

Clausal parsing helps data-driven dependency parsing: Experiments with Hindi

Samar Husain*, Phani Gadde*, Joakim Nivre† and Rajeev Sangal*

* Language Technologies Research Centre, IIIT-Hyderabad, India.

† Department of Linguistics and Philology, Uppsala University, Sweden.

{samar, phani.gadde}@research.iiit.ac.in,

joakim.nivre@lingfil.uu.se, sangal@mail.iiit.ac.in

Abstract

This paper investigates clausal data-driven dependency parsing. We first motivate a clause as the minimal parsing unit by correlating inter- and intra-clausal relations with relation type, depth, arc length and non-projectivity. This insight leads to a two-stage formulation of parsing where intra-clausal relations are identified in the 1st stage and inter-clausal relations are identified in the 2nd stage. We compare two ways of implementing this idea, one based on hard constraints (similar to the one used in constraint-based parsing) and one based on soft constraints (using a kind of parser stacking). Our results show that the approach using hard constraints seems most promising and performs significantly better than single-stage parsing. Our best result gives significant increase in LAS and UAS, respectively, over the previous best result using single-stage parsing.

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; Nivre and McDonald, 2008; Tsarfaty and Sima'an, 2008; Seddah et al., 2009; Gadde et al., 2010; Husain et al., 2009; Eryigit et al., 2008). There has also been a lot of interest in building ensemble systems (Zeman and Zabokrtsky, 2005; Sagae and Lavie, 2006) and parser stacking (Nivre and McDonald, 2008; Martins et al., 2009) to improve the overall parsing accuracy by combining the strengths of multiple parsers.

In this paper, we formulate clausal parsing as a two-stage setup where intra-clausal relations are

identified in the 1st stage and inter-clausal relations are identified in the 2nd stage. We attempt to find out whether this two-stage parsing approach that has earlier been successful in constraint-based systems for parsing Hindi (Bharati et al., 2009) can also benefit data-driven parsing approaches (Nivre et al., 2006), and compare two ways of implementing this idea, one based on hard constraints (similar to the one used in constraint-based parsing), and one based on soft constraints (using a kind of parser stacking; (Nivre and McDonald, 2008)). We show that one of the ways in which clausal parsing helps is by better learning of features that leads to improved label accuracy for Hindi. In particular we show that ambiguous case markers (or suffixes) that appear with many relations can be disambiguated successfully. We also show that the setup reduces many of the traditional MaltParser (Nivre et al., 2006) errors (McDonald and Nivre, 2007). Our results show that the approach using hard constraints seems most promising and performs significantly better than single-stage parsing.

The paper is arranged as follows. In Section 2, we introduce the clause as a basic parsing unit. Section 3 gives a brief overview of two-stage parsing. In Section 4 we discuss data-driven parsing for Hindi and present two methods for implementing two-stage parsing within this framework. Section 5 explains the experimental setup, and Section 6 discusses the results. We conclude the paper in Section 7.

2 Clauses as minimal parsing units

We begin with the observation that certain dependency relations are more likely to occur between the elements inside a clause and a different set of relations are more likely in showing dependencies across clauses. We also note that the notion of clause can be correlated with short distance and long distance dependencies.

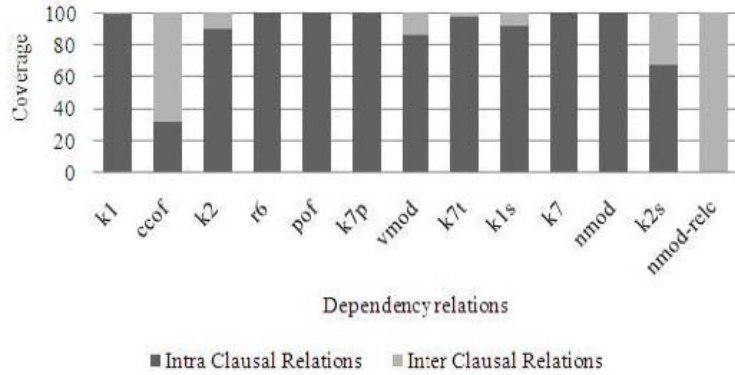


Fig 1. Dependency label distribution.

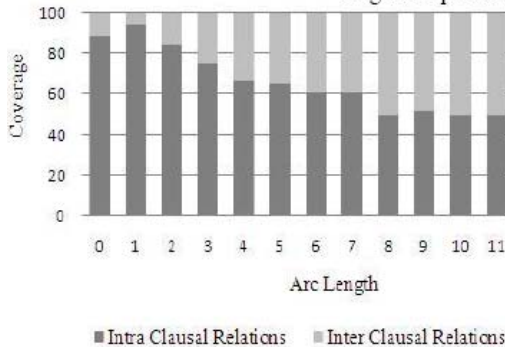


Fig 2. Arc length and relation type

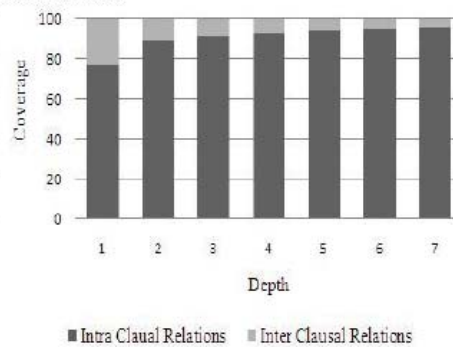


Fig 3. Depth and relation type

Figure 1 shows the distribution of dependency labels with respect to clause type (intra-clausal vs. inter-clausal) in the Hyderabad dependency treebank (Begum et al., 2008; Husain 2009). For ease of exposition, Figure 1 only shows the labels with considerable coverage, together amounting to 93% of all dependency label occurrences. We can see clearly that many labels like $k1^1$, $r6$, etc. are overwhelmingly intra-clausal relation, while others like $nmod-relc$, $ccof$, etc. have an inter-clausal bias.

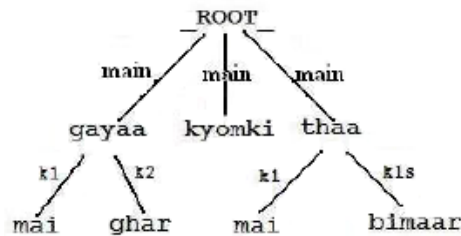
Figure 2 shows that short-distance dependencies are mostly intra-clausal, whereas long-distance dependencies tend to be inter-clausal. It is clear from Figure 1 and 2 that there is a clear correlation between labels and relation type on one hand and arc length and relation type on the other. Further, there is a correlation between inter- vs. intra-clausal relations with respect to depth of relations as well. Figure 3 shows that low depth dependencies are both inter-clausal (in

case of complex sentences involving coordination, relative clauses, embeddings, etc.) and intra-clausal (simple sentences). It also shows that the percentage of inter-clausal relations decrease with increase in depth.

Finally, there is a correlation between clause and non-projectivity: 70% of the non-projective relations are inter-clausal (Mannem et al., 2009).

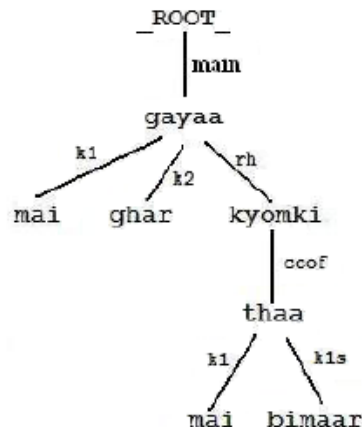
Properties such as relation type, arc length, depth, and non-projectivity are known to have specific effect on errors in data-driven dependency parsing (McDonald and Nivre, 2007). Therefore, it is worth exploring the effect of clause (when treated as a minimal unit) on dependency parsing accuracy. For all the experiment described in this paper, the following definition of clause is used: ‘A clause is a group of words containing a single finite verb and its dependents’. More precisely, let T be the complete dependency tree of a sentence, and let G be a clausal subgraph of T . Then an arc $x \rightarrow y$ in G is a valid arc, if (a) x is a finite verb; (b) y is not a finite verb; (c) there is no z such that $y \rightarrow z$, where z is a finite verb and y is a conjunct.

¹ The dependency label $k1$ can be roughly translated to ‘agent’, $r6$ is a dependency label for genitive relation, $ccof$ is a relation signifying conjunction and $nmod-relc$ is used for relative clause modification. For the complete description of the tagset and the dependency scheme see (Begum et al., 2008).



(a) Intra-Clausal

Fig. 4(a). 1st stage output



(b) Inter-Clausal

Fig. 4(b). 2nd stage output

3 Two-stage parsing

Two-stage parsing has been successfully used in a constraint based system for Hindi (Bharati et al., 2009, 2009b). This parser tries to analyze the given input sentence, which has already been POS tagged and chunked, in 2 stages; it first tries to extract intra-clausal dependency relations. In the 2nd stage it then tries to handle more complex relations such as those involved in constructions of coordination and subordination between clauses.

- (1) *mai ghar gayaa kyomki mai*
 'I 'home' 'went' 'because' 'I'
bimaar thaa
 'sick' 'was'
 'I went home because I was sick'

The 1st stage output for sentence (1) is shown in Figure 4a. In Figure 4a, the parsed matrix clause subtree *mai ghar gayaa* and the subordinate clause are attached to `_ROOT_`. The subordinating conjunction *kyomki* is also seen attached to the `_ROOT_`. The dependency tree thus obtained in the 1st stage is partial, but linguistically sound. By introducing `_ROOT_` we are able to attach all unprocessed nodes to it. `_ROOT_` ensures that the output we get after each stage is a tree. Later in the 2nd stage the relationship between the two clauses are identified. The 2nd stage parse for the above sentence is shown in Figure 4b. The 2nd stage does not modify the parse sub-trees obtained from the 1st stage, it only establishes the relations between the clauses.

4 Two-stage data-driven parsing

Since the availability of the Hyderabad Dependency Treebank for Hindi (Begum et al., 2008) a considerable amount of work has gone into exploring various data-driven approaches for Hindi parsing (Bharati et al., 2008; Husain et al., 2009; Mannem et al., 2009b; Gadde et al., 2010). The ICON09 and ICON10 tools contests on Indian language parsing (Husain, 2009; Husain et al., 2010) have also showcased various parsing efforts and established the state-of-the-art for Hindi dependency parsing. During both these parsing contest MaltParser was used to achieve the best accuracy for Hindi.

Through the experiments described in this paper, we aim to investigate the following questions:

- What are the different ways in which one can treat clauses as minimal unit during the parsing process?
- Will this help improve parsing accuracy using MaltParser?

We now present two data-driven paradigms that incorporate the notion of clause in different ways. Both paradigms use two stages to parse a sentence, but the way the two stages interact is different.

4.1 2-stage parsing with hard constraints (2-Hard)

The basic idea behind this strategy is essentially the same as constraint-based two-stage parsing. The 2nd stage MaltParser takes as input partial 1st stage trees and establishes relationships be-

tween clauses (and conjunctions). The 1st stage predictions are mutually exclusive of the 2nd stage predictions and cannot be overridden in the 2nd stage. However, they can be used as features in the 2nd stage predictions.

We now define the input to the 2nd stage for 2-Hard more precisely. Let T be the complete tree that should be output by the 2nd stage parser and let G be the subgraph of T that is input to the second stage. Then G should satisfy the following constraint: if the arc $x \rightarrow y$ is in G , then, for every z such that $y \rightarrow z$ is in T , $y \rightarrow z$ is also in G . In other words, if an arc is included in the 1st stage partial parse, the complete subtree under the dependent must also be included. Unless this constraint is satisfied, there are trees that the second-stage parser cannot construct. This means that the 2nd stage MaltParser gets initialized with only those nodes that are attached to the `_ROOT_` in the first stage parse (cf. Figure 4(a)). Figure 5 below shows the initial configuration of 2nd stage Malt for sentence 3, the input will be the 1st stage parse shown in Figure 4(a).

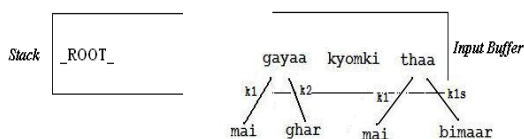


Fig 5. 2nd stage initialization using the 1st stage parse shown in Fig. 4(a)

The 1st stage and 2nd stage parser will cater to different types of constructions. Note that, given the above constraint on the 2nd stage input structures, a relative clause (though being subordinate clause) cannot be handled in the 2nd stage and will have to be handled in the 1st stage itself. We explain the handling of relative clause using sentence (2).

- (2) *vaha vahaan waba puhuchaa*
 ‘He’ ‘there’ ‘when-COREL’ ‘reached’
jaba sab jaa chuke the
 ‘when-REL’ ‘everyone’ ‘go’ ‘had’
 ‘He reached there when everyone had left’

Figure 6(a) shows the 1st stage output of a relative clause construction in a standard 2-stage setup. Both the relative clause and the matrix clause are seen attached to the `_ROOT_`, the analysis of these clauses is complete. In second stage the relation between these two clauses is established (Figure 6b). Recall that we initialize the 2nd stage of 2-Hard with the children of

`_ROOT_` which in this case is the finite verbs of the two clauses (Figure 6c). Now recall the constraint on the input of the 2nd stage in 2-Hard; given this constraint the 2nd stage can only establish a relation between the two verbs and not, as is correct, between the relative clause verb and a noun dependent on the matrix verb. The noun ‘waba’ is not present in the input buffer and can never be considered as a head of ‘jaa’. Because of this reason, 2-Hard handles relative clauses through a separate classifier after the 1st stage. This parse is then fed into the 2nd stage. This system is discussed in the next section.

4.1.1 Handling relative clauses

We add the relative clause relations to the 1st stage parse, before they are fed into the 2nd stage. This task comprises of two sub-tasks, a) relative clause identification from the 1st stage output and b) identifying the head of the relative clause from the matrix clause.

Most of the time, relative clause sentences in Hindi contain relative pronouns such as *jo* ‘who’, *jaba* ‘when’, *jisa* ‘which’ in the relative clause, which modifies an element (sometimes identified as a co-relative pronoun) in the matrix clause. The matrix clause, on the other hand, contains co-relative pronouns like *waba* ‘then’. This can be seen in the example sentence (2) in the previous section. However, both these elements can be dropped (though dropping relative pronouns is rare). This information is used in doing both the sub-tasks for the relative clause relation identification.

The identification of relative clauses is rule based and depends on the presence of an exhaustive list of relative and co-relative pronouns. Such a lexically driven approach is possible because of nature of relative clause constructions in Hindi. This system has an accuracy of 94%. The errors are mainly due to the dropping of the two type of cues discussed earlier.

Having identified the relative clause in a sentence, the remaining finite clauses are considered as possible matrix clauses. The nodes in each of these clauses are considered as possible heads for the relative clause. We use a maximum entropy (MaxEnt) based boolean classifier² to predict whether a node is a head or not. If more than one node is predicted as the head, we pick the node with the highest classifier confidence.

²http://homepages.inf.ed.ac.uk/lzhang10/maxent_toolkit.html

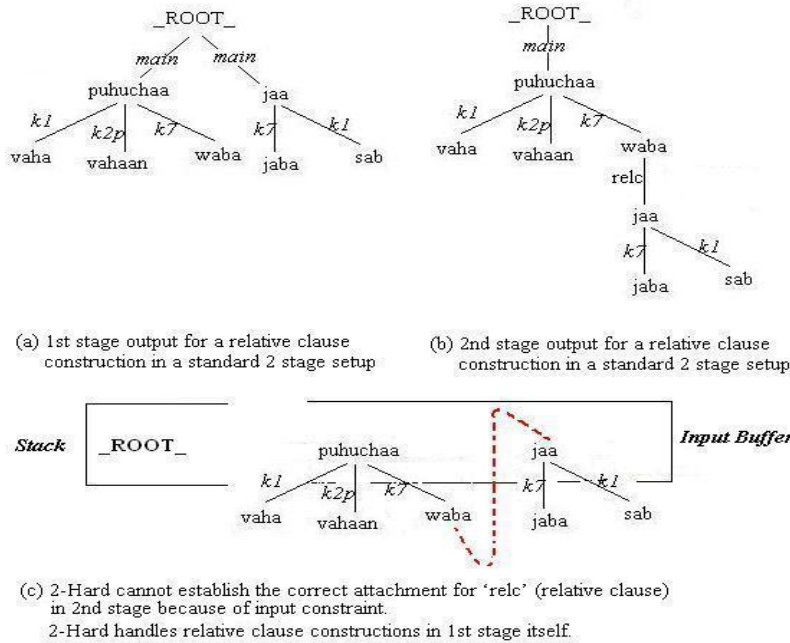


Fig. 6

The Part-Of-Speech (POS) tag of a node, its direction and distance from the relative clause are some of the important features to identify the modified noun. Note that identification of the head noun in the absence of the co-relative pronoun can be very subjective. Table 1 shows the features that are used to train the classifier.

Feature	Description	Values
Lex	Lexical item	the lexical item
POS	POS tag	NN, RB etc..
Dir	Direction of node	-1, 1
Dist	Distance of node	4,8,12, 16, 20, 24
Cue	Relative pronoun	<i>jaba, jo etc..</i>

Table 1: Features used in maxent based node classification

Dir is given 1 if a node is to the left of the relative clause and -1 if it is on the right. The distance of the node from the relative clause, *Dist*, is actually the modulus of the distance as direction is already taken care of. Further, it is normalized to the above mentioned values to reduce the sparsity. *Cue* is given "None" if there is no relative pronoun (if it is dropped) modifying the node.

We note that when compared to Husain et al. (2009) (who also do 2 stage parsing and use clauses as hard constraint) our method differs in two significant ways. The first one is obvious; they don't handle constructions such as relative

clauses etc in their setup. But more importantly, unlike Husain et al. (2009), the novel thing here is the combination of data-driven parsing and hard constraints, made possible by the new version of MaltParser that accepts partial dependency graphs as input (both during training and parsing) (cf. Figure 5).

4.2 2-stage parsing with soft constraints (2-Soft)

We can, instead of treating the output of the first-stage parser as hard constraints for the second-stage parser, treat them as soft constraints by simply defining features over the arcs produced in the first stage and making a complete parse in the second stage. Technically, this is the same technique that (Nivre and McDonald, 2008) used to integrate Malt and MST, called *guided parsing* or *parser stacking*. In this setup we 'guide' Malt with a 1st stage parse by Malt. The additional features added to the 2nd stage parser during 2-Soft parsing encode the decisions by the 1st stage parser concerning potential arcs and labels considered by the 2nd stage parser, in particular, arcs involving the word currently on top of the stack and the word currently at the head of the input buffer. For more details on the guide features for MaltParser, see (Nivre and McDonald, 2008). Note again that, unlike the standard two-stage setup the 1st stage relations can now be overridden during the 2nd stage (because we are guiding), and unlike the standard guided parsing setup a parser guides with only 1st stage relations.

Unlike the 2-stage parsing, guided parsing parses complete sentences twice. The results from one parser are used to extract features that guide the second parser. In 2-stage parsing, different components of a sentence are parsed in two stages. Interestingly, Gadde et al. (2010) have proposed an alternative way of incorporating clauses as soft constraint by using clause boundary and clausal head/non-head features during parsing. Of course, theirs is not a 2-stage setup.

5 Experimental setup

All the results are reported for Hindi. We use the Hindi data set that was released as part of ICON10 parsing contest (Husain et al., 2010). The training set had 2972 sentences, the development and test set had 543 and 321 sentences respectively. The parser models were trained using 5-fold cross validation; all the results are also reported for the cross-validation data. The setup used by (Ambati et al., 2010) for MaltParser³ is used as our baseline.

Recall that the 1st stage and 2nd stage parser of 2-Hard will cater to different types of constructions. This is sometimes also reflected in the features that get selected for each stage when compared to the Baseline settings. One such case was the absence of morphological features of lexical items in the 2nd stage for 2-Hard. The morphological properties such as suffix, category, case, etc. are crucial in establishing relations between verbs and its arguments. This in 2-Hard will be handled in the 1st stage. The relations in the 2nd stage do not require such features. On the other hand, the POS category and the lexical item of the elements in Stack and Input buffer are more crucial for 2nd stage relations than for the 1st stage specific relations. This is reflected in the selection of ‘lemma’ of the word under consideration in 2nd stage and not in baseline.

Both 2-Soft and 2-Hard 1st stage parsers are trained on a modified treebank. The original trees are transformed into 1st stage trees. This is done by using the clause definition described in Section 2. For example, the 1st stage tree for sentence 3 is shown in Figure 4 (a). On the other hand, a normal single stage parser (our baseline parser) is trained on the full tree that looks like Figure 4 (b).

6 Results and discussion

Table 2 shows the performance of the different parsers with 5-fold cross-validation. In all tables statistical significance with respect to baseline is marked with *. Significance is calculated using McNemar’s test ($p \leq 0.05$). These tests were made with MaltEval (Nilsson and Nivre, 2008).

	LAS	UAS	LA
Baseline	75.02	88.82	77.80
2-Soft	75.24	88.92	78.00
2-Hard	75.65*	89.1*	78.73*

Table 2: Overall parsing accuracy (5-fold cross-validation)

We see that both 2-Soft and 2-Hard outperform the Baseline result. However, only 2-Hard is statistically significant with that of Baseline for LAS, UAS as well as LS. 2-Soft, though giving a minimal improvement in the accuracies, is not statistically significant with the baseline. However, on analyzing the output parses of all the three setups, we found clear and similar improvement patterns (listed below) in case of both 2-Hard and 2-Soft. This led us to look at the sentences having at least two clauses, where the effect of the proposed approaches is more prominent. These constitute 50.4% of the total sentences in the data. Table 3 below shows the parsing accuracies of all the setups on these complex sentences.

	LAS	UAS	LA
Baseline	74.87	88.82	77.44
2-Soft	75.25*	89.03	77.78*
2-Hard	75.83*	89.36*	78.85*

Table 3: Parsing accuracy on complex sentences (sentences having >1 finite clauses) (5-fold cross-validation)

Interestingly, the improvements in both 2-Soft and 2-Hard are better than those shown in Table 2. Also, 2-Soft is now statistically significant compared to the Baseline w.r.t. LAS and LS. Overall, both the approaches seem to perform better for labels over attachments. To analyze the results further, we breakdown the overall accuracy (shown in Table 2) into inter-clausal and intra-clausal accuracies. Table 4 below shows these results.

³ MaltParser (version 1.3.1)

	LAS		UAS		LA	
	Intra	Inter	Intra	Inter	Intra	Inter
Baseline	72.18	85.43	89.05	87.98	75.26	87.13
2-Soft	72.44	85.54	89.13	88.16	75.49	87.17
2-Hard	72.36	87.71	88.87	90.10	75.47	90.68

Table 4: Overall accuracy for intra- and inter-clausal dependency relations

Table 4 shows some interesting facts. 2-Hard performs far better than Baseline and 2-Soft in case of inter-clausal relations, where as its effect is less for intra-clausal relations. 2-Soft, on the other hand, gives the best accuracies for intra-clausal relations over all the metrics. Note that the 2nd stage in 2-Soft approach has the flexibility to modify the dependencies given by the 1st stage parse. This could be the possible reason for 2-Soft performing better than 2-Hard for the intra-clausal relations, which are large in number as well as consisting of more deviant patterns compared to the inter-clausal relations.

These experiments show us that there is a clear pattern in cases where parsing benefits from 2-Soft and 2-Hard. These benefits are spread over both 1st stage and 2nd stage. These cases are:

1. Better identification of some relations with deviant/ambiguous postpositions in the 1st stage. For example, when ‘se’ appears for beneficiary/cause, instead of its default usage for instrument. Table 5 shows the label identification for some frequent postpositions.
2. Better handling of non-finite verbs in the 1st stage
3. Better handling of NULL nodes in the 2nd stage. Most NULL nodes are cases of ellipses where a syntactic heads such as finite verb or a conjunct is missing. Most of these cases fall into 2nd stage and are being better handled there.
4. Better handling of some 2nd stage specific constructions, e.g. clausal complements.

Closely related to the above four points is the performance of the clausal setups with respect to arc length, depth and non-projectivity. It is known that Malt suffers on the dependencies closer to the root (less depth) due to error-propagation. Also, Malt suffers at long distance dependencies because of local feature optimization (McDonald and Nivre, 2007). In other words, for Malt, depth and errors are negatively

correlated while arc-distance and errors are positively correlated.

Figure 7 shows the LAS of relations at various arc-lengths for the Baseline and clausal setups. Figure 8 shows the performance of relations at different depths. The 2nd stage of 2-Hard considers the heads of the partial trees produced by the 1st stage as the nodes (minimal parsing unit), which reduces the arc-length of the inter-clausal dependencies. Hence, as the arc-length increases, the advantage of 2-Hard becomes more pronounced.

Postposition	Baseline	2-Hard/ 2-Soft
0		✓
<i>meM</i>		✓
<i>para</i>	✓	
<i>GEN</i>		✓
<i>ko</i>	✓	
<i>se</i>		✓

Table 5: Label identification comparison between Baseline and the clausal approaches for ambiguous postpositions. ✓ signifies better performance. 0 and GEN signify null postposition and genitive postpositions respectively

By distinguishing intra-clausal structures from inter-clausal structures, the 2-Hard setup is using shallower trees and is able to take better global decisions by using more contextual information. It is expected to reduce the error propagation for the low-depth dependency relations. This effect is clearly seen in Figure 8, where for less depth 2-Hard outperforms Baseline. Cases (3) and (4) above reflect this.

Cases (1) and (2) on the other hand show that the clausal setups also effects 1st stage performance by learning verbal arguments (both complements and adjuncts) better. It is known that MaltParser has a rich feature representation but with increase in sentence length its performance gets affected due error propagation. By treating a clause as a parsing unit we reduce this error propagation as the features are being exploited properly.

It was found that both the clausal setups did not help in reducing the non-projective relations. As all the setups use the Arc-Eager parsing algorithm, they fare equally badly in handling non-projectivity. There were some sentences where non-projectivity got removed in the 1st stage,

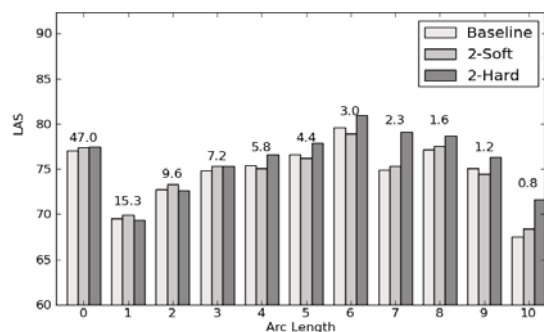


Fig. 7. LAS at arc-length (1-10) for Baseline, 2-Soft and 2-Hard. The numbers above the bars represent the % of relations at respective arc lengths.

however the non-projective arc reappeared in the 2nd stage, this happened in the case of paired connective constructions (cf., Mannem et al., 2009). We are yet to investigate if pseudo-projective parsing in the 2nd stage might prove beneficial in such cases.

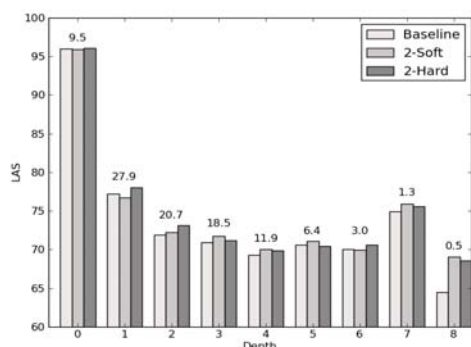


Fig. 8. LAS at depth (1-7) for Baseline, 2-Soft and 2-Hard. The numbers above the bars represent the % of relations at respective depths.

7 Conclusion

This paper investigated clausal data-driven dependency parsing. We implemented this idea using two methods, one based on hard constraints (similar to the one used in constraint-based parsing), and one based on soft constraints (using a kind of parser stacking). Our results showed that the approach using hard constraints seems most promising and performs significantly better than single-stage parsing. We showed that 2-Hard benefits from parsing shallower trees, and shorter arc lengths when compared to the Baseline. We also showed that by better learning of features many case markers that appear with more than one relation can be disambiguated successfully using both 2-Hard and 2-Soft. 2-Hard seems to perform better than 2-Soft in case

of inter-clausal relations w.r.t. all the evaluation metrics, whereas 2-Soft is doing good in intra-clausal relations. This gives us a future direction to explore a combination of 2-Hard and 2-Soft for inter and intra-clausal relations respectively, to see if one can benefit from the other.

Since the improvement in LS and LAS in both 2-Hard and 2-Soft seems to be more than in the UAS, it would be interesting to see the effect of clausal parsing on label identification and attachments separately. To do this, we plan to explore sequential parsing by using different feature models for transitions and labels, as the current parsing schemes do both attachments and labels at the same time.

References

- B.R. Ambati, S. Husain, J. Nivre, R. Sangal. 2010. On the Role of Morphosyntactic Features in Hindi Dependency Parsing. In *NAACL-HLT 2010 workshop on SPMRL*, Los Angeles, CA.
- R. Begum, S. Husain, A. Dhvaj, D. Sharma, L. Bai, R. Sangal. 2008. Dependency annotation scheme for Indian languages. In *IJCNLP08*.
- A. Bharati, S. Husain, D. Misra, R. Sangal 2009. Two stage constraint based hybrid approach to free word order language dependency parsing. In *the 11th IWPT09*. Paris.
- A. Bharati, S. Husain, B. Ambati, S. Jain, D. Sharma, R. Sangal. 2008. Two semantic features make all the difference in parsing accuracy. In *ICON08*.
- G. Eryigit, J. Nivre, K. Oflazer. 2008. Dependency Parsing of Turkish. *Computational Linguistics* 34(3), 357-389.
- P. Gadde, K. Jindal, S. Husain, D. Sharma, R. Sangal. 2010. Improving Data Driven Dependency Parsing using Clausal Information. In *NAACL-HLT 2010*, Los Angeles, CA.
- Y. Goldberg, M. Elhadad. 2009. Hebrew Dependency Parsing: Initial Results. In *the 11th IWPT09*. Paris.
- J. Hall, J. Nilsson, J. Nivre, G. Eryigit, B. Megyesi, M. Nilsson M. Saers. 2007. Single Malt or Blended? A Study in Multilingual Parser Optimization. In *EMNLP-CoNLL shared task*.
- S. Husain, P. Mannem, B. Ambati, P. Gadde. 2010. The ICON-2010 Tools Contest on Indian Language Dependency Parsing. 2010. In *ICON10 NLP Tools Contest on Indian Language Dependency Parsing*. Kharagpur, India.
- S. Husain. 2009. Dependency Parsers for Indian Languages. In *Proceedings of ICON09 NLP Tools Contest: Indian Language Dependency Parsing*. Hyderabad, India.

- S. Husain, P. Gadde, B. Ambati, D. Sharma, R. Sangal. 2009. A modular cascaded approach to complete parsing. In *the COLIPS International Conference on Asian Language Processing (IALP)* Singapore.
- P. Mannem, H. Chaudhry, A. Bharati. 2009. Insights into non-projectivity in Hindi. In *ACL-IJCNLP Student Research Workshop*.
- P. Mannem, A. Abhilash, A. Bharati. 2009b. LTAG-spinal Treebank and Parser for Hindi. In *ICON09*.
- A. F. Martins, D. Das, N. A. Smith, E. P. Xing. 2009. Stacking dependency parsers. In *EMNLP*.
- R. McDonald, F. Pereira, K. Ribarov, J. Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *HLT/EMNLP*.
- R. McDonald, J. Nivre. 2007. Characterizing the Errors of Data-Driven Dependency Parsing Models. In *EMNLP/CONLL*.
- J. Nilsson and J. Nivre. 2008. Malteval: An evaluation and visualization tool for dependency parsing. In *the Sixth LREC*, Marrakech, Morocco.
- J. Nivre. 2009. Non-Projective Dependency Parsing in Expected Linear Time. In *ACL-IJCNLP*.
- J. Nivre, R. McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *ACL-HLT*.
- J. Nivre, J. Hall, J. Nilsson. 2006. MaltParser: A Data-Driven Parser-Generator for Dependency Parsing. In *LREC*.
- J. Nivre. 2003. An efficient algorithm for projective dependency parsing. In *the 8th International Workshop on Parsing Technologies (IWPT)*.
- D. Seddah, M. Candito, B. Crabbé. 2009. Cross parser evaluation : a French Treebanks study. In *the 11th IWPT09*. Paris.
- K. Sagae, A. Lavie. 2006. Parser combination by re-parsing. In *the human Language Technology Conference of the NAACL*.
- R. Tsarfaty, K. Sima'an. 2008. Relational-Realizational Parsing. In *the 22nd COLING*. Manchester, UK.
- R. Tsarfaty, D. Seddah, Y. Goldberg, S. Kuebler, Y. Versley, M. Candito, J. Foster, I. Rehbein, L. Tounsi. 2010. What, How and Wither. In *NAACL-HLT 2010 workshop on SPMRL*.
- D. Zeman, Z. Zabokrtsky. 2005. Improving parsing accuracy by combining diverse dependency parsers. In *the 9th IWPT*.