LTAG-spinal Treebank and Parser for Hindi

by

Prashanth Reddy, Aswarth Abhilash, Akshar Bharati

in

ICON-2009: International Conference on Natural Language Processing
(INCON2009)

Report No: IIIT/TR/2009/216

Centre for Language Technologies Research Centre
International Institute of Information Technology
Hyderabad - 500 032, INDIA
December 2009
LTAG-spinal Treebank and Parser for Hindi

Prashanth Mannem, Aswarth Abhilash, Akshar Bharati
Language Technologies Research Center
International Institute of Information Technology
Hyderabad, AP 500032, India
prashanth@research.iiit.ac.in, abhilash_d@students.iiit.ac.in

Abstract
Statistical parsers need huge annotated treebanks to learn from and building treebanks is an expensive proposition. To create parsers for different grammar formalisms in a language, building separate treebanks for each of those isn’t a feasible task. Treebanks available in one formalism can be converted into an other either automatically or with minimal human effort by exploiting the similarities and differences between the two. In this work, we present an approach to extract an LTAG-spinal treebank from Hyderabad Dependency Treebank for Hindi. LTAG-spinal is a variant of Lexicalized Tree Adjoining Grammar (LTAG) with desirable linguistic, computational and statistical properties. A bidirectional LTAG dependency parser is trained on the extracted treebank and an LTAG dependency accuracy of 80.86% is reported.

1 Introduction
Natural language processing in Indian languages has gained traction in recent years due to the availability of annotated corpora for Part-Of-Speech tagging and chunking. Work has recently started on creating dependency treebank for Hindi. The Hyderabad Dependency Treebank (HyDT) for Hindi uses the annotation scheme proposed in (Begum et al., 2008). It is based on the Paninian framework with the dependency labels being syntactico-semantic in nature. Dependency based representations have become popular in the last few years for building treebanks (and thereby for parsing too) due to their simpler representation when compared to other formalisms like phrase structure grammar (Hajič et al., 2004; Nivre et al., 2007).

However, there are parsers and treebanks in other grammar formalisms like Phrase Structure Grammar (PSG), Lexicalized Tree Adjoining Grammar (LTAG), Combinatory Categorial Grammar (CCG), Lexical Functional Grammar (LFG) for various languages. A wide range of NLP systems have benefited from using parsers in these grammar formalisms depending on the application. A statistical parser needs annotated treebank to learn from and building treebanks in all grammar formalisms is not a good idea due to the time and cost involved in such efforts. An easier way is to convert treebanks in one formalism to an other with either no or minimal manual annotation. Treebanks for various grammar formalisms have been extracted from the Phrase Structure based Penn Treebank (Marcus et al., 1994) for English (Chen et al., 2006; Cahill et al., 2002) and parsers have successfully been trained on them.

Lexicalized Tree Adjoining Grammar (LTAG) (Joshi and Schabes, 1997) is a grammar formalism which has attractive properties from the point of view of Natural Language Processing (NLP). LTAG has appropriate generative capacity and a strong linguistic foundation. Processing over structures in the LTAG representation leads to deeper language processing. In this work, we propose an approach to extract LTAG-spinal treebank from dependency structures in HyDT. LTAG-spinal, a variant of LTAG was proposed for statistical processing in the LTAG framework (Shen et al., 2008). LTAG-spinal treebank for English was extracted from the Penn Treebank (Marcus et al., 1994) with Propbank annotation (Palmer et al.,
We extract LTAG-spinal treebank for Hindi from dependency treebank annotated with the scheme proposed in (Begum et al., 2008). LTAG-spinal defined in Shen et al. (2008), has substitution and non-predicate adjunction merged as *attachment*, so as to encode the ambiguity of argument-adjunct distinction, while *adjunction* is reserved for constructions which may result in *wrapping structures*. Therefore, LTAG-spinal allows non-projective dependencies to reveal deep relations, which makes it different from other simplifications of LTAG, like TIG. *Conjunction* is a special operation used to represent predicate coordination explicitly. While Shen et al. (2008) used both phrase structure trees and predicate argument/adjunct information from Propbank to extract the treebank for English, we have only the dependency treebank to work with. So, for our treebank, the spinal operations are defined in such a way that the LTAG-spinal trees can be extracted from the information available in the dependency structures in HyDT.

The LTAG-spinal treebank provides a very desirable resource for statistical LTAG parsing and dependency parsing. An LTAG dependency parser was trained on the extracted treebank and results are reported.

In the remainder of the paper, we introduce the Hyderabad Dependency Treebank in Section 2 and the LTAG-spinal formalism in Section 3. In Section 4, the extraction process in presented in detail along with some discussion in the end. The Bidirection LTAG Dependency Parser (BLDP) used to train on the extracted Hindi LTAG-spinal treebank is described in Section 5 and the results are reported in Section 6. Section 7 gives the summary of our work and scope for future work in this direction.

2 Hyderabad Dependency Treebank (HyDT)

HyDT is a dependency annotated treebank for Hindi. The annotation scheme used for HyDT is based on the Paninian framework (Begum et al., 2008). The dependency relations in the treebank are syntactico-semantic in nature where the main verb is the central binding element of the sentence. The arguments including the adjuncts are annotated taking the meaning of the verb into consideration. The participants in an action are labeled with *karaka* relations (Bharati et al., 1995). Syntactic cues like case-endings and markers such as post-positions and verbal inflections, help in identifying appropriate *karakas*.

The dependency tagset in the annotation scheme has 28 relations in it. These include six basic karaka relations (adikarana [location], apaadaan [source], sampadraan [recipient], karana [instrument], karma [theme] and karta [agent]). The rest of the labels are non-karaka labels like vmod, adv, nmod, rbmod, jjmod etc...

In the annotation scheme used for HyDT, relations are marked between chunks instead of words. A chunk (with boundaries marked) in HyDT, by definition, represents 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 dependency arc. The relations among the words in a chunk are not marked. Thus, in a dependency tree in HyDT, each node is a chunk and the edge represents the relations between the connected nodes labeled with the karaka or other relations. All the modifier-modified relations between the heads of the chunks (inter-chunk relations) are marked in this manner. The annotation is done using Sanchay² mark up tool in Shakti Standard Format (SSF) (Bharati et al., 2005). For the work in this paper, to get the complete dependency tree, we used an automatic rule based intra-chunk relation identifier. The rules mark these intra-chunk relations with an accuracy of 99.5%, when evaluated on a test set.

The treebank has 1781 sentences with a total of 31857 words.

3 LTAG-spinal formalism

LTAG-spinal was introduced in (Shen et al., 2008). It was mainly designed for statistical processing in the LTAG framework. In LTAG-spinal, there are two kinds of elementary trees, initial trees and auxiliary trees, as shown in Figure 1. What makes LTAG-spinal different from

---

1The entire dependency tagset can be found at http://ltrc.deptagset.googlepages.com/k1.htm
2http://sourceforge.net/projects/nlp-sanchay
LTAG is that elementary trees are in the spinal form. A spinal initial tree is composed of a lexical spine from the root to the anchor, and nothing else (the substitution nodes that are present in LTAG are missing). A spinal auxiliary tree is composed of a lexical spine and a recursive spine from the root to the foot node. The common part of a lexical spine and a recursive spine is called the shared spine of an auxiliary tree. For example, in Figure 1, the lexical spine for the auxiliary tree is \( B_1, \ldots, B_i, \ldots, B_n \), the recursive spine is \( B_1, \ldots, B_i, \ldots, B_1^* \) and the shared spine is \( B_1, \ldots, B_i \).

![Figure 1: Types of spinal elementary trees](image)

There are three operations to combine trees in LTAG-spinal. They are attach, adjoin and conjoin. Attach operation makes the root of an initial tree as a child of a node of another spinal tree. Attachment is used to represent both substitution and sister adjunction in traditional LTAG. Adjoin is used to combine an auxiliary tree to another spinal tree. Adjunction in LTAG-spinal is the same as adjunction in the traditional LTAG. It can effectively do wrapping leading to non-projective dependencies. In the English LTAG-spinal treebank, raising verbs and passive ECM verbs are represented as auxiliary trees to be adjoined. In addition, adjoin is used to handle many cases of discontinuous arguments in Propbank. Conjoin is a special operation used to encode the ambiguity of argument sharing in coordination structures. The conjuncts within a coordination structure are related using the conjunction operation to denote the shared structure.

LTAG-spinal differs from LTAG in the way it underspecifies predicate argument-adjunction distinction and subcategorization frames. In (Shen et al., 2008), an English LTAG-spinal treebank was extracted from the Penn Treebank (PTB) reconciled with Propbank annotation. Graph transformations were applied to the phrase structures in the PTB for the three spinal operations based on the predicate argument structure in Propbank. The PTB which is a phrase structure treebank was used in extracting the spines of the elementary trees for words in a sentence. LTAG-spinal elementary trees were then extracted from the transformed PTB subtrees recursively, with respect to Propbank annotations for predicates and a head table for all other constituents.

To extract spines for these words, its phrase structure tree is needed. Since HyDT is a dependency treebank, spines can’t be extracted from it. So, we use a set of rules to identify spinal elementary trees for the words in the transformed tree. These rules take into account information like the word’s POS tag, its children, its spinal operation in the transformed tree and clause information.

4 LTAG-spinal treebank for Hindi

While English had Penn Treebank and Propbank to extract the LTAG-spinal treebank, we only have the dependency treebank (HyDT) for Hindi. To create the Hindi LTAG-spinal treebank, LTAG-spinal trees have to extracted from the labeled dependency structures in HyDT.

Before we describe the extraction process in detail, the roles of each of the three spinal operations in Hindi and the kind of constructions that are handled using these operations are defined in the next section.

4.1 Operations in Hindi

The dependency labels in HyDT don’t make any distinction between predicate arguments and adjunctions. It is not a trivial task to accurately extract this information from the dependency treebank. So, attachment operation is used to represent both arguments and adjunctions.

In the current work, LTAG-spinal adjunction (adjoin operation) is only used to handle non-projective structures in HyDT. A relation is non-projective if the yield/projection of the parent in a relation is not contiguous. The head in a non-projective relation in a dependency tree is connected to its parent with the adjoin operation.

Coordination structures in HyDT are represented with the connective (CC POS tag) as the head and the conjuncts as its children. A cc-of dependency label is given to the dependency between the connective and the coordination conjunct. The conjoin operation is used to address such coordinate constructions. In HyDT, the relations from both coordinate and subordinate conjuncts to the connectives are labeled with cc-of in
the treebank. Coordinate conjuncts can be distinguished from subordinate ones by looking at the number of children of the connective and their chunks labels. If a connective has more than one child and their chunk labels are same, then it is a case of coordination conjunction. In (Shen et al., 2008), conjoin operation is used for predicate coordination only and not NP coordination. In this work on Hindi, since ccof relation is used to represent both NP and predicate coordination, we use the conjoin operation for NP coordination too.

The rest of the relations in the dependency tree, which are not coordination constructions and non-projective relations, are handled using the attach operation.

4.2 LTAG-spinal trees from HyDT

Extracting LTAG-spinal trees from HyDT involves two steps. The first is to identify the substructures in the dependency tree which correspond to the three spinal operations defined in the previous section. The dependency tree is transformed to reflect the LTAG-spinal kind of structure for the three operations. The second step involves assigning the spinal elementary trees to each of the nodes in the transformed tree. Apart from assigning the spines, the point of attachment of each child node (along with its spine) on the spine of the parent node is also determined.

Section 4.2.1 lists out the extraction process for the three LTAG-spinal operations from the dependency tree in HyDT. Section 4.2.2 details the process of assigning spines and their point of attachment on the parent spine for each node in the derivation tree.

4.2.1 Extraction from HyDT

The constructions handled by the spinal actions described in Section 4.1 are identified in the dependency tree of HyDT and the tree is transformed to reflect LTAG-spinal tree. This is done by cutting the dependency tree into segments and reconnecting them with LTAG-spinal operations i.e. attach, adjoin and conjoin. A dependency tree along with the spinal operations identified are listed in Figures 2 and 3.

Conjoin: The connective (with CC POS tag) is the head of the conjuncts in HyDT. Whereas, in LTAG-spinal treebank, the conjunct nearest to the connective’s parent is made the head of the coordination structure and the other conjuncts are conjoined to the previous conjunct. The connective is then attached to its immediate conjunct. For example, in Figure 2, the dependency coordination structure involving parisa, aur and praaga se is transformed with praaga se (conjunct nearest to the parent) attached to its parent VG, parisa conjoined to the conjunct praaga se and CC aur attached to praaga se. The transformed tree is in Figure 3.

Adjoin: Non-projective arcs in the dependency tree are identified and the parent in each of these arcs is connected to its own parent using the adjoin operation. In Figure 2, since there is a non-projective arc from vaha sikke to the extraposed relative clause headed by laaye the, vaha sikke is adjoined to its parent kho gaye.

The tree after identifying the three operations and performing the transformations is in Figure 3.

4.2.2 Assigning spines to words

Apart from spinal operations, spinal elementary trees are required for each of the words in the sentence to get the LTAG-spinal tree. The transformed dependency tree from the previous section is similar to the LTAG-spinal derivation tree except that the nodes in the former tree are words and not elementary trees. This transformed tree is an intermediate tree between the dependency tree and the final LTAG-spinal tree.

To extract spines for these words, its phrase structure tree is needed. Since HyDT is a dependency treebank, spines can’t be extracted from it.
that coins lose PAST. which I-ERG Paris and Prague from bring PAST.

I lost the coins I brought from Paris and Prague
So, we use a set of rules to identify spinal elementary trees for the words in the transformed tree. These rules take into account information like the word’s POS tag, its children, its spinal operation in the transformed tree and clause information.

Two issues need to be addressed when assigning spinal elementary trees for words in a dependency tree. The first is the about which phrase categories to include in the spine and the length of the spine. The second one is about where in the spine should each of its child spinal trees be combined. These two vary depending on the grammar one wants to build and are not fixed. Our rules are crafted to produce minimal spines which retain the properties of the dependency treebank in LTAG-spinal tree. In HyDT, as mentioned in Section 2, dependency relations are established between chunks in a sentence. Each chunk is labeled with a chunk label and intra-chunk relations are left unspecified. We use a rule-based intra-chunk dependency identifier to establish these unspecified intra-chunk relations in the dependency tree. The result is a complete dependency tree with relations between chunk heads and their modifiers within a chunk also established. The spines generated by the rules retain the chunk level information in LTAG-spinal trees.

We define 5 different types of spinal elementary trees for the Hindi LTAG-spinal treebank. They are listed in Figure 5. The rules for assigning these spines to the words are (the which question discussed above):

1) The first spine contains just the POS tag of the word. This spine is assigned to non-head leaf nodes of a chunk in the tree. Since these don’t have any children and are neither head words of a chunk, they have the shortest spine consisting of only the POS tag. Words vaha, gaye, se and the in Figure 4 are non-head words of their respective chunks and have just the POS tag as part of their spines.

2) The second spine contains the label of the chunk (CHK) the words belong to and the POS tag. This spine is assigned to words which are either head words of chunks with no modifiers other than the ones within a chunk or non-heads of chunks with modifiers present within the same chunk. In Figure 4, jo, maine and aur are chunk head words with no modifiers. Hence, their spines contain only the chunk label and the POS tag.

3) The third spine has, apart from the chunk label and the POS tag, a XP node at the top. This spinal elementary tree is defined for words which have a modifier outside its own chunk. In the example, kho, parisa, praaga and laaye get these spines.

4) The fourth spine is for words which are adjointed on the left in the transformed dependency tree. In Figure 4, sikke which is adjointed to its parent kho gets this auxiliary spine.

5) The fifth spine is for words which are adjointed on the right in the transformed dependency tree.

Coming to the where question, which deals with where to combine the modifier elementary tree on the parent’s spine. We have two simple rules. 1) If the modifier is an intra-chunk modifier then it is connected at the CHK node. 2) Else, (i.e. if the modifier is outside its own chunk), the modifier is combined at the XP node. In Figure 4, vaha, gaye, se and the are connected to their chunk heads at the chunk label in the spine. The rest of the words are connected at the XP node in the spine.

4.3 Discussion

The extraction approach described above, identifies segments in the dependency tree corresponding to the three spinal operations and transforms the dependency tree to reflect the spinal derivation tree. Spinal elementary trees are then assigned to nodes in this transformed dependency tree. Another way to extract spinal trees is to first convert dependency structures in HyDT into phrase structure trees and then extract spinal trees. We decided upon the former approach because of the simplicity and flexibility dependency structures provided in identifying non-projectivity for adjunction and also in tree transformation needed for spinal conjoin operation.

The spines in the elementary trees are minimal due to the lack of phrase structure trees. They can be expanded further to distinguish clauses, predicate adjunction and predicate coordination by including richer (and longer) spines in the rules. But, the problem arises when the point of attachment (where to combine) has to be decided when combining an elementary tree with another. This is another reason we restrict ourselves to minimal spines.

In this work, the adjunction operation is used
exclusively to denote non-projective relations. The operation can be used for phenomena which may result in non-projectivity (and need wrapping adjunction) similar to (Shen et al., 2008). Man-ne-m et al. (2009) classified the non-projectivity occurring in HyDT into various classes and provided cues to identify them. Their work can be used to further enrich adjunction operation in the extracted treebank.

New efforts are being put into developing multi-representational and multi-layered treebank for Hindi and Urdu (Bhatt et al., 2009). This treebank will have both dependency as well as phrase structure representations. It will be multi-layered with dependency and Propbank structure annotated for each sentence. Phrase structure treebank is produced automatically from these using conversion rules. This Hindi/Urdu treebanking work concentrates on two representations only. Since the multi-representational treebank discussed in Bhatt et al. (2009) is under construction, we used the smaller dependency treebank annotated with the scheme proposed in (Begum et al., 2008) for this work. Once the new treebank with multiple layers and multiple representations is available, we hope to extract much richer and deeper LTAG-spinal treebank from it.

5 Bidirectional LTAG Dependency Parser

Shen and Joshi (2008) proposed a bidirectional dependency parsing algorithm which does greedy search over the sentence and picks the relation between two words with the best score each time and builds the parse tree instead of doing a left-to-right or right-to-left parsing. The search can start at any position and can expand the partial results in any direction. The order of search is learned automatically. The parser uses the three spinal operations attach, adjoin and conjoin to establish a relation between two entities.

In the rest of the section, we give an overview of the parsing algorithm along with the training and inference processes presented in (Shen and Joshi, 2008).

5.1 Parsing Algorithm

We are given a linear graph \( G = (V, E) \) with vertices \( V = \{ v_i \} \) and \( E(v_{i-1}, v_i) \) with a hidden structure \( U = \{ u_k \} \). The hidden structure \( u_k = (v_{s_k}, v_{e_k}, b_k) \), where vertex \( v_{e_k} \) depends on vertex \( v_{s_k} \) with label \( b_k \) is what we want to find (the parse tree). A sentence is a linear graph with an edge between the adjacent words. A fragment is a connected sub-graph of \( G(V, E) \). Each fragment \( x \) is associated with a set of hypothesized hidden structures, or fragment hypotheses for short: \( Y^x = \{ y^x_1, \ldots, y^x_j \} \). Each \( y^x_j \) is a possible fragment hypothesis of \( x \). A fragment hypothesis represents a possible parse analysis for a fragment. Initially, each word with its POS tag comprises a fragment.

Let \( x_i \) and \( x_j \) be two two fragments, where \( x_i \cap x_j = \emptyset \) and are directly connected via an edge in \( E \). Let \( y^{x_i} \) be a fragment hypothesis of \( x_i \), and \( y^{x_j} \) a fragment hypothesis of \( x_j \). We can combine the hypotheses for two nearby fragments with the operations attach, adjoin and conjoin. Suppose we choose to combine \( y^{x_i} \) and \( y^{x_j} \) with an operation \( R_{type, dir} \) to build a fragment hypothesis for \( x_k = x_i \cup x_j \). The output of the operation is

\[
y^{x_k} = R_{type, main}(x_i, x_j, y^{x_i}, y^{x_j}) \supseteq y^{x_i} \cup y^{x_j}
\]

where \( type \in \{ attach, adjoin, conjoin \} \) is the type of operation and \( main \in \{ left, right \} \), representing whether the left or the right fragment is the parent. \( y^{x_i} \) and \( y^{x_j} \) stand for the fragment hypotheses of the left and right fragments \( x_i \) and \( x_j \).

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

\[
s(R) = W.\phi(R)
\]

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

\[
\text{score}(y(R)) = s(R) + \text{score}(R.y^{x_i}) + \text{score}(R.y^{x_j})
\]

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

$$ \text{score}(c) = \sum_{\text{fragment hypothesis } y^z \text{ of } c} \text{score}(y^z) $$

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

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

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

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

**Algorithm 1 BLDP: Inference Algorithm**

**INPUT:** graph $G=(V, E)$ and weight vector $W$; 
**INITIATE** cut $T$, hypotheses $H^T$, queue of candidate operations $Q$; 
**while** $Q$ is not empty **do** 
 operation $y \leftarrow \arg\max_{op \in Q} \text{max score}(op, W)$; 
 **if** compatible$(H^c, y)$ **then** 
 promote($y^*$); 
 **else** 
 demote($y$); 
 **end if** 
 **UPDATE** $T$, $H^T$, $Q$ with $y$; 
**end while**

**Algorithm 2 BLDP: Training Algorithm**

**W** $\leftarrow 0$; 
**for** round $= 1..T$; $i = 1..n$ **do** 
**LOAD** graph $G=(V, E)$, hidden structure $H^r$; 
**INITIATE** cut $T$, hypotheses $H^T$, queue $Q$; 
**while** $Q$ is not empty **do** 
 operation $y \leftarrow \arg\max_{op \in Q} \text{max score}(op, W)$; 
 **if** compatible$(H^c, y)$ **then** 
 **UPDATE** $T$, $H^T$, $Q$ with $y$; 
 **else** 
 $y^* \leftarrow \text{searchCompatible}(Q, y)$; 
 promote($y^*$); 
 demote($y$); 
 **UPDATE** $Q$ with $W$; 
**end if** 
**end while**

6 Results

The Bidirectional LTAG Dependency Parser (BLDP) discussed in the previous section was trained and tested on the extracted Hindi LTAG-spinal treebank. Since there are no standard training and testing division, we did a ten fold cross validation on the entire data (1781 sentences with 31857 words) to get the accuracies. So, the test corpus is a different set of 178 sentences for each of the ten evaluations.

Features are defined on POS tags and lexical items of the nodes in the relation as well as of the nodes in the context. Two kinds of context are used a) context from the input sentence (sentence context) and b) local context from the hypotheses of the nearby fragments. For a detailed description of features, see (Shen and Joshi, 2008).

BLDP was trained on the Hindi LTAG-spinal treebank with the same feature set. Sentence context (SC) and local context (LC) were varied and accuracies are reported in Table 1. The LTAG dependency accuracy is the percentage of words with correct operations and parents. BLDP doesn’t produce spines and is only restricted to producing LTAG dependencies. Therefore, only LTAG dependency accuracies are reported. The highest accuracy reached is 80.86% for sentence context of 2 and local context of 2.

<table>
<thead>
<tr>
<th>Sentence</th>
<th>SC(0)</th>
<th>SC(1)</th>
<th>SC(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LC(0)</td>
<td>80.30%</td>
<td>80.75%</td>
<td>80.24%</td>
</tr>
<tr>
<td>LC(1)</td>
<td>80.44%</td>
<td>80.49%</td>
<td>80.00%</td>
</tr>
<tr>
<td>LC(2)</td>
<td>80.49%</td>
<td>80.49%</td>
<td>80.86%</td>
</tr>
</tbody>
</table>

Table 1: LTAG dependency accuracies for various context.

Bharati et al. (2008) reported an unlabeled attachment score of 88.67% trained on 1200 sentences from HyDT and tested on 350 sentences. Their work only considers relations between chunks and not the complete dependency tree with intra-chunk relations also. The input to their system is the gold standard chunks with
POS tags. The input to our system is a sentence with POS tags. The output is a complete LTAG dependency tree with dependencies for all words in the sentence identified. So, the accuracies of the two systems can not be compared. Moreover, the LTAG dependency trees are richer with LTAG-spinal operations and are harder to learn than dependency structures.

7 Conclusion and Future Work

In this work, we proposed an approach to extract LTAG-spinal treebank from Hyderabad Dependency Treebank. The LTAG-spinal operations attach, adjoin and conjoin are defined for Hindi LTAG-spinal treebank and are identified in HyDT. Tree transformations on dependency structures are done to convert it into a LTAG-spinal derivation tree. Finally, minimal spines are assigned to nodes in the derivation tree to get the complete LTAG-spinal tree. The bidirectional LTAG dependency parser was trained using the extracted treebank and a highest LTAG dependency accuracy of 80.86% was reached when a ten fold cross validation was done.

The spines extracted in the treebank are minimal and in the future we plan to define richer spines taking cue from the phrase structure conversion rules from (Xia et al., 2009). Once richer spines are defined, an LTAG-spinal parser which predicts and uses spine information can be trained for use in various applications. Different methods can be tried out in assigning spines from the perspective of their respective contributions towards the parsing accuracy.

This work can also be the starting point towards generic LTAG-spinal extraction algorithms from dependency treebanks of various languages.

References


Akshar Bharati, Samar Husain, Bharat Ambati, Sambhav Jain, Dipti 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), Pune, India.


