	86.49	93.50	88.16	86.22	93.25	87.95
Telugu-Fine	67.78	88.48	69.95	69.28	90.48	70.95	68.11	90.15	70.12
Bangla-Coarse	73.15	86.47	76.69	73.26	87.51	75.67	75.65	88.14	78.67
Hindi-Coarse	88.57	94.01	90.75	88.96	94.13	90.91	88.96	94.13	90.91
Telugu-Coarse	68.95	88.81	71.45	69.62	90.15	71.79	69.45	89.98	71.29

Table 5: Final results on the test set for different languages

els. Based on the insights we gained about the skewed distribution of the edges with respect to the direction of their heads (see table 1), we chose to experiment with the InputArcDir feature function in MaltParser. In our experiments, we observed that merging this feature function that deals with arc-directionality of the arc outgoing from the top of Stack with POS-tag, CPOS-tag and lemma features of both the head of top of Stack and the next token in the input buffer brings about an improvement in the LAS for Telugu. Information about the direction of the arc seems to be helpful for label prediction in Telugu. Tuning the feature models this way provides a better understanding of how a particular feature works across different languages. Apart from automating feature selection from a large pool of features, tuning the feature model using handpicked features based on linguistic properties of individual languages can give insights into how different features interact during parsing. The accuracies obtained in this set of experiments are also shown in table 4. While the accuracies for Telugu and Bangla show improvements, there is a drop in the case of Hindi. This can be attributed to the difference in the nature of the datasets- the Telugu and Bangla datasets contain chunk-level dependency trees while the Hindi data consists of word-level trees. There are a number of intra-chunk dependencies (*lwg**) in the case of Hindi which are not head-final (see table 1).

4.6 Blended Malt

Finally, following Hall et al. (2007), we decided to blend the different parsers (using the Maltblender tool) built in our experiments to create Blended MaltParsers for each language. An important point about an ensemble approach like blending is that its efficacy depends on the degree of variation among the parsers being blended. In our initial experiments on blending, we noticed that it is beneficial to include parsers which are trained using entirely different settings although their performance may be relatively low. In other words, blending systems built using different parameter settings is preferable to simply blending systems with the highest accuracies. Further, we also observed that weighting the dependencies according to labeled accuracy per coarse-grained POS-tag of the parsers produces the best results. Here again, as elsewhere, we resort to the typological properties of languages to decide which parsers to combine for each language. We blend the single Malt parsers built using our feature models with the ones built using feature models of non-configurational languages from the CoNLL shared task- Arabic, Basque, Czech, Greek, Hungarian and Turkish. In the case of Telugu, we additionally used the parser built using the feature model for Catalan as it gave good accuracies on the development set. In the case of Bangla, blending was done similarly except that we used the parser built using the feature model of English instead of Catalan. For Hindi, as mentioned earlier, we did not have such a variety of systems and therefore, the blended system was created by combining the output of the only five parsers. As a result, blending does not bring about any improvements in the case of Hindi.

The final results obtained on the test sets for all three languages using the various systems built in our experiments are shown in Table 5. The accuracies in column 2 are using the best feature model for each language obtained in our experiments on feature model tuning.

5 Conclusions and Future work

In this paper, we experiment with different parameters in the MaltParser to build dependency parsers for three Indian languages: Hindi, Telugu and Bangla. The novelty of our experiments lies in

the exploration of new features pertaining to valency and arc-directionality. Using feature propagation, we make valency information accessible to the parser during parsing. We motivate the use of arc-directionality features on the basis of empirical evidence collected from the treebank. The results of our preliminary experiments show that the use of both these novel features can improve parsing performance. Finally, we blend the output of different single Malt parsers to create a blended Malt parser for each of the languages. We report the parsing accuracies obtained in each set of experiments. Since our main aim was to build high quality parsers for Indian languages, we tried out different ways to this end. The experiments in our work must, therefore, be seen as preliminary and further experiments need to be carried out before definitive claims can be made about the impact of both valency and arc-directionality information. Also, with regards to blending, in hindsight, it would have been really useful to have more base parsers for Hindi. However, the results of blending for Telugu and Bangla show ensemble approaches to be a new direction worth pursuing for parsing Indian Languages.

Acknowledgements

We thank Prof. Joakim Nivre for helping us out with the new propagation feature in MaltParser 1.4. We would also like to thank Bharat Ram Ambati and Phani Gadde for giving us the baseline feature models. Thanks are also due to two reviewers for their useful comments and feedback.

References

B.R. Ambati, P. Gadde, and K. Jindal. 2009. Experiments in Indian Language Dependency Parsing. *Proceedings of the ICON09 NLP Tools Contest: Indian Language Dependency Parsing*, pages 32–37.

B.R. Ambati, S. Husain, S. Jain, D.M. Sharma, and R. Sangal. 2010a. Two methods to incorporate local morphosyntactic features in Hindi de-pendency parsing. In *The First Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2010)*, page 22.

B.R. Ambati, S. Husain, J. Nivre, and R. Sangal. 2010b. On the role of morphosyntactic features in Hindi dependency parsing. In *The First Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2010)*, page 94.

B.R. Ambati. 2010. Importance of linguistic constraints in statistical dependency parsing. In *Proceedings of the ACL 2010 Student Research Workshop*, pages 103–108. Association for Computational Linguistics.

R. Begum, S. Husain, A. Dhwaj, D.M. Sharma, L. Bai, and R. Sangal. 2008. Dependency annotation scheme for Indian languages. *Proceedings of IJCNLP-2008*.

K. Bengoetxea and K. Gojenola. 2009. Application of feature propagation to dependency parsing. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 142–145. Association for Computational Linguistics.

K. Bengoetxea and K. Gojenola. 2010. Application of different techniques to dependency parsing of Basque. In *The First Workshop on Statistical Parsing of Morphologically Rich Languages*, page 31.

A. Bharati and R. Sangal. 1993. Parsing free word order languages in the Paninian framework. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pages 105–111. Association for Computational Linguistics.

A. Bharati, S. Husain, B. Ambati, S. Jain, D. Sharma, and R. Sangal. 2008. Two semantic features make all the difference in parsing accuracy. *Proc. of ICON*, 8.

G. Eryigit, J. Nivre, and K. Oflazer. 2008. Dependency parsing of Turkish. *Computational Linguistics*, 34(3):357–389.

J. Hall, J. Nilsson, J. Nivre, G. Eryigit, B. Megyesi, M. Nilsson, and M. Saers. 2007. Single malt or blended? A study in multilingual parser optimization. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 933–939. Association for Computational Linguistics.

S. Husain, P. Mannem, B. Ambati, and P. Gadde. 2010. The ICON-2010 tools contest on Indian language dependency parsing. *Proceedings of ICON10 NLP Tools Contest on Indian Language Dependency Parsing*.

S. Husain. 2009. Dependency Parsers for Indian Languages. *Proceedings of ICON09 NLP Tools Contest: Indian Language Dependency Parsing*.

P. Mannem. 2009. Bidirectional Dependency Parser for Hindi, Telugu and Bangla. *ICON09 NLP TOOLS CONTEST: INDIAN LANGUAGE DEPENDENCY PARSING*, page 49.

R. McDonald, K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 91–98. Association for Computational Linguistics.

J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kübler, S. Marinov, and E. Marsi. 2007. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(02):95–135.

J. Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*. Citeseer.

J. Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57. Association for Computational Linguistics.

J. Nivre. 2009. Parsing Indian Languages with MaltParser. *Proc. of ICON09 NLP Tools Contest: Indian Language Dependency Parsing*, pages 12–18.

L. Shen and A.K. Joshi. 2008. Ltag dependency parsing with bidirectional incremental construction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 495–504. Association for Computational Linguistics.

Experiments on Indian Language Dependency Parsing

Prudhvi Kosaraju, Sruthilaya Reddy Kesidi, Vinay Bhargav Reddy Ainavolu and Puneeth Kukkadapu
Language Technologies Research Centre, IIIT-Hyderabad, India.
{prudhvi, sruthilaya, vinay.ainavoluug08, puneeth.kukkadapuug08}@students.iiit.ac.in,

Abstract

In this paper we present our experiments of dependency parsing on three morphologically rich free word order Indian Languages. The languages are Hindi, Telugu and Bangla and these differ in their morphological richness. We explore the state-of-the-art Malt parser on grounds of a large feature pool with different parsing strategies. For Hindi, we show that the utilization of the shallow parser output as features helps in improving the accuracies. We also attempted to incorporate semantics in parsing. We report our results on the test data provided at ICON tools contest, 2010. The labelled attachment scores on the fine-grained test data for Hindi, Telugu and Bangla are 88.63, 70.12, and 70.55 respectively. We reported the best average labelled attachment scores in the tools contest.

1 Introduction

Natural Language Parsing is useful in major NLP applications like Machine Translation, Dialogue systems, text generation, word sense disambiguation etc. Parsing is being done for a quite long time but parsing morphologically rich, free word order languages (MoR-FWO) is still a challenging task. Dependency parsing has been approved advantageous for free word order languages (Shieber,1985; Mel'Cuk, 1988; Bharati et al,1995). The dependency parsers can be grammar driven, data driven or hybrid. Grammar driven parsers are relatively hard to construct and can be done only by a professional of the language. Data driven parsers are those that are trained on syntactically annotated corpora and a new sentence can be parsed efficiently [Nivre 2009]. Due to availability of syntactically annotated treebanks for Hindi, data driven parsing has been analyzed in various instances (Bharati et al., 2008; Husain et al., 2009; Ambati et al., 2009). Many techniques were recently tried out in this data driven parsing (Ambati et al., 2009). Incorporation of local morphosyntactic features has also shown improvements over the state-of-the-art parsers. (Ambati et al., 2010b).

In this paper we present our experiments of dependency parsing on three morphologically rich free word order Indian Languages. The languages are Hindi, Telugu and Bangla and these differ in their morphological richness. We explore the state-of-the-art Malt parser on grounds of a large feature pool with different parsing strategies. For Hindi, we show that the utilization of the output of shallow parser that reflects various chunk information as features helps in improving the accuracies. A chunk is a minimal phrase consisting of co-related, inseparable words/entities, such that intra chunk dependencies are not distorted (Bharati et al., 2006). We also attempted to incorporate semantics in parsing. We report our results on the test data provided at ICON tools contest 2010. The labelled attachment scores on the fine-grained test data for Hindi, Telugu and Bangla are 88.63, 70.12 and 70.55 respectively. For coarse-grained data, we achieved the labelled attachment scores of 88.87, 69.12 and 73.67 for Hindi, Telugu and Bangla achieved respectively.

This paper is organized as follows. In Section 2, we discuss about the ICON 2010 tools contest data and data specifications. Section 3 describes about the approach followed. In Section 4, we talk about incorporating semantics in parsing and in Section 5, we provide the parser settings for different languages and results. We conclude our work and provide the future work in Section 6.

2 Tools Contest, Data and specifications

We are provided data for three languages Hindi, Telugu and Bangla in form of training, development and testing. So far the dependency parsing experiments for Indian languages are done at the chunk level (Husain, 2009). Previous work on full sentence parsing for Hindi has been done in Ambati et al. (2010b). In the contest for Hindi, parsing is done at the sentence level and for Telugu and Bangla at the chunk level. The following table brings out the statistics from the data.

	Hindi	**Telugu**	**Bangla**
Sentences	3837	1600	1279
Words	83657	21492	28593
Unique Words	11345	3783	4924
Chunks	40506	8213	6321
Fine grained tags	101	43	43
Coarse grained tags	47	23	25

Table 1: Data Statistics

3 Approach

3.1 Malt Parser

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

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

We performed our experiments on malt parser version 1.4.1. Malt parser provides options for Arc-Standard, Arc-Eager, Covington Projective, Covington Non-Projective, Planar and Stack parsing algorithms. It also provides options for LIBSVM and LIBLINEAR learning algorithms. We experimented with different combinations of these algorithms to arrive at the best settings.

3.2 Features, Feature selection and Templates

An extensive list of features is prepared which we thought are appropriate in helping to form a better parse. Best settings for each data set are selected with a simple forward selector. The simple forward selector runs by appending the feature to template file one by one seeing when it meets the given criteria. A set of algorithms and classifiers are to be provided prior to the forward selector. We include the corresponding feature in the template file if the LAS increase and UAS does not decrease. We have used LIBLINEAR and LIBSVM classifiers and found out the LIBLINEAR has achieved slightly better accuracies over the LIBSVM. This is because of the new version of Malt parser has updated the LIBSVM and LIBLINEAR packages. A 5 fold cross validation has been done for selecting the best template, best algorithm and best classifier for Telugu and Bangla. For Hindi, we experimented on training and development sets to get the best templates, algorithms and classifiers.

From the set of morphological information provided along with the data, vibhakti and tam has helped in improving accuracies (Bharati et al., 2008; Ambati et al., 2009, 2010a). For Hindi, we experimented incorporating local morphosyntactic information similar to Ambati et al. (2010b). The local morphosyntactic information reflects various chunk information in the form of chunk type, head-non head information, chunk boundary information and distance to the end of chunk. From these, best chunk features for coarse and fine grained data sets are derived. The current baseline is formed from the best settings given by the forward selector for both fine grained and coarse grained Hindi. We observed improvement in the accuracies and the following is the table depicting the improvement on the development data over the baseline.

	Fine Grained Hindi Data			**Coarse Grained Hindi Data**		
	LAS	UAS	LS	LAS	UAS	LS
Baseline	86.32	92.43	88.51	87.45	92.34	89.85
Baseline + Best Chunk features	88.19	94.00	89.77	89.34	94.21	91.15

Table 2: Improvement in Accuracies with chunk features

4 Semantics in Parsing

4.1 Introduction

It has been shown that use of minimal semantics can help in identifying certain core dependency relations (Ambati et al., 2009). An iterative approach between parsing and semantic role labeling also showed improvement in the parsing accuracies. (Dia et al.,2009)

4.2 Data and Semantic labels

We used the data of Ambati et al. (2009) which has semantic labels for the heads of the Noun Phrase (NP) chunks. The semantic labels we chose to use for our experiments are from those prescribed in Ambati et al. (2009). The semantic labels we used are Human, Nonhuman, Inanimate and Rest.

Briefly summarising the tags:
Human (h): This tag is assigned to the nouns which represent human
Inhuman (nh): This tag is assigned to the nouns which represent animate but not humans
Inanimate (in): This tag is assigned to all the inanimate nouns
Rest (rest): This tag is assigned to all other nouns which doesn't belong to above categories

With the annotated data we built a semantic tagger using Maximum Entropy Model. (A.Ratnaparkhi, 1998).
Below statistics about the data, gives us the occurrence of each tag in the data.

	Noun Chunks
Total data size	5711
h-tag	1456
nh-tag	196
in-tag	1752
rest-tag	2307

Table 3: Semantics Annotated Data

The Hindi data provided in the contest is different from this data**.** Also the size of the data is relatively large compared to the data with semantic annotation.

4.3 Building a Semantic Tagger

Our aim is to build a robust and accurate tool which assigns the semantic tags (h, nh, in, rest) to the parsed data. We use a maximum entropy model to build a semantic labeller. We designed a large pool of features which are supposed to help in the semantic labelling and performed a 5 fold cross validation on the available data to select the best features.

Entity	**Features**
Word	Lemma, POS tag, Dependency relation, Number of children, Number of siblings
Parent	Lemma, POS tag, Dep. Relation, Chunk Information.
Child	Lemma, POS tag, Dep. Relation, Chunk Information.
Siblings	Lemma, POS tag, Dep. Relation, Chunk Information.
Previous Word	Lemma, POS tag, Dep. Relation, Chunk Information.
Next Word	Lemma, POS tag, Dep. Relation, Chunk Information.
Others	Gender, Number, Person

Table 4: Feature Pool formed by combining entity with its features

Feature Selection: The above table gives the overview of the feature pool. Feature selection is done by performing forward. We finally arrived at a best feature set which made our semantic labeller 69.53% accurate.

Best Template	**Accuracy**
Lemma (word), Dependency Relation (word), POS (parent), Chunk label (parent), Chunk label (Children)	69.527

Table 5: Best Features and Accuracy of Semantic tagger

4.4 Semantic tags as feats to Parsing

The semantic labeller built utilises the dependency parse information i.e. attachments and dependency relations. We have seen in the previous section that incorporating the local morphosyntactic information (chunk features information) has given better parses. Baseline is now set to this approach. We

ran the model on development data and to this output we appended the semantic tags with the tagger. The gold training data is also assigned with the semantic tags. These outputs are provided to semantic labeller to get the semantic tags for chunk heads with label NP. These semantic labels are appended to the FEATS column of the CoNLL format, along with vibhakti and TAM information. The semantic information failed to provide improvement in the accuracies

	Baseline	Baseline + Semantics
LAS	88.19	88.15
UAS	94.00	93.88
LS	89.77	89.77

Table 6: Accuracies when semantic tags incorporated in Parsing

4.5 Reasons for no improvement in parsing accuracies

The major reason for no improvement of parsing accuracies is due to less efficient semantic tagger. The features that we have used are failing to capture the completeness of the semantic labeller. Other major issue is with the size of the semantics annotated data. Irregular distribution of tags in the annotated data i.e. counts of occurrence of each tag, can also be a factor for less accuracy. Data provided at the tools contest is different from the data with semantic tags.

The accuracy of the parser built so far is also an issue. Though it has achieved a labelled attachment score of 88.19, the contribution for achieving high accuracy is by the intra chunk relations. Hence the inter chunk relations are still suffering with the problem of perfection. Table 7 shows the results when gold development is considered for semantic experiments.

	Baseline	Baseline + Semantics
LAS	88.19	89.29
UAS	94.00	94.06
LS	89.77	90.88

Table 7

5 Results and Observations

5.1 Results on test data

With the forward selector we arrived at the following best settings which included algorithm and classifier for each of the data set.

Data	Algorithm	Classifier
Hindi Fine Grained	Arc Standard	LIBLINEAR
Telugu Fine Grained	Arc Eager	LIBLINEAR
Bangla Fine Grained	Arc Standard	LIBLINEAR
Hindi Coarse Grained	Arc Eager	LIBLINEAR
Telugu Coarse Grained	Arc Eager	LIBSVM
Bangla Coarse Grained	Covington Non Projective	LIBLINEAR

Table 8: Best Settings from Forward Selector

The training and development are combined and then with the above settings the malt parser is run on the test data. Table 8 displays out the results on the fine grained data of three languages and Table 9 displays the results of test data on coarse grained data.

	LAS	UAS	LS
Hindi	88.63	94.54	90
Telugu	70.12	91.82	71.95
Bangla	70.55	86.16	73.36

Table 8: Fine grained accuracies

	LAS	UAS	LS
Hindi	88.87	93.62	90.79
Telugu	69.12	89.48	71.29
Bangla	73.67	86.16	77.42

Table 9: Coarse grained Accuracies

The average of the labelled attachment, unlabelled attachment and label scores on all the data sets are 76.83, 90.3 and 79.14 respectively.

5.2 Error Analysis

A basic Error analysis has been done with the Hindi fine grained results. In previous sections an overall increment is witnessed in the system's performance due to incorporation of local morphosyntactic features and a slight decrement by semantic tags. In this sub section, we shall see how the major relations are effected due to the above incorporations. Result1 are the results obtained by taking the best settings from the forward selection experiments. Result2 is the effect on results when chunk features are used on the best settings of forward selection experiments. Result3 are the results when the semantics are incorporated on the above experiment. The LAS values of the results are 86.32, 88.19 and 88.15 respectively. Here in this table we present the F-measure values of the most affected tags due to the above mentioned experiments.

Label	Result1	Result2	Result3
k1	80.24	81.54	**81.92**
k2	72.97	**72.80**	72.42
ccof	86.34	88.95	**89.24**
pof	83.29	**84.10**	83.92
r6	83.35	86.70	**87.03**
lwg__psp	99.28	**99.77**	99.77
lwg__vaux	99.69	**99.69**	99.51
nmod_adj	94.59	98.59	**98.80**

Table 10: Error Analysis

We also see the recall of k3, k4 and k5 tags dropped when the chunk features are appended. Semantics showed a minimal effect in these relations.

6 Conclusion and Future work

We explored the malt parser with its transition algorithms and classifiers. A feature pool and a feature selection algorithm are designed to get the best settings for each of the 6 data sets. Incorporation of chunk features has shown performance improvement and an effective way of incorporating semantics has been left for the future work.

Apart from the study of detailed error analysis, the other things that can be focussed are in choosing a better strategy for forming the template and in building a more accurate semantic tagger.

More accurate Semantic tagger: Improving the efficiency of semantic tagger. The present semantic tagger is only 69% accurate and amount of data annotated with semantic labels is also too low. Hence increase in the data size is one way which improves the performance. Another way is to exploit features that help in building more accurate semantic tagger. Further fine grained classification of the semantic tags might also be useful since these tags are much close to the dependency labels. Unsupervised learning can be replaced by other methods for building the semantic tagger.

Also a detail analysis can be done to observe the huge difference between the labelled and unlabelled attachment scores in Telugu.

Acknowledgements

We would like to thank Mr. Samar Husain and Mr. Bharat Ram Ambati for their valuable suggestions.

References

B. R. Ambati, S. Husain, J. Nivre and R. Sangal, 2010a. On the Role of Morphosyntactic Features in Hindi Dependency Parsing. In Proceedings of NAACL-HLT 2010 workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2010), Los Angeles, CA.

B. R. Ambati, S. Husain, S. Jain, D. M. Sharma, R. Sangal (2010b). Two methods to incorporate local morphosyntactic features in Hindi dependency parsing. In Proceedings of NAACL-HLT 2010 workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2010), Los Angeles, CA.

B. R. Ambati, P. Gadde and K. Jindal. (2009). Experiments in Indian Language Dependency Parsing. In Proceedings of ICON09 NLP Tools Contest: Indian Language Dependency Parsing. Hyderabad, India.

B. R. Ambati, P. Gade, C. GSK and S. Husain. 2009. Effect of Minimal Semantics on Dependency Parsing. In proceedings of RANLP 2009 Student Research Workshop.

A. Bharati, S. Husain, A. Dhwaj. D. M. Sharma, and R. Sangal. 2008. Two Semantic features make all the difference in parsing accuracy. In Proceedings of ICON-08.

A. Bharati, R. Sangal, D. M. Sharma and L. Bai. 2006. AnnCorra: Annotating Corpora Guidelines for POS and Chunk Annotation for Indian Languag-

es.Technical Report (TR-LTRC-31), LTRC, IIIT-Hyderabad.

A. Bharati, V. Chaitanya and R. Sangal. 1995. Natural Language Processing: A Paninian Perspective, Prentice-Hall of India, New Delhi, pp. 65-106.

Q. Dai, E. Chen, and L. Shi. 2009. An iterative approach for joint dependency parsing and semantic role labelling. In Proceedings of Computational Natural Language Learning (CoNLL-2009), June 4-5, Boulder, Colarado,USA. June 4-5

S. Husain. Dependency Parsing for Indian Languages In Proceedings of ICON09 NLP Tools Contest: Indian Language Dependency Parsing. Hyderabad, India. 2009.

J. Nivre, J. Hall, and J. 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.

J. Nivre. Non-projective dependency parsing in expected linear time. In Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP, pages 351–359, Suntec, Singapore, August 2009. Association for Computational Linguistics.

I. A. Mel'čuk. 1988. Dependency Syntax: Theory and Practice, State University, Press of New York.

S. M. Shieber. 1985. Evidence against the context- freeness of natural language. In Linguistics and Philosophy, p. 8, 334–343.

A. Ratnaparkhi. Maximum entropy models for natural language ambiguity resolution. Ph.D. Dissertation, University of Pennsylvania. IRCS Tech Report IRCS-98-15.