A Rule Based Approach for Automatic Annotation of a Hindi Treebank

Mridul Gupta, Vineet Yadav, Samar Husain and Dipti M Sharma
Language Technologies Research Center,
IIIT Hyderabad, India.
{mridulgupta,vineetyadav}@students.iiit.ac.in,
{samar, dipti}@mail.iiit.ac.in

Abstract

The paper describes an approach to automatically annotate a Hindi Treebank using Paninian dependency framework. The annotator is a rule based system and the rules use certain syntactic cues available in a sentence. This automated annotation scheme aims at facilitating manual annotation by reducing time and effort of manual annotators. Also, the aim of automatic annotation, among other things, is to increase the efficiency of a broad coverage constraint based Hindi parser. We also evaluate this tool and show its accuracy and coverage.

1 Introduction

This paper describes a rule based effort designed to automatically annotate a Hindi treebank in an efficient and robust manner. There is a need for creating a large treebank for carrying out various tasks in NLP for Hindi and other Indian languages. The lack of such tree banks has been a major bottleneck in the development of good natural language tools and applications for Indian languages. Treebanks have been successfully created for various other languages like English (Marcus et. al, 1993), German (Brants and Skut, 1998), Czech (Hajicova, 1998), Portuguese (Bick, 2007), etc.

The automatic annotation done here is based on the Paninian grammatical model (Begum et. al, 2008; Bharati et. al, 1995) which is a dependency grammar (Kiparsky and Staal, 1969). Dependency frameworks have been employed for languages like Dutch (van der Beek et. al, 2002), Italian (Bosco and Lambardo, 2004), Czech (Hajicova, 1998), etc. It has been argued that dependency frameworks are well suited for morphologically rich and relatively free word order languages (Hudson, 1984; Mel'Cuk, 1988).

Automatic annotation tools have been built for languages like German (Brants and Skut, 1998), Portuguese (Bick, 2007), etc. Our aim in building such a rule based tool for automated annotation[1] is:

1. To facilitate the process of annotation for the manual annotators.
2. To correct and fine grain the output of a broad coverage constraint based parser.
3. To explore the extent and performance of a purely rule based system[2] in the field of parsing.

The system works on the rules formulated by exploiting certain syntactic cues such as post-positions, TAM (tense, aspect and modality) labels, POS (part of speech) labels, etc. In fact, the mapping of a *karaka*[3] with post-positions is pretty strong. There have been some previous attempts in exploring this mapping between a karaka and post-position. One such work (Begum et al., 2008) worked with 1400 sentences. It was found that some karaka labels bear striking correspondence with certain post positions which has since proved critical in designing a rule based system. The results of the above experiment have been shown in Table 1.1.

[1]The size of the Hindi treebank used for development and testing the tool is small in size.

[2]Note that since this is an ongoing work, the observations and results pertaining to purely rule-based parsing are in no way final.

[3]The elements modifying the verb participate in the action specified by the verb. These participant relations with the verb are called *karakas*. For a detailed analysis see Bharati et al. (1995). Many relations mentioned in this paper are described in Begum et al. (2008). For the complete tagset description, see http://ltrc.iiit.ac.in/MachineTrans/publications/technicalReports/tr032/treebank.pdf

Proceedings of ICON-2008: 6th International Conference on Natural Language Processing
Macmillan Publishers, India. Also accessible from http://ltrc.iiit.ac.in/proceedings/ICON-2008

Kr/V	kA	se	para	meM	ne	ko
k1	53	5	0	0	**344**	127
k2	174	70	5	5	0	**280**
k3	0	**51**	0	0	0	0
k4	0	9	0	0	0	**77**
k5	1	**97**	0	0	0	0
k7	1	4	38	**141**	0	0
k7p	45	11	72	**302**	0	0
k7t	13	12	6	**31**	0	7

Table 1.1.

As can be seen, the experiment clearly shows that the dependency relation 'k1' bears a strong mapping with the post position 'ne'. Similarly, k7(p/t) overwhelmingly map to 'meM'.

Our paper explores various ways to extend the experiment described above. The paper is arranged as follows; Section 2 briefly explains the annotation scheme. In Section 3 we describe the approach and the algorithm used. Section 4 describes the experiments conducted and their results. We discuss the results and show the error analysis in Section 5. Section 6 finally concludes the paper.

2 Annotation Scheme

As mentioned in Section 1, the annotation scheme used is based on the Paninian dependency framework. The reason for working under a dependency framework is the rich morphology and the relatively free word order of Indian languages. This framework defines the relations between a verb and its participants as karakas. A karaka is defined as a participant in an action. The karaka relations are syntactico-semantic[4] in nature.

There are six basic karakas and some other non-karaka relations. In total the annotation scheme has about 28 labels (Begum et al., 2008). The six karakas are:

k1 – *karta*, k2 – *karma*, k3 – *karana*, k4 – *sampradaan*, k5 – *apaadaan* and k7 – *adhikaran.*

Some of the non-karaka labels are r6 (possessive), rh (purpose), rt (cause), etc.

As mentioned earlier, we plan to identify these dependency relations using different robust strategies incorporated in a rule based system.

[4]k1 (karta) and k2 (karma) are syntactico-semantic labels which have some properties of both grammatical roles and thematic roles. k1 for example, behaves similar to subject and agent.

3 Approach

The approach mainly focuses on building a system which uses some set of rules for each dependency relation to annotate the corpus. We mark only the inter-chunk[5] dependency labels. Intra-chunk dependencies are not marked by the automatic annotation tool.

3.1 Corpus Description

The sentences for the task of annotation were obtained from the Hyderabad Dependency Treebank (HyDT) (Begum et al., 2008). The sentences were in the SSF format (Bharati et. al, 2005). In all there were 2053 Hindi sentences. The corpus had 41803 words and 6244 unique tokens. The average length of a sentence was 20.05.

This corpus was divided into a development data set and a testing data. Development data contained 1741 sentences and testing 312. Reference data was also used for evaluating the tool. The reference data had all the dependency labels marked.

3.2 Rules for Marking Dependency Relations

3.2.1 Relations marked

We mark only those dependency relations that have frequency higher than a certain threshold[6]. However, this set excludes relations like pof[7] and ccof[8] even though they are above the threshold value. A 'pof' relation generally occurs in the case of complex verb, coming up with rules to identify a complex verb in Hindi is rather difficult. 'ccof' also cannot be easily marked as its identification is more structural than lexical. Moreover, our aim is not to emulate the functions of a broad coverage parser entirely. These unmarked relations are best left to be marked by a parser, or in the case of manual annotation, by the manual annotators. In all we mark 15 dependency relations.

[5]In the treebank chunk head appear as nodes. A chunk is a set of adjacent words which are in dependency relation with each other, and are connected to the rest of the words by a single incoming arc to the chunk.

[6]The threshold set was 15.

[7]pof: part of

[8]ccof: conjuct of

3.2.2 Rule Format

Based on the experiments done and statistics collected as well as on the basis of certain linguistic cues, as described above, rules for identifying the dependency relations were formed.

Each rule follows a specific format. A rule is an 8-tuple containing eight fields, these are:

1. Modified Group i.e., the parent chunk,
2. Modified Constraints,
3. Modifier Group, i.e., the child chunk,
4. Modifier Category,
5. Relation,
6. Dependence of the relation on another relation,
7. Multiplicity of the relation,
8. Weight of the rule.

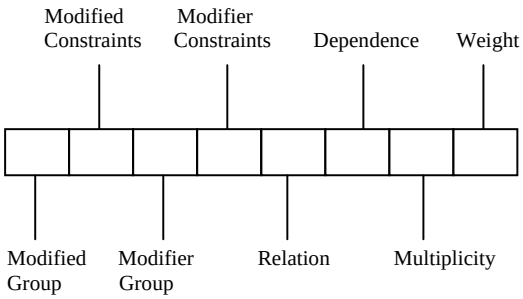

Figure 3.1.

The 6th field (dependence) signifies the dependence of one relation on the other i.e., the presence of an already marked relation within the clause boundary (Clause boundary is explained in Section 3.4.1). Multiplicity indicates the possible number of occurrences of a dependency relation within a clause boundary.

A non-negative weight is associated with each rule. Higher the rule weight, higher its priority over other rules. But, a '0' weight indicates that the priority of the rule over other rules for the same relation does not matter. These weights have been assigned to the rules after conducting some experiments. In these experiments we found that some rules are very robust and hence were assigned weights in the decreasing order of their robustness. An example of a rule is stated below:

VG tam=ko NP vib[9]=ko reln=k1 dep=X mult=1 weight=4

The above rule states that there is a modifier chunk NP that possesses the post position or vibhakti 'ko'. This chunk must modify a verb group. The modified verb group has a constraint that its TAM must belong to the list of TAMs associated with the 'ko' post-position. If these constraints are satisfied, then the modifier-modified relation is k1. This rule is independent of other rules and the multiplicity of the dependency relation is at most 1. It carries a weight equal to 4. Another example is given below.

VG tam=X NP vib=ko reln=k2 dep=k1 mult=1 weight=0

The above rule is read similar to the previous example of the rule. The difference here is in the modified constraint (2nd field). Unlike the rule we saw earlier, there are no constraints for TAM (denoted by an 'X') and the modifier-modified relation (k2) depends on another relation (6th field). This means that a 'k1' should have already been marked before we can apply this rule successfully.

All the rules are placed in a rule file. The automated annotator (henceforth annotator, unless otherwise specified) applies each and every rule by scanning the rule file from top to bottom in a sequential manner.

3.3 Categorization of Rules

The rules can be categorized based on the following two criteria:

1. Functionality: The rules that have been formed explore the possibility of certain dependency relation existing between two nodes in a context.
2. Effectiveness: There are some rules, which we define as robust (or strong and effective). Section 4.3 explains the experiments on these rules in detail.

3.4 Algorithm

The algorithm used by the annotator is a multiple iterative process. It takes one chunk of the input sentence at a time, and then applies rules from the rule file one by one. The rules are in the form of a tuple mentioned in Section 3.2. The algorithm works on certain assumptions and heuristics which are described in the following sections.

[9]vib: vibhakti. 'vibhakti' is a generic term for preposition, post-position and suffix.

3.4.1 Assumptions and Heuristics

Linguistic Attributes or Features: The rules made are dependent on certain linguistic attributes. Some of these attributes include:

1. Post-positions,
2. TAM,
3. Category of a lexical item,
4. List of verbs, nouns belonging to a particular class,
5. POS tags of lexical items.

Clause Boundary: Hindi is a verb final language. Based on this assumption, the annotator divides the SSF sentence into certain boundaries, termed as clause boundaries, before marking the dependency relations. The sentence has one or more clause(s) and a boundary is imposed on each clause that possesses a VGF (a verb group chunk) with a finite verb as its head. Note that, in case the final clause in a sentence does not possess any finite VGF, a boundary is imposed on it. This verb group with a finite verb becomes the boundary and recurring occurrences of such VGs are formed into different boundaries. This implies that in the Hindi, any verb (generally the root of the complete dependency tree), seeks its children or dependency relations towards its left. The whole sentence is scanned from left to right on the basis of a clause boundary. Of course, there are instances where such definition of a clause will fail. Making the clause boundary identifier more sophisticated will be taken up in the future.

Correct Input: The input representation for our annotator is a sentence in SSF. The sentence is POS tagged and chunked (Bharati et al., 2006). We assume that the sentence has been POS tagged and chunked correctly. The input has no dependency labels marked.

3.4.2 Steps of the Algorithm

The algorithm can be explained with the help of an example sentence. This explains how the annotator works using the rule file. The example sentence is as follows:

(1) [[((koI))$_{NP}$ ((rAma kI))$_{NP}$ ((cIjZa))$_{NP}$
'someone' 'ram's' 'thing'

((le lewA))$_{VGF}$]]_1 [[((waba BI))$_{NP}$
'take' 'then' 'EMPH'

((use))$_{NP}$ ((gussA))$_{NP}$ ((nahIM AwA
'him' 'anger' 'not' 'came'

WA))$_{VGF}$]]_2 .
'was'

"Ram did not get angry even if someone took his thing."

The sentence is in wx[10] notation. This sentence contains two clause boundaries, which have been labeled as _1 and _2 respectively. This sentence is fed as input to the tool. The tool iterates over all the chunks in a sequential order to mark the dependency labels. For each chunk it looks for all the rules which might possibly be applicable. Sometimes, this linearity might get violated because of the rule prioritization.

First of all, the first NP chunk 'koI' gets marked as k1 with its parent chunk being the first VGF (finite verb group) 'le lewA' i.e., the first verb within the clause boundary. This rule states that the first occurrence of a '0' case marked noun chunk is labeled as k1. The rule can be seen in Figure 3.2.

In the next iteration the second chunk gets marked as r6 with its parent being the next noun chunk i.e., 'cIjZa', as the constraints of the rule for r6 are satisfied by this chunk. The third chunk is marked as k2. The rule states that k2 would only be marked when a k1 has already been marked within the clause boundary.

Similarly, after the iterations within the first clause boundary are finished, the tool iterates over the next clause boundary and marks the first chunk as k7t with its parent being the first verb in the clause boundary which is ((nahIM AwA WA)). The head (underlined word) of this noun chunk is a time expression and hence the relation k7t. Likewise, the second noun chunk 'use' is marked as k1. This is marked as k1 because the rules states that the first occurrence of a noun chunk having a 'ko' post-position is to be marked as k1. This chunk has an implicit post position 'ko' (dative case) and thus is marked as k1.

The next noun chunk 'gussA' although being a 'pof' is marked k2. The rule for k2 for a '0' post position is perfectly applicable to this chunk. We do not mark any 'pof' relation

[10] In this notation, capitalization roughly means aspiration for consonants and longer length for vowels. In addition, `w' represents `t' as in French *entre* and `x' means something similar to `d' in French *de*, hence the name of the notation. For mapping with Devnagari see, http://ltrc.iiit.ac.in/MachineTrans/research/tb/map.pdf

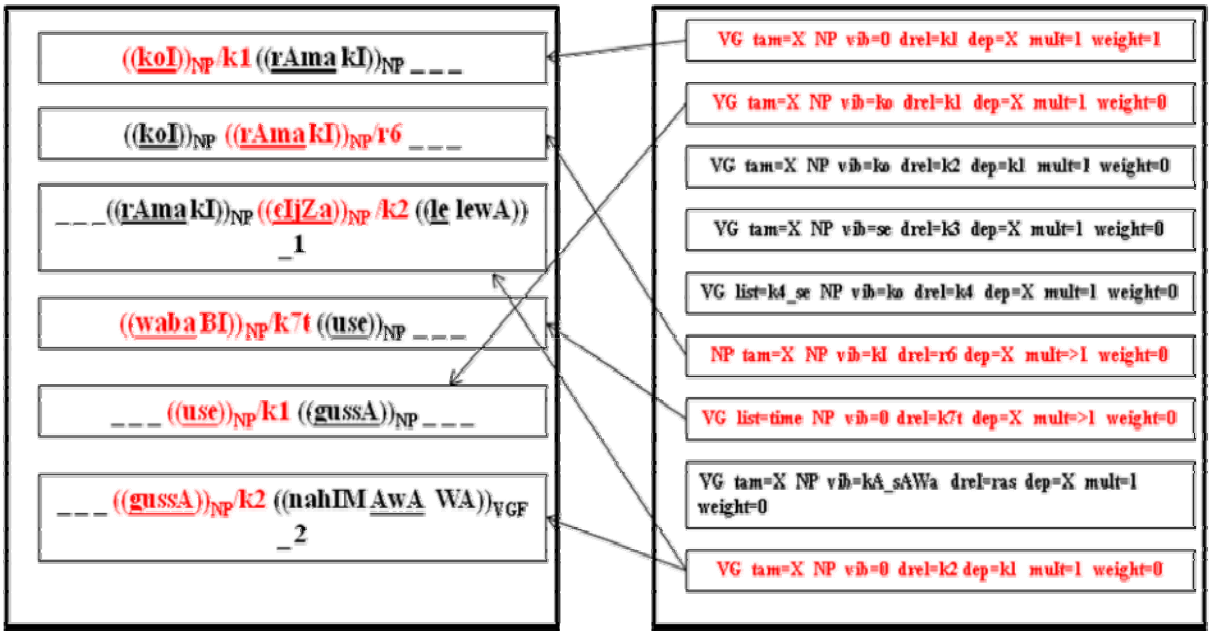

Figure 3.2.

(cf. section 3.2.1) in a sentence and so the chunk gets marked as k2. Figure 3.2 shows the complete iteration and the selection of rules clearly.

4 Experiments and Evaluation

Experiments were conducted using the annotator to

1. define the scope of automatic annotation.
2. find out the accuracy of the process of automated annotation
3. judge the extent to which this tool would help the manual annotators in annotating the treebank.

4.1 Baseline

Before establishing the baseline we need to know the fact that a post-position may be overloaded when it comes to its mapping with different karakas and non-karakas. For example, 'ko' maps to k1, k2, k4. 'se' maps to k3, k5 as well as rh. Hence, it makes the task of automatic annotation a non-trivial task. It is important, therefore, to come up with appropriate rules for marking dependency labels correctly.

A baseline was established to determine the minimum threshold value of precision and recall of various dependency relations. The baseline consists of only those rules for which a dependency relation has the highest frequency of occurrence with its corresponding post-position. The baseline was evaluated on the test data set of 312 sentences. The baseline results are given in Table 4.1.

Table 4.1 shows the labeled accuracy[11] (LA) and labeled attachment accuracy[12] (LAA) values of precision and recall for the relations marked for the baseline.

Relation	Precision		Recall	
	LA	LAA	LA	LAA
k1	54.7%	50.1%	53.1%	48.9%
k2	49.1%	43.4%	19.9%	17.6%
k3	9.3%	9.3%	50.0%	50%
k4	-	-	-	-
k5	-	-	-	-
k7	86.1%	83.0%	42.7%	41.2%
k1s	-	-	-	-
r6	82.1%	78.5%	79.0%	75.5%
rh	100%	100%	15.7%	15.7%
rd	66.6%	66.6%	25.0%	25.0%
rt	69.5%	60.8%	84.2%	73.7%
ras	66.6%	66.7%	75.0%	75.0%
adv	67.3%	63.4%	56.4%	53.2%
vmod	89.6%	75.8%	38.2%	38.2%
nmod	23.2%	21.9%	27.4%	25.8%

Table 4.1.

[11] Labeled Accuracy (LA) values for a relation only consider the edge label without considering its attachment with head.

[12] Labeled Attachment Accuracy (LAA) values for a relation consider both attachment and edge label.

It can be seen that k7, r6 and vmod have a higher threshold value for the baseline. This is due to the fact that these relations have a very strong mapping with their corresponding post-positions. This set of rules formed is pretty robust. On the other hand, relations like k3 and nmod do not have such a correspondence with the post-positions. Thus, their baseline values are on the lower side. Recall for these relations is also low.

As compared to the results for LA, the results for LAA are lower. This is understandable as the attachment of a relation with its parent is not always correct.

Note that we do not have any baseline rules for relations like k4, k5 and k1s. This is because baseline rules for other relations have the same post-position as that of these relations. For example, the relations k3 and k5 have the same post position 'se' for which they record their highest respective occurrences. But, k3 was chosen for the baseline rule as the number of occurrences for k3 with the post position 'se' exceeds than that of k5.

4.2 Evaluation of Test Data

After establishing the baseline, experiments were conducted to test the rules described in section 3.2. These rules were tested on the test data set.

Table 4.2 shows the LA and LAA for the evaluation experiment.

Relation	Precision		Recall	
	LA	LAA	LA	LAA
k1	66.0%	57.7%	65.1%	57.6%
k2	31.3%	28.3%	27.8%	25.1%
k3	13.0%	13.0%	64.2%	64.2%
k4	40.6%	40.6%	54.1%	54.1%
k5	44.4%	44.4%	21.0%	21.0%
k7(p/t)	80.8%	77.2%	61.0%	58.4%
k1s	51.0%	51.0%	18.5%	18.5%
r6	82.1%	78.7%	89.6%	85.8%
rh	100%	100%	15.7%	15.7%
rd	66.7%	66.6%	25.0%	25.0%
rt	69.5%	60.8%	84.2%	73.7%
ras	66.6%	22.2%	75.0%	25.0%
adv	67.3%	63.4%	56.4%	53.2%
vmod	89.6%	75.8%	45.2%	38.2%
nmod	23.2%	21.9%	27.4%	25.8%

Table 4.2.

.

Note that**, t**he results for k7 have been calculated considering all k7's, k7p/t's together as k7[13]. This is true for every result we present henceforth. The results are higher than that of the baseline which is shown in Table 4.1. Relations like k1 showed significant improvement in precision and recall. Recall value of r6 also went up. However, k2 recorded a lower precision than the baseline. This may be attributed to the fact that the baseline rule for k2 was formed for the post position 'ko'. But while evaluating the accuracy of the tool k1 and k2 both had rules for the post position 'ko'.

The tool marked a total of 1654 dependency instances. Out of these, 1016 were correctly marked. This amounts to an overall LA precision of 61.4%. The overall recall stands at 55.3%. Among the relations that record a higher value of precision are k7(p/t), r6, vmod, rh and rd. In fact, relations like r6 have an even higher recall value. This goes to show that rules for relations like r6 have broad coverage in addition to them being robust. It can be seen that the relations like r6, rh, rd, rt and vmod have higher baseline values and thus remain unchanged when the rules for these relations are tested on the test data.

The table shows significant improvement for relations like k1 as compared to the baseline. The overall precision of the tool when run on the test data set for the results for LAA is 56.4%, which is lower than the overall precision of correctly marked dependency labels. The results for LAA also take into account correct attachment with their respective parent chunks along with the label. As can be seen, the recall is also expectedly lower at 51% than that of LA.

Note here the sharp fall (almost 15%) in the precision value of vmod considering head attachment as opposed to vmod without head attachment. This happens because, while attaching the relation with its parent verb, the finiteness and non-finiteness is not considered. This, in some cases results in erroneous attachment with its parent verb.

4.3 Ease of Annotation

We took the rules which we describe as 'robust' and calculated the precision and recall values of these rules. These rules have been termed 'robust' because for these rules high accuracy. The results of these rules are shown in Table 4.3.

[13] k7 thus becomes a generic label for location. It entails location in time, place and location elsewhere.

Relation	Precision		Recall	
	LA	LAA	LA	LAA
k1	91.5%	67.6%	13.8%	10.3%
k7(p/t)	80.8%	77.2%	61.1%	58.4%
r6	82.1%	78.7%	89.6%	85.8%
rh	100%	100%	15.7%	15.7%
rd	66.6%	66.6%	25.0%	25.0%
rt	69.5%	60.8%	84.2%	73.7%
adv	67.3%	63.4%	56.4%	53.2%
vmod	89.6%	75.8%	45.2%	38.2%

Table 4.3.

As can be seen the precision for k1, without checking head attachment, goes up to 91.5%. However, there is also a steep fall in its recall value. This shows that although the rule is very strong, it has less coverage.

Results for LAA have also been shown in Table 4.3. It is shown that LAA precision for k1 as compared to that of LA has gone up but recall has come down. However, there is a steep fall in the precision of k1 (about 24%) with head attachment as compared to that of k1.

It can be seen that some of the rules formulated are robust which would in turn aid manual annotators to annotate a large sized corpus reducing time and effort. This is one of the objectives for building the system which has been highlighted in Section 1. With a higher precision value, this tool would facilitate the process of annotation.

5 General Discussion

It was noted there were quite a few conflicts that the tool had to confront while marking the dependency labels. These conflicts resulted in errors. These errors have been analyzed in the following section.

5.1 Error Analysis

Error analysis was done on the development data set as it makes the tool unbiased. The size of the development data has been mentioned in section 3.1. The results for the development data are shown in Table 5.1.

Relation	Precision		Recall	
	LA	LAA	LA	LAA
k1	66.7%	58.5%	68.3%	59.6%
k2	37.4%	35.1%	32.2%	30.2%
k3	20.9%	19.6%	66.7%	62.5%
k4	44.7%	43.2%	57.3%	55.3%
k5	43.1%	37.9%	27.1%	23.9%
k7(p/t)	84.1%	77.3%	52.6%	60.0%
k1s	39.8%	38.2%	16.0%	15.4%
r6	82.1%	78.5%	82.7%	79.0%
rh	93.7%	75.0%	21.1%	16.9%
rd	90.9%	81.8%	25.0%	45.0%
rt	92.0%	88.6%	69.8%	67.2%
ras	71.7%	26.4%	67.8%	25.0%
adv	49.3%	42.7%	46.2%	40.0%
vmod	95.0%	83.2%	43.1%	37.7%
nmod	18.4%	17.2%	18.5%	17.4%

Table 5.1.

Overall LA precision for running the tool on the development data set is 63.3%, with a total of 7859 relations marked, out of which, 4975 dependency instances are correctly marked. The recall stands at 56.2%. We describe the results for LAA later in section 5.1.6.

The fact that the tool when run on the development data set records a higher value for precision and recall than that for the test data set can be interpreted easily. Since the rules were formed after analyzing the development data set, the results are bound to be slightly higher than that of the test data. Moreover, the number of sentences in the development data set being much greater than the test data set, also results in a better coverage for the rules.

Below, we show the error analysis for incorrect labels marked for relations k1, k2, k3, k4, k5. There were also cases where the tool failed to mark any dependency relation for a chunk. This happened because the chunk failed to satisfy all the required constraints of the rules.

5.1.1 Error Analysis for k1

Table 5.2 below shows the major conflicts the annotator confronted with while marking k1.

.Relation	k2	k4	k7(p/t)	adv	pof
No. of errors	258	4	173	18	173

Table 5.2.

As can be seen from the Table 5.2, k1 records the highest number of errors with k2 followed by the pof relation. k1 conflicts with k2 for ‘0’ and ‘ko’ post positions primarily. This happened because it is difficult to distinguish between the semantics of a ‘k1’ chunk with that of a ‘k2’ chunk. Currently, the tool does not use semantic cues to disambiguate between the two relations. It was observed that a large number of instances of k1 have a ‘0’ post position i.e. no post posi-

tion. The disambiguation can be done by bringing into account semantic features like animacy. Recent machine learning experiments in parsing Hindi have shown that such semantic features prove to be very helpful in disambiguating certain relations (Bharati et al., 2008). We also intend to incorporate this feature in our tool.

The reason for conflicts with k7(p/t) is similar to that of k2. Also, not all time and place expressions which are generally k7t and k7p respectively are identified by the annotation tool. We do not mark the pof relation hence, this results in a high number of errors while marking k1.

5.1.2 Error Analysis for k2

As can be shown in Table 5.3, the following relations have major conflicts when k2 is wrongly marked by the tool.

Relation	k1	k4	k1s	pof	k7(p/t)
No. of errors	285	42	170	190	34

Table 5.3.

As mentioned in section 5.1.1, k1 and k2 have a high degree of conflicts, which a rule based system using syntactic cues alone cannot handle. We need to take into account certain other features like animacy (Section 5.1.1). Similarly, the relation k1s is also difficult to disambiguate from k2.

Another important observation here is that many noun or adjectival chunks which are 'part of' a complex verb represented as 'pof' are also not marked by the tool. These 'pof' chunks exhibit very similar surface properties with the k2 chunks. This results in a number of conflicts between the two. However, their ambiguity may be resolved to a certain degree by considering their semantic properties as well as the relative distance of these chunks from the main verb.

5.1.3 Error Analysis for k3

Table 5.4 shows the major conflicts the annotator confronted with while marking k3.

Relation	k5	rh	k2	adv
No. of errors	54	15	50	24

Table 5.4.

The tool records the highest number of conflicts with k5. This is due to the fact that k3 and k5 have almost similar syntactic cues. For example, most of the chunks which are either k3 or k5 have the post position 'se'. The reason for conflict with 'k2' is also the same as above. We however, include a verb class for those verbs which have a sense of separation from the source for marking k5.

5.1.4 Error Analysis for k4

Most of the errors while marking k4, are with k2 for the 'ko' post-position as shown in Table 5.5.

Relation	k2
No. of errors	42

Table 5.5.

The relation k4 is marked for the chunk with post position 'ko' only when its parent verb in question belongs to a predetermined list of recipient or beneficiary verbs. But there are cases where the parent verb may not always semantically be a recipient or a beneficiary verb, although it may be present in such a list.

5.1.5 Error Analysis for k5

Relation	k3	k2	vmod
No. of errors	11	5	9

Table 5.6.

k5 being syntactically very similar to k3, records the highest number conflicts with it.

k5 is marked only for that chunk whose parent verb is a motion verb with a sense of separation from its source. Examples of such verbs are 'jA', 'A', 'gira' etc. But in some cases, even though the parent verb is a motion verb it may not represent the sense of separation in a particular context. As a result, k5 gets erroneously marked. The same reason holds true for vmod and k2 as well.

5.1.6 Error Analysis for Incorrect Head Attachment

The error analysis, as described above, was done for incorrect and unmarked dependency labels. We now describe the error analysis for the head attachment of these dependency labels. In Table 5.1 we have also showed the LAA values for the dependency labels for the development data.

Again, the results for checking accuracy of dependency labels marked along with correct

attachment with their corresponding parent are lower than the corresponding values for LA as shown in Table 5.1. The overall LAA precision stands at 57.6%. 4531 relations are correctly marked including their attachment with parent. Recall comes around 51.2%.

We have not taken into account the finiteness and non-finiteness of a parent chunk which happens to be a verb group while attaching the child with its parent. This could be the reason for lower accuracy for the head attachment. For relations like r6 and nmod whose parent chunk is generally not a verb group, the heuristic that its parent would be towards the immediate right of the chunk does not always hold true. This could also be the reason for lower accuracy.

6 Comparison with Parser

We compared our annotator with a broad coverage constraint based parser for Hindi (Bharati et al., 2002). For comparison, only the rules which were robust were considered for the annotator. The results for robust rules have been shown in Table 4.3. Our results were compared with the first parse given by the parser for a sentence. We consider only the first parse for comparison with the annotator. The results of the comparison are shown in Tables 6.1 and 6.2.

Table 6.1 shows the LA values of the output of the parser compared with the LA values after running the annotator as a post-processing tool for the parser. We have shown only the precision values of the relations as the annotator intends to fine tune and improve the results of the first parse of the parser rather than provide higher coverage or recall. Table 6.1 shows improved results for all the relations after running the annotator over the first parse output.

Relation	Precision of Parser	Precision after Post Processing
k1	39.8%	54.8%
k7(p/t)	48.4%	71.5%
r6	81.6%	82.2%
rh	75.0%	85.7%
rd	- [14]	66.6%
rt	-	69.5%
adv	0	68.6%
vmod	4.2%	7.3%

Table 6.1.

[14] A hyphen ('-') against a relation indicates that the parser gave no output i.e., the first parse did not show the relation under consideration.

It is evident from the table that for relations like adv, the parser failed to mark even one instance correctly for the first parse. But, the annotator corrects this anomaly and records a high value of precision for the same of 68.6%. For relations like rh and rd, the parser gave no output in the first parse. This is also corrected by the annotator. However, the annotator records only a marginal increase in the precision value for vmod. This is due to that fact that the parser marks a lot of vmod relations. Thus, the overall precision for vmod, after running the annotator, is low even though the annotator has a high precision for vmod (see Table 4.3).

Table 6.2 shows the LAA results after running the annotator as a post processing tool for the first parse of the broad coverage parser. Again, the recall values have not been shown.

Relation	Precision of Parser	Precision after Post Processing
k1	37.1%	46.6%
k7(p/t)	42.1%	67.3%
r6	77.5%	78.5%
rh	25.0%	57.1%
rd	-	66.6%
rt	-	60.8%
adv	0	64.7%
vmod	3.9%	6.2%

Table 6.2.

As can be seen from Table 6.2, it is evident that the annotator helps in improving the output of the first parse of the parser. It also improves the head attachment accuracy for the dependency labels shown in the table. Hence, this initial experiment shows that the annotator would prove helpful in fine tuning and improving the results of the parser.

7 Conclusion and Future Work

In this paper we showed how a rule based system has been built for automatic annotation for a Hindi Tree Bank. We also explained the rule format and the algorithm used by the annotation tool. The performance of such a system was also discussed. We showed how the system would facilitate manual annotation. We also showed results of the experiments conducted and in the end issues related to these experiments were discussed as part of error analysis.

Based on the issues discussed in section 5, we intend to exploit certain linguistic cues as described in section 5.1 to reduce the errors, while

marking dependency relations in future. Also, we intend to take into consideration the finiteness and non-finiteness of the parent verb to reduce errors while attaching the child head with its parent.

Acknowledgements

We express our sincere thanks to Rafiya Begum, Itisree Jena, Sapna Sharma, Dilip Singh and Vishwanath Naidu (research students at LTRC, IIIT-Hyderabad), who have played an important role in the proof-checking of many linguistic resources including the treebank and have helped us carry out various experiments for the system. The treebank (Hyderabad dependency treebank, version 0.05) used was prepared at LTRC, IIIT-Hyderabad. The subsequent work reported in this paper was partially supported by the NSF grant (Award Number: CNS 0751202; CFDA Number: 47.070).

.

References

L. van der Beek, G. Bouma, R. Malouf, and G. van Noord. 2002. The Alpino dependency treebank. *Computational Linguistics in the Netherlands.*

Rafiya Begum, Samar Husain, Arun Dhwaj, Dipti Misra Sharma, Lakshmi Bai and Rajeev Sangal. 2008. *In Proceedings of The Third International Joint Conference on Natural Language Processing (IJCNLP), Hyderabad, India. 2008. http://www.iiit.net/techreports/2007_78.pdf*

Eckhard Bick. 2007. Automatic Semantic Role Annotation for Portuguese. In *Proceedings of TIL 2007 - 5th Workshop on Information and Human Language Technology*, Rio de Janeiro, Brazil.

Akshar Bharati, Samar Husain, Bharat Ambati, Sambhav Jain, Dipti M Sharma and Rajeev 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), CDAC Pune, India. 2008.*

Akshar Bharati, Rajeev Sangal, Dipti Misra Sharma and Lakshmi Bai. 2006. AnnCorra: Annotating Corpora Guidelines for POS and Chunk Annotation for Indian Languages. *Technical Report, Language Technologies Research Centre IIIT, Hyderabad.*

Akshar Bharati, Rajeev Sangal and Dipti M Sharma. 2005. ShaktiAnalyser: SSF Representation. *http://shiva.iiit.ac.in/SPSAL2007/ssf-analysis-representation.pdf*

Akshar Bharati, Rajeev Sangal, T Papi Reddy. 2002. A Constraint Based Parser Using Integer Programming, In *Proceedings of ICON-2002: International Conference on Natural Language Processing,2002*.
http://www.iiit.net/techreports/2002_3.pdf

Akshar Bharti, Vineet Chaitanya and Rajeev Sangal. 1995. *Natural Language Processing: A Paninian Perspective*, Prentice-Hall of India, New Delhi, pp. 65-106. http://ltrc.iiit.ac.in/downloads/nlpbook/nlp-panini.pdf

Akshar Bharati and Rajeev Sangal. 1993. Parsing Free Word Order Languages in the Paninian Framework, *ACL93: Proc. of Annual Meeting of Association for Computational Linguistics*.

Cristina Bosco and V. Lombardo. 2004. Dependency and relational structure in treebank annotation. In *Proceedings of Workshop on Recent Advances in Dependency Grammar at COLING'04*.

Thorsten Brants and Wojciech Skut. 1998. Automation of Treebank Annotation. In *Proceedings of New Methods in Language Processing (NeMLaP-98)*

.

E. Hajicova. 1998. Prague Dependency Treebank: From Analytic to Tectogrammatical Annotation. In *Proceedings of the First Workshop on Text, Speech, Dialogue (TSD'98).*

R. Hudson. 1984. *Word Grammar*, Basil Blackwell, 108 Cowley Rd, Oxford, OX4 1JF, England.

P. Kiparsky and J. F. Staal. 1969. 'Syntactic and Relations in Panini', *Foundations of Language* 5, 84-117.

M. Marcus, B. Santorini, and M.A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank, *Computational Linguistics 1993*.

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