Two methods to incorporate local morphosyntactic features in Hindi dependency parsing

Bharat Ram Ambati, Samar Husain, Sambhav Jain, Dipti Misra Sharma and Rajeev Sangal
Language Technologies Research Centre, IIIT-Hyderabad, India - 500032.
{ambati,samar}@research.iiit.ac.in, sambhav-jain@students.iiit.ac.in,{dipti,sangal}@mail.iiit.ac.in

Abstract

In this paper we explore two strategies to incorporate local morphosyntactic features in Hindi dependency parsing. These features are obtained using a shallow parser. We first explore which information provided by the shallow parser is most beneficial and show that local morphosyntactic features in the form of chunk type, head/non-head information, chunk boundary information, distance to the end of the chunk and suffix concatenation are very crucial in Hindi dependency parsing. We then investigate the best way to incorporate this information during dependency parsing. Further, we compare the results of various experiments based on various criterions and do some error analysis. All the experiments were done with two data-driven parsers, MaltParser and MSTParser, on a part of multi-layered and multi-representational Hindi Treebank which is under development. This paper is also the first attempt at complete sentence level parsing for Hindi.

1 Introduction

The dependency parsing community has since a few years shown considerable interest in parsing morphologically rich languages with flexible word order. This is partly due to the increasing availability of dependency treebanks for such languages, but it is also motivated by the observation that the performance obtained for these languages have not been very high (Nivre et al., 2007a). Attempts at handling various non-configurational aspects in these languages have pointed towards shortcomings in traditional parsing methodologies (Tsarfaty and Sima'an, 2008; Eryigit et al., 2008; Seddah et al., 2009; Husain et al., 2009; Gadde et al., 2010). Among other things, it has been pointed out that the use of language specific features may play a crucial role in improving the overall parsing performance. Different languages tend to encode syntactically relevant information in different ways, and it has been hypothesized that the integration of morphological and syntactic information could be a key to better accuracy. However, it has also been noted that incorporating these language specific features in parsing is not always straightforward and many intuitive features do not always work in expected ways.

In this paper we explore various strategies to incorporate local morphosyntactic features in Hindi dependency parsing. These features are obtained using a shallow parser. We conducted experiments with two data-driven parsers, MaltParser (Nivre et al., 2007b) and MSTParser (McDonald et al., 2006). We first explore which information provided by the shallow parser is most beneficial and show that local morphosyntactic features in the form of chunk type, head/non-head information, chunk boundary information, distance to the end of the chunk and suffix concatenation are very crucial in Hindi dependency parsing. We then investigate the best way to incorporate this information during dependency parsing. All the experiments were done on a part of multi-layered and multi-representational Hindi Treebank (Bhatt et al., 2009)[1].

The shallow parser performs three tasks, (a) it gives the POS tags for each lexical item, (b) provides morphological features for each lexical item, and (c) performs chunking. A chunk is a minimal (non-recursive) phrase consisting of correlated, inseparable words/entities, such that the intra-chunk dependencies are not distorted (Bharati et

[1] This Treebank is still under development. There are currently 27k tokens with complete sentence level annotation.

Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages, pages 22–30,
Los Angeles, California, June 2010.

al., 2006). Together, a group of lexical items with some POS tag and morphological features within a chunk can be utilized to automatically compute local morphosyntactic information. For example, such information can represent the postposition/case-marking in the case of noun chunks, or it may represent the tense, aspect and modality (TAM) information in the case of verb chunks. In the experiments conducted for this paper such local information is automatically computed and incorporated as a feature to the head of a chunk. In general, local morphosyntactic features correspond to all the parsing relevant local linguistic features that can be utilized using the notion of chunk. Previously, there have been some attempts at using chunk information in dependency parsing. Attardi and Dell'Orletta (2008) used chunking information in parsing English. They got an increase of 0.35% in labeled attachment accuracy and 0.47% in unlabeled attachment accuracy over the state-of-the-art dependency parser.

Among the three components (a-c, above), the parsing accuracy obtained using the POS feature is taken as baseline. We follow this by experiments where we explore how each of morph and chunk features help in improving dependency parsing accuracy. In particular, we find that local morphosyntactic features are the most crucial. These experiments are discussed in section 2. In section 3 we will then see an alternative way to incorporate the best features obtained in section 2. In all the parsing experiments discussed in section 2 and 3, at each step we explore all possible features and extract the best set of features. Best features of one experiment are used when we go to the next set of experiments. For example, when we explore the effect of chunk information, all the relevant morph information from previous set of experiments is taken into account.

This paper is also the first attempt at complete sentence level parsing for Hindi. Due to the availability of dependency treebank for Hindi (Begum et al., 2008), there have been some previous attempts at Hindi data-driven dependency parsing (Bharati et al., 2008; Mannem et al., 2009; Husain et al., 2009). Recently in ICON-09 NLP Tools Contest (Husain, 2009; and the references therein), rule-based, constraint based, statistical and hybrid approaches were explored for dependency parsing. Previously, constraint based approaches to Indian language (IL) dependency parsing have also been explored (Bharati et al., 1993, 1995, 2009b, 2009c). All these attempts, however, were finding inter-chunk dependency relations, given gold-standard POS and chunk tags. Unlike these previous parsers, the dependencies in this work are between lexical items, i.e. the dependency tree is complete.

The paper is arranged as follows, in section 2 and 3, we discuss the parsing experiments. In section 4, we describe the data and parser settings. Section 5 gives the results and discusses some related issues. General discussion and possible future work is mentioned in section 6. We conclude the paper in section 7.

2 Getting the best linguistic features

As mentioned earlier, a shallow parser consists of three main components, (a) POS tagger, (b) morphological analyzer and (c) chunker. In this section we systematically explore what is the effect of each of these components. We'll see in section 2.3 that the best features of a-c can be used to compute local morphosyntactic features that, as the results show, are extremely useful.

2.1 Using POS as feature (PaF):

In this experiment we only use the POS tag information of individual words during dependency parsing. First a raw sentence is POS-tagged. This POS-tagged sentence is then given to a parser to predict the dependency relations. Figure 1, shows the steps involved in this approach for (1).

(1) raama ne eka seba khaayaa
'Ram' ERG 'one' 'apple' 'ate'
'Ram ate an apple'

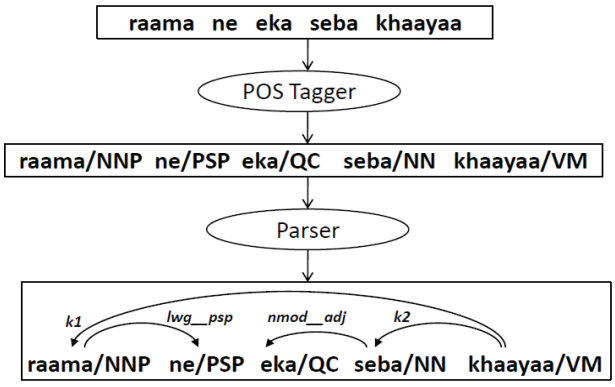

Figure 1: Dependency parsing using only POS information from a shallow parser.

In (1) above, 'NN', 'PSP', 'QC', 'NN' and 'VM' are the POS tags[2] for *raama, ne, eka, seba* and *khaayaa* respectively. This information is provided as a feature to the parser. The result of this experiment forms our baseline accuracy.

2.2 Using Morph as feature (MaF):

In addition to POS information, in this experiment we also use the morph information for each token. This morphological information is provided as a feature to the parser. Morph has the following information

- Root: Root form of the word
- Category: Course grained POS
- Gender: Masculine/Feminine/Neuter
- Number: Singular/Plural
- Person: First/Second/Third person
- Case: Oblique/Direct case
- Suffix: Suffix of the word

Take *raama* in (1), its morph information comprises of root = '*raama*', category = 'noun' gender = 'masculine', number = 'singular', person = 'third', case = 'direct', suffix = '0'. Similarly, *khaayaa* ('ate') has the following morph information. root = '*khaa*', category = 'verb' gender = 'masculine', numer = 'singular', person = 'third', case = 'direct', suffix = 'yaa'.

Through a series of experiments, the most crucial morph features were selected. Root, case and suffix turn out to be the most important features. Results are discussed in section 5.

2.3 Using local morphosyntax as feature (LMSaF)

Along with POS and the most useful morph features (root, case and suffix), in this experiment we also use local morphosyntactic features that reflect various chunk level information. These features are:

- Type of the chunk
- Head/non-head of the chunk
- Chunk boundary information
- Distance to the end of the chunk
- Suffix concatenation

In example 1 (see section 2.1), there are two noun chunks and one verb chunk. *raama* and *seba* are the heads of the noun chunks. *khaayaa* is the head of the verb chunk. We follow standard IOB[3] notation for chunk boundary. *raama*, *eka* and *khaayaa* are at the beginning (B) of their respective chunks. *ne* and *seba* are inside (I) their respective chunks. *raama* is at distance 1 from the end of the chunk and *ne* is at a distance 0 from the end of the chunk.

Once we have a chunk and morph feature like suffix, we can perform suffix concatenation automatically. A group of lexical items with some POS tags and suffix information within a chunk can be utilized to automatically compute this feature. This feature can, for example, represent the postposition/case-marking in the case of noun chunk, or it may represent the tense, aspect and modality (TAM) information in the case of verb chunks. Note that, this feature becomes part of the lexical item that is the head of a chunk. Take (2) as a case in point:

(2) [NP raama/NNP ne/PSP] [NP seba/NN]
'Ram' ERG 'apple'
[VGF khaa/VM liyaa/VAUX]
'eat' 'PRFT'
'Ram ate an apple'

The suffix concatenation feature for *khaa,* which is the head of the VGF chunk, will be '0+yaa' and is formed by concatenating the suffix of the main verb with that of its auxiliary. Similarly, the suffix concatenation feature for *raama,* which is head of the NP chunk, will be '0+ne'. This feature turns out to be very important. This is because in Hindi (and many other Indian languages) there is a direct correlation between the TAM markers and the case that appears on some nominals (Bharati et al., 1995). In (2), for example, *khaa liyaa* together gives the past perfective aspect for the verb *khaanaa* 'to eat'. Since, Hindi is split ergative, the subject of the transitive verb takes an ergative case marker when the verb is past perfective. Similar

[2] NN: Common noun, PSP: Post position, QC: Cardinal, VM: Verb. A list of complete POS tags can be found here: http://ltrc.iiit.ac.in/MachineTrans/research/tb/POS-Tag-List.pdf. The POS/chunk tag scheme followed in the Treebank is described in Bharati et al. (2006).

[3] Inside, Outside, Beginning of the chunk.

correlation between the case markers and TAM exist in many other cases.

3 An alternative approach to use best features: A 2-stage setup (2stage)

So far we have been using various information such as POS, chunk, etc. as features. Rather than using them as features and doing parsing at one go, we can alternatively follow a 2-stage setup. In particular, we divide the task of parsing into:

- Intra-chunk dependency parsing
- Inter-chunk dependency parsing

We still use POS, best morphological features (case, suffix, root) information as regular features during parsing. But unlike LMSaF mentioned in section 2.3, where we gave local morphosyntactic information as a feature, we divided the task of parsing into sub-tasks. A similar approach was also proposed by Bharati et al. (2009c). During intra-chunk dependency parsing, we try to find the dependency relations of the words within a chunk. Following which, chunk heads of each chunk within a sentence are extracted. On these chunk heads we run an inter-chunk dependency parser. For each chunk head, in addition to POS tag, useful morphological features, any useful intra-chunk information in the form of lexical item, suffix concatenation, dependency relation are also given as a feature.

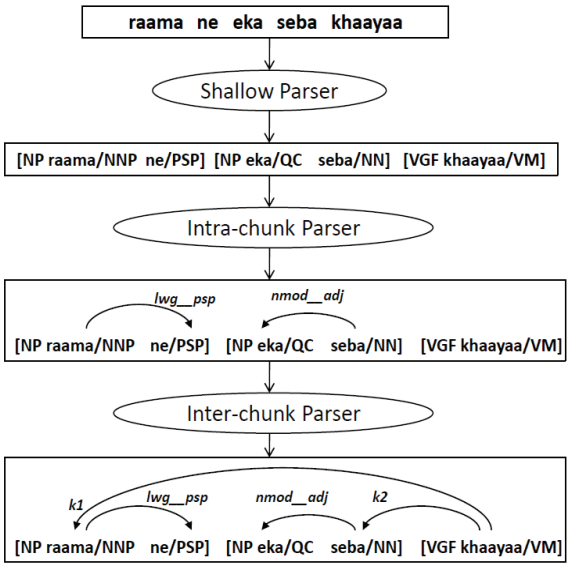

Figure 2: Dependency parsing using chunk information: 2-stage approach.

Figure 2 shows the steps involved in this approach for (1). There are two noun chunks and one verb chunk in this sentence. *raama* and *seba* are the heads of the noun chunks. *khaaya* is the head of the verb chunk. The intra-chunk parser attaches *ne* to *raama* and *eka* to *seba* with dependency labels 'lwg__psp' and 'nmod__adj'[4] respectively. Heads of each chunk along with its POS, morphological features, local morphosyntactic features and intra-chunk features are extracted and given to inter-chunk parser. Using this information the inter-chunk dependency parser marks the dependency relations between chunk heads. *khaaya* becomes the root of the dependency tree. *raama* and *seba* are attached to *khaaya* with dependency labels 'k1' and 'k2'[5] respectively.

4 Experimental Setup

In this section we describe the data and the parser settings used for our experiments.

4.1 Data

For our experiments we took 1228 dependency annotated sentences (27k tokens), which have complete sentence level annotation from the new multi-layered and multi-representational Hindi Treebank (Bhatt et al., 2009). This treebank is still under development. Average length of these sentences is 22 tokens/sentence and 10 chunks/sentence. We divided the data into two sets, 1000 sentences for training and 228 sentences for testing.

4.2 Parsers and settings

All experiments were performed using two data-driven parsers, MaltParser[6] (Nivre et al., 2007b), and MSTParser[7] (McDonald et al., 2006).

[4] nmod__adj is an intra-chunk label for quantifier-noun modification. lwg__psp is the label for post-position marker. Details of the labels can be seen in the intra-chunk guidelines http://ltrc.iiit.ac.in/MachineTrans/research/tb/IntraChunk-Dependency-Annotation-Guidelines.pdf

[5] k1 (karta) and k2 (karma) are syntactico-semantic labels which have some properties of both grammatical roles and thematic roles. k1 behaves similar to subject and agent. k2 behaves similar to object and patient (Bharati et al., 1995; Vaidya et al., 2009). For complete tagset, see (Bharati et al., 2009).

[6] Malt Version 1.3.1

[7] MST Version 0.4b

	Malt						MST+MaxEnt					
	Cross-validation			Test-set			Cross-validation			Test-set		
	UAS	*LAS*	*LS*	*UAS*	*LAS*	*LS*	*UAS*	*LAS*	*LS*	*UAS*	*LAS*	*LS*
PaF	89.4	78.2	80.5	90.4	80.1	82.4	86.3	75.1	77.9	87.9	77.0	79.3
MaF	89.6	80.5	83.1	90.4	81.7	84.1	89.1	79.2	82.5	90.0	80.9	83.9
LMSaF	91.5	82.7	84.7	91.8	84.0	86.2	90.8	79.8	82.0	92.0	81.8	83.8
2stage	**91.8**	**83.3**	**85.3**	**92.4**	**84.4**	**86.3**	**92.1**	**82.2**	**84.3**	**92.7**	**84.0**	**86.2**

Table 1: Results of all the four approaches using gold-standard shallow parser information.

Malt is a classifier based shift/reduce parser. It provides option for six parsing algorithms, namely, arc-eager, arc-standard, convington projective, covington non-projective, stack projective, stack eager and stack lazy. The parser also provides option for libsvm and liblinear learning model. It uses graph transformation to handle non-projective trees (Nivre and Nilsson, 2005). MST uses Chu-Liu-Edmonds (Chu and Liu, 1965; Edmonds, 1967) Maximum Spanning Tree algorithm for non-projective parsing and Eisner's algorithm for projective parsing (Eisner, 1996). It uses online large margin learning as the learning algorithm (McDonald et al., 2005). In this paper, we use MST only for unlabeled dependency tree and use a separate maximum entropy model[8] (MaxEnt) for labeling. Various combination of features such as node, its parent, siblings and children were tried out before arriving at the best results.

As the training data size is small we did 5-fold cross validation on the training data for tuning the parameters of the parsers and for feature selection. Best settings obtained using cross-validated data are applied on test set. We present the results both on cross validated data and on test data.

For the Malt Parser, arc-eager algorithm gave better performance over others in all the approaches. Libsvm consistently gave better performance over liblinear in all the experiments. For SVM settings, we tried out different combinations of best SVM settings of the same parser on different languages in CoNLL-2007 shared task (Hall et al., 2007) and applied the best settings. For feature model, apart from trying best feature settings of the same parser on different languages in CoNLL-2007 shared task (Hall et al., 2007), we also tried out different combinations of linguistically intuitive features and applied the best feature model. The best feature model is same as the feature model used in Ambati et al. (2009a), which is the best performing system in the ICON-2009 NLP Tools Contest (Husain, 2009).

For the MSTParser, non-projective algorithm, order=2 and training-k=5 gave best results in all the approaches. For the MaxEnt, apart from some general useful features, we experimented considering different combinations of features of node, parent, siblings, and children of the node.

5 Results and Analysis

All the experiments discussed in section 2 and 3 were performed considering both gold-standard shallow parser information and automatic shallow parser[9] information. Automatic shallow parser uses a rule based system for morph analysis, a CRF+TBL based POS-tagger and chunker. The tagger and chunker are 93% and 87% accurate respectively. These accuracies are obtained after using the approach of PVS and Gali, (2007) on larger training data. In addition, while using automatic shallow parser information to get the results, we also explored using both gold-standard and automatic information during training. As expected, using automatic shallow parser information for training gave better performance than using gold while training.

Table 1 and Table 2 shows the results of the four experiments using gold-standard and automatic shallow parser information respectively. We evaluated our experiments based on unlabeled attachment score (UAS), labeled attachment score (LAS) and labeled score (LS) (Nivre et al., 2007a). Best LAS on test data is 84.4% (with 2stage) and 75.4% (with LMSaF) using gold and automatic shallow parser information respectively. These results are obtained using MaltParser. In the following subsection we discuss the results based on different criterion.

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

[9] http://ltrc.iiit.ac.in/analyzer/hindi/

	Malt						MST+MaxEnt					
	Cross-validation			Test-set			Cross-validation			Test-set		
	UAS	*LAS*	*LS*	*UAS*	*LAS*	*LS*	*UAS*	*LAS*	*LS*	*UAS*	*LAS*	*LS*
PaF	82.2	69.3	73.4	84.6	72.9	76.5	79.4	66.5	70.7	81.6	69.4	73.1
MaF	82.5	71.6	76.1	84.0	73.6	77.6	82.3	70.4	75.4	83.4	72.7	77.3
LMSaF	**83.2**	**73.0**	**77.0**	**85.5**	**75.4**	**78.9**	**82.6**	**71.3**	**76.1**	**85.0**	**73.4**	**77.3**
2stage	79.0	69.5	75.6	79.6	71.1	76.8	78.8	66.6	72.6	80.1	69.7	75.4

Table 2: Results of all the four experiments using automatic shallow parser information.

POS tags provide very basic linguistic information in the form of broad grained categories. The best LAS for PaF while using gold and automatic tagger were 80.1% and 72.9% respectively. The morph information in the form of case, suffix and root information proved to be the most important features. But surprisingly, gender, number and person features didn't help. Agreement patterns in Hindi are not straightforward. For example, the verb agrees with k2 if the k1 has a post-position; it may also sometimes take the default features. In a passive sentence, the verb agrees only with k2. The agreement problem worsens when there is coordination or when there is a complex verb. It is understandable then that the parser is unable to learn the selective agreement pattern which needs to be followed.

LMSaF on the other hand encode richer information and capture some local linguistic patterns. The first four features in LMSaF (chunk type, chunk boundary, head/non-head of chunk and distance to the end of chunk) were found to be useful consistently. The fifth feature, in the form of suffix concatenation, gave us the biggest jump, and captures the correlation between the TAM markers of the verbs and the case markers on the nominals.

5.1 Feature comparison: PaF, MaF vs. LMSaF

Dependency labels can be classified as two types based on their nature, namely, inter-chunk dependency labels and intra-chunk labels. Inter-chunk dependency labels are syntacto-semantic in nature. Whereas intra-chunk dependency labels are purely syntactic in nature.

Figure 3, shows the f-measure for top six inter-chunk and intra-chunk dependency labels for PaF, MaF, and LMSaF using Maltparser on test data using automatic shallow parser information. The first six labels (k1, k2, pof, r6, ccof, and k7p) are the top six inter-chunk labels and the next six labels (lwg__psp, lwg__aux, lwg__cont, rsym, nmod__adj, and pof__cn) are the top six intra-chunk labels. First six labels (inter-chunk) correspond to 28.41% and next six labels (intra-chunk) correspond to 48.81% of the total labels in the test data. The figure shows that with POS information alone, f-measure for top four intra-chunk labels reached more than 90% accuracy. The accuracy increases marginally with the addition of morph and local morphosytactic features. The results corroborates with our intuition that intra-chunk dependencies are mostly syntactic. For example, consider an intra-chunk label 'lwg__psp'. This is the label for postposition marker. A post-position marker succeeding a noun is attached to that noun with the label 'lwg__psp'. POS tag for post-position marker is PSP. So, if a NN (common noun) or a NNP (proper noun) is followed by a PSP (post-position marker), then the PSP will be attached to the preceding NN/NNP with the dependency label 'lwg_psp'. As a result, providing POS information itself gave an f-measure of 98.3% for 'lwg_psp'. With morph and local morphosytactic features, this got increased to 98.4%. However, f-measure for some labels like 'nmod__adj' is around 80% only. 'nmod__adj' is the label for adjective-noun, quantifier-noun modifications. Low accuracy for these labels is mainly due to two reasons. One is POS tag errors. And the other is attachment errors due to genuine ambiguities such as compounding.

For inter-chunk labels (first six columns in the figure 3), there is considerable improvement in the f-measure using morph and local morphosytactic features. As mentioned, local morphosyntactic features provide local linguistic information. For example, consider the case of verbs. At POS level, there are only two tags 'VM' and 'VAUX' for main verbs and auxiliary verbs respectively (Bharati et al., 2006). Information about finite/non-finiteness is not present in the POS tag. But, at chunk level there are four different chunk tags for

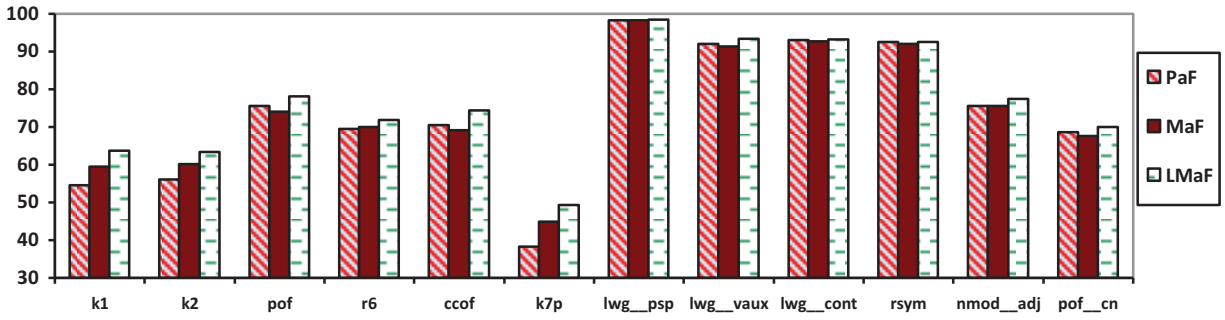

Figure 3: F-measure of top 6, inter-chunk and intra-chunk labels for PaF, MaF and LMSaF approaches using Malt-parser on test data using automatic shallow parser information.

verbs, namely VGF, VGNF, VGINF and VGNN. They are respectively, finite, non-finite, infinitival and gerundial chunk tags. The difference in the verbal chunk tag is a good cue for helping the parser in identifying different syntactic behavior of these verbs. Moreover, a finite verb can become the root of the sentence, whereas a non-finite or infinitival verb can't. Thus, providing chunk information also helped in improving the correct identification of the root of the sentence.

Similar to Prague Treebank (Hajicova, 1998), coordinating conjuncts are heads in the treebank that we use. The relation between a conjunct and its children is shown using 'ccof' label. A coordinating conjuct takes children of similar type only. For example, a coordinating conjuct can have two finite verbs or two non-finite verbs as its children, but not a finite verb and a non-finite verb. Such instances are also handled more effectively if chunk information is incorporated. The largest increase in performance, however, was due to the 'suffix concatenation' feature. Significant improvement in the core inter-chunk dependency labels (such as k1, k2, k4, etc.) due to this feature is the main reason for the overall improvement in the parsing accuracy. As mentioned earlier, this is because this feature captures the correlation between the TAM markers of the verbs and the case markers on the nominals.

5.2 Approach comparison: LMSaF vs. 2stage

Both LMSaF and 2stage use chunk information. In LMSaF, chunk information is given as a feature whereas in 2stage, sentence parsing is divided into intra-chunk and inter-chunk parsing. Both the approaches have their pros and cons. In LMSaF as everything is done in a single stage there is much richer context to learn from. In 2stage, we can provide features specific to each stage which can't be done in a single stage approach (McDonald et al., 2006). But in 2stage, as we are dividing the task, accuracy of the division and the error propagation might pose a problem. This is reflected in the results where the 2-stage performs better than the single stage while using gold standard information, but lags behind considerably when the features are automatically computed.

During intra-chunk parsing in the 2stage setup, we tried out using both a rule-based approach and a statistical approach (using MaltParser). The rule based system performed slightly better (0.1% LAS) than statistical when gold chunks are considered. But, with automatic chunks, the statistical approach outperformed rule-based system with a difference of 7% in LAS. This is not surprising because, the rules used are very robust and mostly based on POS and chunk information. Due to errors induced by the automatic POS tagger and chunker, the rule-based system couldn't perform well. Consider a small example chunk given below.

((		NP
meraa	'my'	PRP
bhaaii	'brother'	NN
))		

As per the Hindi chunking guidelines (Bharati et al., 2006), *meraa* and *bhaaii* should be in two separate chunks. And as per Hindi dependency annotation guidelines (Bharati et al., 2009), *meraa* is attached to *bhaaii* with a dependency label 'r6'[10]. When the chunker wrongly chunks them in a single

[10]'r6' is the dependency label for genitive relation.

chunk, intra-chunk parser will assign the dependency relation for *meraa*. Rule based system can never assign 'r6' relation to *meraa* as it is an inter-chunk label and the rules used cannot handle such cases. But in a statistical system, if we train the parser using automatic chunks instead of gold chunks, the system can potentially assign 'r6' label.

5.3 Parser comparison: MST vs. Malt

In all the experiments, results of MaltParser are consistently better than MST+MaxEnt. We know that Maltparser is good at short distance labeling and MST is good at long distance labeling (McDonald and Nivre, 2007). The root of the sentence is better identified by MSTParser than MaltParser. Our results also confirm this. MST+MaxEnt and Malt could identify the root of the sentence with an f-measure of 89.7% and 72.3% respectively. Presence of more short distance labels helped Malt to outperform MST. Figure 5, shows the f-measure relative to dependency length for both the parsers on test data using automatic shallow parser information for LMSaF.

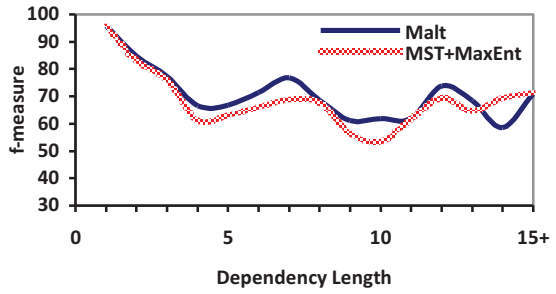

Figure 5: Dependency arc f-measure relative to dependency length.

6 Discussion and Future Work

We systematically explored the effect of various linguistic features in Hindi dependency parsing. Results show that POS, case, suffix, root, along with local morphosyntactic features help dependency parsing. We then described 2 methods to incorporate such features during the parsing process. These methods can be thought as different paradigms of modularity. For practical reasons (i.e. given the POS tagger/chunker accuracies), it is wiser to use this information as features rather than dividing the task into two stages.

As mentioned earlier, this is the first attempt at complete sentence level parsing for Hindi. So, we cannot compare our results with previous attempts at Hindi dependency parsing, due to, (a) The data used here is different and (b) we produce complete sentence parses rather than chunk level parses.

As mentioned in section 5.1, accuracies of intra-chunk dependencies are very high compared to inter-chunk dependencies. Inter-chunk dependencies are syntacto-semantic in nature. The parser depends on surface syntactic cues to identify such relations. But syntactic information alone is always not sufficient, either due to unavailability or due to ambiguity. In such cases, providing some semantic information can help in improving the inter-chunk dependency accuracy. There have been attempts at using minimal semantic information in dependency parsing for Hindi (Bharati et al., 2008). Recently, Ambati et al. (2009b) used six semantic features namely, human, non-human, in-animate, time, place, and abstract for Hindi dependency parsing. Using gold-standard semantic features, they showed considerable improvement in the core inter-chunk dependency accuracy. Some attempts at using clause information in dependency parsing for Hindi (Gadde et al., 2010) have also been made. These attempts were at inter-chunk dependency parsing using gold-standard POS tags and chunks. We plan to see their effect in complete sentence parsing using automatic shallow parser information also.

7 Conclusion

In this paper we explored two strategies to incorporate local morphosyntactic features in Hindi dependency parsing. These features were obtained using a shallow parser. We first explored which information provided by the shallow parser is useful and showed that local morphosyntactic features in the form of chunk type, head/non-head info, chunk boundary info, distance to the end of the chunk and suffix concatenation are very crucial for Hindi dependency parsing. We then investigated the best way to incorporate this information during dependency parsing. Further, we compared the results of various experiments based on various criterions and did some error analysis. This paper was also the first attempt at complete sentence level parsing for Hindi.

References

B. R. Ambati, P. Gadde, and K. Jindal. 2009a. Experiments in Indian Language Dependency Parsing. In *Proc of the ICON09 NLP Tools Contest: Indian Language Dependency Parsing*, pp 32-37.

B. R. Ambati, P. Gade, C. GSK and S. Husain. 2009b. Effect of Minimal Semantics on Dependency Parsing**.** In *Proc of RANLP09 student paper workshop.*

G. Attardi and F. Dell'Orletta. 2008. Chunking and Dependency Parsing. In *Proc of LREC Workshop on Partial Parsing: Between Chunking and Deep Parsing.* Marrakech, Morocco.

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

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

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

A. Bharati, R. Sangal, D. M. Sharma and L. Bai. 2006. AnnCorra: Annotating Corpora Guidelines for POS and Chunk Annotation for Indian Languages. *Technical Report (TR-LTRC-31), LTRC, IIIT-Hyderabad.*

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 and R. Sangal. 2009b. Two stage constraint based hybrid approach to free word order language dependency parsing. *In Proc. of IWPT.*

A. Bharati, S. Husain, M. Vijay, K. Deepak, D. M. Sharma and R. Sangal. 2009c. Constraint Based Hybrid Approach to Parsing Indian Languages. *In Proc of PACLIC 23. Hong Kong. 2009.*

R. Bhatt, B. Narasimhan, M. Palmer, O. Rambow, D. M. Sharma and F. Xia. 2009. Multi-Representational and Multi-Layered Treebank for Hindi/Urdu. *In Proc. of the Third LAW at 47th ACL and 4th IJCNLP.*

Y.J. Chu and T.H. Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

J. Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.

J. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proc of COLING-96*, pp. 340–345.

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

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

E. Hajicova. 1998. Prague Dependency Treebank: From Analytic to Tectogrammatical Annotation. In *Proc of TSD'98.*

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 the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, 933—939.

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

S. Husain, P. Gadde, B. Ambati, D. M. Sharma and R. Sangal. 2009. A modular cascaded approach to complete parsing. In *Proc. of the COLIPS IALP.*

P. Mannem, A. Abhilash and A. Bharati. 2009. LTAG-spinal Treebank and Parser for Hindi. In *Proc of International Conference on NLP, Hyderabad. 2009*.

R. McDonald, K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *Proc of ACL*. pp. 91–98.

R. McDonald, K. Lerman, and F. Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proc of the Tenth (CoNLL-X)*, pp. 216–220.

R. McDonald and J. Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proc. of EMNLP-CoNLL.*

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.

J. Nivre and J. Nilsson. 2005. Pseudo-projective dependency parsing. In *Proc. of ACL-2005*, pp. 99–106.

Avinesh PVS and K. Gali. 2007. Part-Of-Speech Tagging and Chunking Using Conditional Random Fields and Transformation Based Learning. *In Proc of the SPSAL workshop during IJCAI '07.*

D. Seddah, M. Candito and B. Crabbé. 2009. Cross parser evaluation: a French Treebanks study. In *Proc. of IWPT,* 150-161.

R. Tsarfaty and K. Sima'an. 2008. Relational-Realizational Parsing. In *Proc. of CoLing,* 889-896.

A. Vaidya, S. Husain, P. Mannem, and D. M. Sharma. 2009. A karaka-based dependency annotation scheme for English**.** In *Proc. of CICLing,* 41-52.