Cost Effective Dependency Parsing for Indian Languages

Thesis submitted in partial fulfillment
of the requirements for the degree of

MS by Research
in
Computational Linguistics

by

Aniruddha Tammewar
201025163
uttam.tammewar@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
July, 2015

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Cost Effective Dependency Parsing for Indian Languages" by Aniruddha Tammewar, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Adviser: Prof. Dipti Mishra Sharma

To My **Family**

Acknowledgments

First and most of all, I would like to thank my adviser Prof. Dipti Misra Sharma for her expertise, guidance and patience throughout the five years of my undergrad and masters at IIIT Hyderabad. Her faith and confidence in me, encouraged me to get involved in the NLP research. I would like to thank her and other faculty members including Dr. Radhika Mamidi, Dr. Soma Paul, Late Prof. Lakshmi Bai, Dr. Prashanth Mannem, Dr. Manish Shrivastava, Prof Vasudeva Verma for conducting courses in various domains of NLP, which helped in building up the essential concepts required for research in NLP. A special mention to Dr. Srinivas Banglore and Prof. Michael Carl for giving me an opportunity to work on a real World application as a research intern.

I am highly grateful to my mentors, friends Sambhav Jain and Riyaz Bhat, for their valuable feedbacks on my ideas and giving them due direction with their experience. I would like to thank both of them for getting me involved in their research ideas, which worked as an initial step towards building and working on my own ideas. Naman Jain and Karan Singla deserve a special mention for always supporting and helping me in my research work.

I am grateful to the vibrant and stimulating lab environment of LTRC, which instigated several active discussions to come up with new ideas and scrutinizing the existing ones. Some contributors deserving special mention: Bhasha, Rishabh, Himanshu, Kunal, Arpita, Urmi, Rahul, Ankush, Vandan, Vighnesh, Nikhilesh, Ganesh, Pruthwik, Silpa. I also thank the administration, management staff for making the administrative matters run smoothly.

A special acknowledgment to my friends Akshay, Shariq, Sasha, Karan, Praveen, Abhineet, Kartik and all other *duals* for not letting the fifth year be monotonous, rather making it eventful by constant breaks from academic life. Finally I would like to thank my family for their constant support.

Abstract

Indian languages are MoR-FWO[1] and hence differ from English in structure and morphology. There are many distinguished characteristics possessed by Indian languages. While working with these languages we have to keep in mind, these characteristics and plan strategies accordingly. We worked on improving Dependency Parsing for Indian Languages, more specifically for Hindi, an Indo-Aryan Language.

In the conventional Dependency Parsing methods, the focus has been on developing robust data driven dependency parsing techniques. This initiated efforts in creating hand annotated large treebanks, consisting of hand annotated features. These treebanks serve as input for the training of data-driven parsers. The annotations in Indian Languages' treebanks are generally multi-layered and furnish information on part of speech category of word forms, their morphological features, related word groups and the syntactic relations. For improvements, richer and richer features are being added. This process of manual annotation is expensive, as it requires a lot of human efforts. It is a tedious task to create treebanks for all the languages. Even if we make the treebanks available, in the real time scenario we require many tools to extract features automatically. Building such tools is also a complex task. We are in an era with almost unlimited access to raw data. Nevertheless, we often struggle to make sense of most of it. Much of this data is unlabeled and thus useless in many of the traditional supervised machine learning scenarios, that require explicit labeled/hand-annotated examples. In this work, we present our efforts towards exploring cost effective approaches for building and improving parsers for resource-poor languages. For this purpose we try to use unsupervised techniques to extract features from the largely available mono-lingual raw corpus.

Using cross-lingual treebank transfer, we exploit the available treebanks for other languages and using some techniques like MT[2] and try to generate a treebank for the target language. We can use this treebank for training of parser. We first try this approach for Hindi. An important constraint for using this approach is that the annotation of treebank needs to be similar cross-linguistically. For this, we use UD[3] framework. Universal Dependencies is an initiative to create cross-linguistically consistent treebank annotation for many languages, with the goal of

[1] morphologically rich and free word order

[2] Machine Translation

[3] Universal Dependencies

facilitating multilingual parser development, cross-lingual learning, and parsing research from a language typology perspective. In the previous studies, we have seen that CPG[4] framework is better suited for Indian Languages. So we try to compare other techniques with the cross-lingual parsing in UD framework.

In the concluding work, we try to make use of Vector Space Modeling on a large monolingual raw data, a recent technique being used widely across different tasks. Use of large monolingual corpus helps reducing the problem of data sparsity. We try to explore this technique to achieve three goals. Using word-embeddings extracted from vector space modeling as features, we first try to improve the state-of-the art accuracy for Hindi. Here we use word-embeddings as additional features other than the conventional features. The second goal is to help building parsers for less resourced languages. This is done by replacing the costly linguistic features with word-embeddings. This requires minimal human annotation. The third goal we achieve, is improving parser's performance in general domain data. We show results where parser is trained on News domain and input sentences are from four different domains Box-Office, Cricket, Gadget and Recipe.

[4]Computational Paninian Grammar

Contents

List of Figures

List of Tables

Chapter *1*

Introduction

Natural language processing (NLP) aims to deliver computational models which constitute intelligent systems aiming to match human capabilities of language processing. In a common scenario, humans deal with two major tasks involved in language processing namely, language understanding and language generation. Similarly NLP also deals with two processes, Natural Language Understanding/Analysis (enabling computers to derive meaning from human or natural language input) and Natural Language Generation (generating human language from meaning). Natural language come into sight in two major forms, spoken utterances i.e. *speech* and written form i.e. *text*. From computational point of view there is a lot of difference in processing of these two forms of language. In our work we focus on analysis of natural language text input. Processing of a written text demands analysis at different levels like word, phrase, sentences, etc.Figure 1.1 reflects NLP anatomy and depicts a higher level view of *language analysis hierarchy*

Moving down the hierarchy, the NLP research community has successfully delivered analyzers for major languages like English, Hindi etc., at word level and phrase level, which meet industry standards. Moreover active research is also in progress to further improve the quality of Part Of Speech (POS) taggers, morph analyzers etc., and port them to other languages. The current period is a transition phase for syntactic analyzers or *Syntactic Parsers*, which are moving from research environment and finding utility in practical real world applications. But again the efforts have focused on improving widely used languages. Research on minority languages is still yet to reach this point. The research on semantics is also gaining pace and expected to meet industrial standards in some time. Some of the applications have already started using these tools, though. Pragmatics and discourse are comparatively harder to analyze and therefore are still open NLP challenges. Although the research in these areas is very active, they are far behind the industrial benchmarks.

Syntactic parsing, which often make use of word and phrase level analysis, provides vital information of grammatical relations between words of a sentence. It is useful in several NLP applications such as statistical machine translation [97], question answering [94], natural

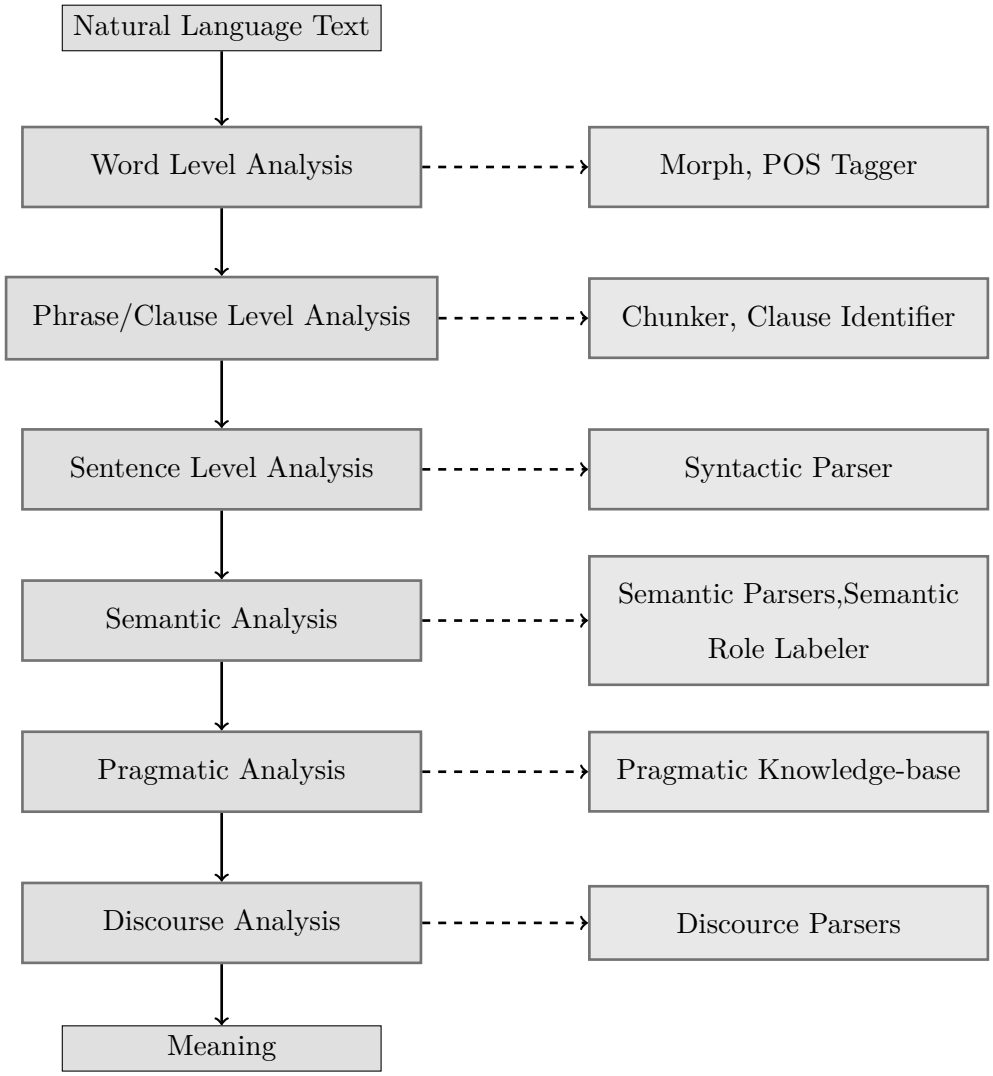


Figure 1.1: Overview of Natural Language Analysis of Text

Credits: The diagram is taken from Sambhav Jain's masters thesis

language interface [30], natural language generation [95], text summarization evaluation [80], sentiment analysis [63], etc. Owing to a pivotal role in varied NLP applications, much focus has been on developing quality parsers, furnishing eminent syntactic analysis for a sentence, in the last decade or two.

The goal of syntactic parsing research is to ensure quality and reliability of syntactic analysis, delivered to various NLP applications. The research in this area till now, has produced good quality parsers for majority languages, 'Hindi' in case of Indian Indian languages. But other Indian languages and other minority languages across the world have not got enough attention. Now that research in mainstream languages has produced good results, it's time to apply those

techniques to other languages. But before applying the same techniques to the new languages, we must look back and see if the process was easy. *Not at all!*. The traditional techniques involve a lot of manual work and are very expensive in terms of the investment of time, human power and money.

The main problem with the statistical approaches is the tagged data. Before shifting our focus to other languages, we must try and reduce the overall cost of the whole process involved. In our work in this thesis, we try to exploit cheap resources and make the process cost effective. We try to make use of already existing resources that we created for mainstream languages to create resources for other languages, using cross-lingual treebank transfer techniques. In another approach we exploit raw mono-lingual corpus and learn features from it which in turn can replace the conventional features which are costly to obtain using tools (for many languages, these tools are not available). We do this using a recent technique of vector space modeling.

We also try to improve the quality of existing parsers by adding rich information from resources like Hindi WordNet and produce a new state-of-the-art parser. Despite of extensive advancements in parsing research, it is observed that parsers perform clumsily when incorporated in NLP applications. One of the reason behind this is the domain confinement of parsers. The statistical parsers are trained on data compiled from finite domains, but NLP applications are often domain unbound and freely accept data from any general domain. To address this issue we again exploit the use of Vector Space Modeling on large mono-lingual data, which covers sentences from a large variety of domains and hence helps parser work better with unknown domains.

1.1 Contributions

There are two main contributions of this thesis.

1. **State-of-the-art dependency parser for Hindi**
   We first explore different settings in MaltParser and come up with best settings and strategy suitable for Hindi, to beat the previous State-of-The-Art parser and set up a new baseline for further experiments. [5] This was done as a part of a shared task on Hindi Dependency Parsing in MTPIL workshop, COLING 2014. We achieved highest scores for gold data. We then do error analysis and try to figure out different strategies to tackle them. To handle some type of errors, we try to explore if semantic information extracted automatically from concept ontology present in HWN[1] can help improve the parser quality. Using this approach we get significant improvement over our baseline, and we get a new state-of-the-art dependency parser for Hindi. Then in the end we try to make sense out of large raw mono-lingual corpus that is very easily available for any

[1]Hindi WordNet

language, to improve the quality further. We use Vector Space Modeling and use Word-Embeddings as features to conclude that this information captures a lot of information which is generally not captured by the typical features we use.

2. **Parsing strategies when less resources are available, to make it cost effective**
   In this work we first try to develop treebank, the most important resource for data driven parsing, for Hindi using cross lingual parsing within Universal Dependency framework. Once we get treebank from other source languages we train a parser on it and evaluate the parsing quality. A series of experiments are done with different strategies.
   In the second task we try to make the parsing cost effective by removing the features extracted automatically using costly-to-build tools which are required for parsing and keep only the simpler features like word-embeddings learned using vector space modeling, POS tags and other trivial features. This strategy shows a significant improvement in real world scenario of parsing new sentences.
   In the third task, we show that when the sentences to be parsed are from a different domain than that of the training data, we need not create different treebanks for each domain and we can improve the results significantly by using only the clustering of word-embeddings. This makes it cost effective as a lot of manual work is saved.

1.2 Outline

1. **Chapter 2:** This chapter introduces the concept of syntactic parsing and the two frameworks Dependency Parsing and Constituency Parsing. Different approaches for dependency parsing are discussed. Then we discuss in more detail, the two paradigms involved in data driven dependency parsing, followed by a survey of popular tools. Next we discuss in detail about the scenario of dependency parsing in Indian Languages, the problems involved in Indian Languages. We then describe the most important resource 'Treebank For Hindi'.

2. **Chapter 3:** This chapter, first tries to beat the benchmarks in Hindi Parsing, with a series of experiments exploiting different settings in Malt Parser and different strategies for parsing. We setup a new state-of-the-art system which would be the baseline for further experiments. Next, we try to show how adding richer semantic information from Hindi WordNet helps improve parse quality.

3. **Chapter 4:** This chapter focuses on exploiting different techniques to reduce the overall cost involved in the process of Dependency Parsing. First a series of experiments are done using cross-lingual treebank transfer technique to show how it can help produce treebanks for resource-poor languages. This technique uses the already built treebanks to produce new treebank and hence reduces the manual work. In the next part we try to remove

the expensive tools involved in gathering all the features needed for parsing a new unseen sentence. Instead, we try to use simple features which are easy to obtain . It makes use of large raw mono-lingual corpus to learn a vector space model, and uses word embeddings as features. We also try to show that if these features used as complementary features rather than replacements, it helps improve the parse quality further which is useful for resource rich languages. The next part tackles the problem of domain confinement of the parser by clustering the word vectors.

4. **Chapter 5:** We conclude the thesis in this chapter providing summary of the thesis. We also put forward some insights about possible future work in the direction of making the parsing much easier and optimized. In future work we will try to change the pipeline architecture to a parallel architecture, which handles the problem of error propagation and reduces the overall time complexity.

Chapter *2*

Dependency Parsing

In abstract terms, *Syntactic Parsing* refers to the study of structure of language (how words connect each other). Specifically, the goal is to relate surface form i.e. the sentence itself, to semantics (the encoded meaning). The representational device is the *tree structure*. An input sentence to a parser encodes language specific properties (in terms of word-order, word-forms, lexical items, etc.), whereas the output abstracts away from these properties and decode in order to yield a structured formal representation that explicitly reflects the functions of aforementioned different elements in the sentence.

2.1 Dependency Formalism and Phrase Structure Formalism

(contents of this section are based on Dr. Prashanth Mannem's slides on Dependency parsing[1])Syntactic parsing establishes the representational relations pertaining to a grammatical framework like phrase structure grammar, dependency grammar, categorical grammar, etc. It has been suggested that morphologically rich and free word languages can be handled more efficiently using the dependency based framework than the constituency based [45] [83] [12]. The use of dependency formalism for various NLP/CL tasks, especially for parsing, has increased many folds since the last decade. Consequently, most of the parsers for MoR-FWO languages are dependency based [93]. In dependency grammar, syntactic structure consists of lexical items, linked by binary asymmetric relations called *dependencies*. We are interested in grammatical relations between individual words (*governing* and *dependent* words). It does not propose a recursive structure, rather gives a network of relations. These relations also have labels.

Dependency structures explicitly represent:

- Head-dependent relations (directed arcs)
- Functional categories (arc labels)

[1] http://slidegur.com/doc/1797000/state-of-art-in-dependency-parsing

- Possibly some structural categories (parts-of-speech)

Phrase structure explicitly represent:

- Phrases (non-terminal nodes)
- Structural categories (non-terminal labels)
- Possibly some functional categories (grammatical functions)

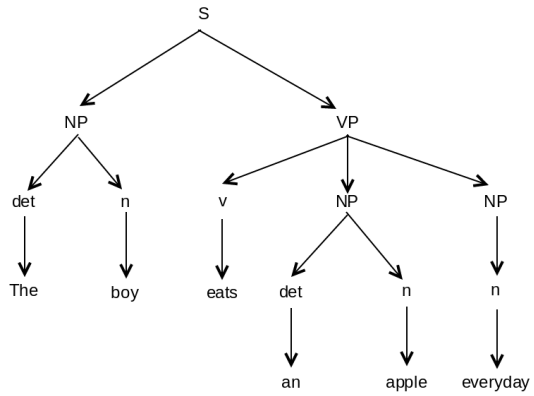


Figure 2.1: Example of constituency based representation

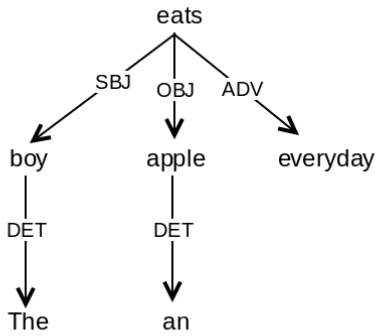


Figure 2.2: Example of dependency based representation

Figure 2.1 and Figure 2.2 respectively show a simplified phrase structure and a dependency structure for the sentence '*The boy eats an apple everyday.*'

2.2 Approaches for Dependency Parsing

In general, dependency parsing can be broadly divided into *grammar-driven* (Parsing done using grammars) and *data-driven* (Parsing by training on annotated/un-annotated data) dependency parsing. These approaches are not mutually exclusive, though. Before moving to these approaches, we first give a formal definition of *dependency tree*.

We look at the given sentence as a sequence of words w1,w2...wn. Dependency graph is defined as *D=(W,E)*. Where W is the set of nodes i.e. word tokens in the input sequence, E is the set of unlabeled tree edges (wi, wj) (wi, wj $\in$ W). (wi, wj) indicates an edge from wi (parent) to wj (child). Task of mapping an input string to a dependency graph satisfying certain *conditions* is called *dependency parsing*.

A dependency graph is well-formed iff:

- *Single head*: Each word has only one head.
- *Acyclic*: The graph should be acyclic.
- *Connected*: The graph should be a single tree with all the words in the sentence.
- *Projective*: If word A depends on word B, then all words between A and B are also subordinate to B (i.e. dominated by B).

2.2.1 Grammar Driven

The grammar-driven constraint based dependency parsing is based on the notion of eliminative parsing, where sentences are analyzed by successively eliminating representations that violate constraints until only valid representations remain. Parsers based on this idea use underspecified dependency structures, represented as syntactic tags and disambiguated by a set of constraints intended to exclude ill-formed analyses [50][61][91]. In the eliminative approach, parsing is viewed as a constraint satisfaction problem, where any analysis satisfying all the constraints of the grammar is a valid analysis. For a fully defined constraint satisfaction problem, the variables, their domains and the set of constraints that define the permissible values for the values need to be satisfied. The constraints mentioned above also need to be satisfied for the tree to be well-formed.

Constraint satisfaction in general is an NP-complete decision problem, which means that to ensure reasonable efficiency in practice one has to use controlled heuristics. Early version of this approach used local consistency[61], which attained polynomial worst case complexity by only considering local information in the application of constraints. In the more recently developed XDG framework [34], the problem is negotiated using constraint programming to solve the satisfaction problem defined by the grammar for a given input string.

In realistic world natural languages are not much restricted to certain rules. While parsing such text using constraint satisfaction technique, two problems may arise. First, for a given input string, there may exist no analysis that can satisfy all the constraints, which lead to a robustness problem. Secondly, there may exist more than one analysis, that can lead top the problem of disambiguation.

2.2.2 Data Driven

A data-driven approach to parsing, primarily makes use of machine learning from an annotated data set in order to parse a new sentence. More precisely such methods are called supervised data-driven methods. There are two main problems, (a)*learning problem*, which is the task of learning a parsing model from a representative sample of structure of sentences (training data), and (b) the *parsing problem* (or inference/decoding problem), which is the task of applying the above learned model for the analysis of a new sentence. The data-driven methods differ in the type of algorithms used for learning and parsing a new sentence. Data-driven dependency was first established by Eisner (1996)[40] using graph based methods, the transition based approach was first explored by Kudo and Matsumoto (2002) [58] and Yamada ana Matsumoto (2003)[98].

Data driven parsing ensures that an output in any unrestricted natural language text, makes it a more robust approach as compared to its grammar driven counterpart. Data driven parsers generally give a single best output, thereby they do not create a disambiguation problem at user's end.

2.3 Two Traditions in Data Driven Dependency Parsing

The two major classes of approaches for data driven dependency parsing fall either into *transition based* or *graph based* methods.

2.3.1 Graph Based

Graph based parsers such as MSTParser[62] build a complete graph over words of a sentence, each word being a node. It gives weights to the edges of the graph based on prior learned contexts. It then proceeds to choose the maximum spanning tree as the output parse.

MSTParser represents a generic directed graph $G = (V, E)$ by its vertex set $V = \{v1, ..., vn\}$ and set $E \subseteq [1 : n] \times [1 : n]$ of pairs (i, j) of directed edges $vi \rightarrow vj$. Each such edge has a score $s(i, j)$. Since G is directed, $s(i, j)$ does not necessarily equal $s(j, i)$. A maximum spanning tree (MST) of G is a tree $y \subseteq E$ that maximizes the value $\sum_{(i,j)\in y} s(i, j)$ such that every vertex in V appears in y. The maximum projective spanning tree of G is constructed similarly except that it can only contain projective edges relative to some total order on the vertices of G.

For each sentence x we define the directed graph $G_x = (V_x, E_x)$ given by

$$V_x = \{x_0(= ROOT), x_1, ..., x_n\}$$

$$E_x = \{(i, j) : i \neq j, (i, j) \in [0 : n] \times [1 : n]\}$$

That is, G_x is a graph with the sentence words and the dummy $ROOT$ symbol as vertices and a directed edge between every pair of distinct words and from the $ROOT$ symbol to every word. It is clear that dependency trees for x and spanning trees for G_x coincide, since both kinds of trees are required to be rooted at the dummy $ROOT$ and reach all the words in the sentence. Hence, finding a (projective) dependency tree with highest score is equivalent to finding a maximum (projective) spanning tree in G_x.

Now, to find highest scoring non-projective tree one simply needs to search the entire space of spanning trees with no restrictions. MSTParser uses the Chu-Liu-Edmonds algorithm [23][39]. The algorithm has each vertex in the graph greedily select the incoming edge with highest weight. If a tree results, it must be the maximum spanning tree. If not, there must be a cycle. The procedure identifies a cycle and contracts it into a single vertex and recalculates edge weights going into and out of the cycle. It can be shown that a maximum spanning tree on the contracted graph is equivalent to a maximum spanning tree in the original graph [43]. Hence the algorithm can recursively call itself on the new graph. Naively, this algorithm runs in $O(n^3)$ time since each recursive call takes $O(n^2)$ to find the highest incoming edge for each word and to contract the graph. There are at most $O(n)$ recursive calls since we cannot contract the graph more then n times. To find the highest scoring non-projective tree for a sentence, x, the graph G_x is constructed and ran through the Chu-Liu-Edmonds algorithm. The resulting spanning tree is the best non-projective dependency tree. Using Chu-Liu-Edmonds algorithm, non-projective dependency trees can be generated. Many languages that allow non-projectivity are still primarily projective. So, for projective dependency parsing, MSTParser uses Eisner algorithm [40]. This algorithm has a runtime of $O(n^3)$ and has been employed successfully in both generative and discriminative parsing models[40][62]. Furthermore, it is trivial to show that the Eisner algorithm solves the maximum projective spanning tree problem. The Eisner algorithm differs significantly from the Chu-Liu-Edmonds algorithm. First of all, it is a bottom-up dynamic programming algorithm as opposed to a greedy recursive one. A bottom-up algorithm is necessary for the projective case since it must maintain the nested structural constraint, which is unnecessary for the non-projective case.

The weight vectors in this setup can be learned using statistical model. MSTParser uses Margin Infused Relaxed Algorithm (MIRA) [29], an online large-margin algorithm, to compute the weight vector. Its power lies in the ability to define rich set of features over parsing decisions, as well as surface level features relative to these decisions.

2.3.2 Transition Based

A typical transition based parser like MaltParser [77] tries to derive a single syntactic representation (dependency graph) through a deterministic sequence of elementary parsing action, sometimes combined with backtracking or repair. The process is motivated by psycholinguistic modeling, its efficiency and simplicity. Dependency parsing can be realized as deterministic search through the transition system, guided by the classifier. With this technique, parsing can be performed in linear time complexity for projective dependency trees and quadratic time complexity for arbitrary (possibly non-projective) trees[76].

MaltParser comes with a number of built-in transition systems. We describe the arc-eager projective system first described in [75](Nivre, 2003). Other systems are minor variations of these two. For a more detailed analysis of this and other transition systems for dependency parsing, see Nivre (2008)[76].

The arc-eager algorithm builds a labeled dependency graph in one left-to-right pass over the input. A configuration in the arc-eager projective system contains a stack holding partially processed tokens, an input buffer containing the remaining tokens, and a set of arcs representing the partially built dependency tree. There are four possible transitions (where *top* is the token on top of the stack and *next* is the next token in the input buffer):

- LEFT-ARC (r): Add an arc labeled r from *next* to *top*; pop the stack.
- RIGHT-ARC (r): Add an arc labeled r from *top* to *next*; push *next* onto the stack.
- REDUCE: Pop the stack.
- SHIFT: Push next onto the stack.

Although this system can only derive projective dependency trees, the fact that the trees are labeled allows non-projective dependencies to be captured using the pseudo-projective parsing technique proposed in Nivre and Nilsson (2005)[78]. MaltParser also provides an option for a non-projective transition system based on the method described by Covington (2001)[28]. This system uses a similar type of configuration of arc-eager described above, but adds a second temporary stack. Unlike the arc-eager, this allows the derivation of arbitrary non-projective dependency trees.

Classifiers: Classifiers can be induced from treebank data using a wide variety of different machine learning methods. MaltParser employs Support Vector Machine [27] for classification and provides an option between LIBSVM [21] and LIBLINEAR [41] to build the classifier.

Features are very crucial for any classifier. The features used in Malt are all symbolic and extracted from the following fields of the CoNLL data representation (Buchholz and Marsi, 2006): FORM, LEMMA , CPOSTAG , POSTAG , FEATS , and DEPREL. Symbolic features are converted to numerical features using the standard technique of binarization. Once we have the list of all possible features, then getting the best feature set is the next important step.

General procedure for feature optimization in Malt is, base model is defined and using forward and backward feature selection algorithms, language-specific feature selection is done.

2.4 Dependency Parsing For Indian Languages

Indian languages have richer morphology as compared to English. They exert a relatively free word order with SOV being the default configuration. Due to the flexible word order, dependency representation is preferred over constituency for its syntactic analysis [15]. The dependency representation do not constrain the order of words in a sentence and thus are better suited for free word order languages. The dependency grammar formalism largely used for Indian languages and thus for Hindi is *Computational Paninian Framework(CPG)*[8][16]. The dependency relations in CPG formalism are close to semantics (meaning of the sentence) and hence they are also denoted as *syntactico-semantic* relations.

2.4.1 Challenges in Parsing Indian Languages

Parsing morphologically rich languages (MRLs) like Arabic, Czech, Turkish, etc., is a challenging task [74]. A large inventory of word-forms, higher degrees of argument scrambling, discontinuous constituents, long distance dependencies and case syncretism are some of the major challenges which any statistical parser has to meet for efficient parsing of MRLs. As discussed earlier, ILs being MRL and have relatively free word order, dependency parsing is better suited for ILs. Like any other MRL, ILs are rich in morphology and allow higher degrees of argument scrambling. Begum et al. (2008) [8] have proposed a dependency based annotation scheme for the syntactic analysis of ILs.

- **Non-Projectivity:** Non-projective structures in contrast to projective dependency structures contain a node with a discontinuous yield. The phenomenon of non-projectivity poses problems for both grammar formalisms as well as syntactic parsing using transition based algorithms [59]. For instance there are about 1.1% non projective arcs in the Hindi Treebank.

- **Argument Scrambling:** In Indian languages like Hindi, because case markers carry the information about the relation between words, these words can freely change their positions in the sentence. Argument scrambling often leads to discontinuities in syntactic constituents and many a times lead to non-projective structures and long distance dependencies thus posing a challenge to parsing.

- **Case Syncretism:** In Hindi, case markers and case roles do not have one to one mapping. Each case marker is distributed over a number of case roles. Among the six case markers, only Ergative case marker is unambiguous[17]. Although case markers are good indicators

of the relation a nominal bears in a sentence, the phenomenon of case syncretism bars their ability in effectively identifying the role of the nominal while parsing. Consider the examples from (1a-1e), the instrumental से is extremely ambiguous. It can mark the instrumental adjuncts as in (1a), source expression as in (1b), material as in (1c), comitatives as in (1d) and causes as in (1e).

1. (a) (मोहन ने) (चाबी से) (ताला) (खोला)।
   (Mohan-*Erg*) (key-*Inst*) (lock-*Nom*) (opened).
   'Mohan opened the lock with a key.'
   (b) गीता ने दिल्ली से सामान मंगवाया।
   (Geeta-*Erg*)(Delhi-*Inst*) (luggage-*Nom*) (procure.)
   'Geeta procured the luggage from Delhi.'
   (c) (शिल्पकार ने) (पत्थर से) (मूर्ति) (बनायि.)
   (sculptor-*Erg*) (stone-*Inst)* (idol-*Nom*) (made.)
   'The sculptor made an idol out of stone.'
   (d) (राम की) (श्याम से) (बात) (हुई.)
   (Ram-*Gen*) (Shyam-*Inst*)(talk-*Nom*) (happen.)
   'Ram spoke to Shyam.'
   (e) (बारिश से) (कई फसलें) (नष्ट) (हो गयीं।)
   (rain-*Inst*) (many crops-*Nom*) (destroy) (happened-*Perf*)
   'Many crops were destryed due to the rain.'
2. (a) (चिड़िया) (दाना) (चुग रही है)।
   (bird-*Nom*) (grain-*Nom*) (devour-*Prog*).
   (b) (दाना) (चिड़िया) (चुग रही है।)
   (grain-*Nom*) (bird-*Nom*) (devour-*Prog.*)
   'A bird is devouring grain.'

Not all instances of a nominal in Hindi are case marked as shown in Table 2.1. In

	Patient-Unmarked	Patient-Marked
Agent-Unmarked	1276	741
Agent-Marked	5373	966

Table 2.1: Co-occurrence of Marked and Unmarked verb arguments in Hindi Dependency Treebank

appropriate contexts, a nominal can also bear a nominative case which is morphologically null (unmarked nominals). It is possible, in fact quite frequent, to have more than one

unmarked nominals within a single clause and due to the relative free word order, the movement can result in different surface configurations.

A conventional parser has no cue for the disambiguation of instrumental case marker से in examples (1a-e) and similarly, in examples (2a-b), it is hard for the parser to know whether 'bird' or 'grain' is the agent of the action 'devour'. The traditional features like POS-tags are way too coarse to provide deep information valuable for syntactic parsing while on the other hand lexical items often suffer from lexical ambiguity or out of vocabulary problem. In the next chapter, we try to solve these problems using semantic features extracted automatically from word-net.

- **Data Sparsity:** In Indian langusges, words take various lexical forms based on the grammatical information they carry. In case if nouns, they try to reflect number and case, while verbs inflect for TAM (tense, aspect, modality) and carry agreement features (gender, number, person) of one of its arguments. Such multiple forms of words increase the vocabulary size of the language, which causes problem of data sparsity in a number of NLP applications, including Parsing. In Chapter 4, we try to handle the problem of OOV using large mono-lingual corpus.

2.4.2 Hindi Dependency Treebank

Hindi dependency treebank (HTB) and Urdu dependency treebanks are being developed under the Hindi-Urdu treebank project [18][81]. A part of Hindi Treebank (HTB ver-0.5.1) was released for the Hindi Dependency Parsing shared task, MTPIL, COLING 2012. A newer (pre-release) version of HDTB has been made available for download for researchers[2]. The treebank is multi-layered and multi-representational. It is considered to be multi-representational as linguistic information is represented in different forms, dependency and phrase structure. There is a clear distinction in how the same piece of linguistic information is shown in different forms. There is merit in both these forms of representation. On the other hand, multi-layered alludes to the different types of information stored in different forms. For instance, information is annotated at the morpho-syntactic (morphological, POS, chunk), synactico-semantic (dependency relations) and lexical semantic levels (PropBank). The Dependency annotations follow computational paninian framework (CPG). The sentences are from the news & heritage domains. A separate corpus on conversation domain is also provided.
The features/information provided in the treebank are listed below.

- ***Word-form:*** The word itself.

[2]http://ltrc.iiit.ac.in/treebank_H2014

- ***POS Information:*** POS tags are annotated for each node in the sentence following the POS and chunk annotation guidelines [11]. List of POS tags used for annotation can be found in Appendix 13.1.

- ***Morph Information:*** The morphological features have eight mandatory feature attributes for each node. These features are classified as root(lemma), category(CPOS), gender, number, person, case, post position/Vibhakti (for a noun) or tense, aspect, modality (TAM) (for a verb).

- ***Chunk Information:*** After annotation of POS tags, chunk boundaries are marked with appropriate assignment of chunk labels [11]. List of Chunk tags used for annotation can be found in Appendix 13.2.

- ***Other Features:*** In the dependency treebank, apart from POS, morph, chunk and dependency annotations, some special features for some nodes are marked. For example, for the main verb of a sentential clause, information about whether the clause is declarative, interrogative or imperative is marked (stype). Similarly, whether the sentence is in active or passive voice is also marked (voice-type).

The annotations are done systematically following a generic tree-banking pipeline for Indian languages which is a series of successive steps. Annotation begins with the tokenisation of the raw text. The tokens obtained during tokenisation are, in the next steps, annotated with morphological and POS tag information. After the word level annotations, the next step in the pipeline is the grouping of correlated, inseparable words into chunks (non-recursive phrase). The processing at the steps mentioned thus far are automated by tools (tokenizer, morph-analyzer, POS-tagger and chunker). The output of each tool is, however, manually corrected and validated by the human annotators. The final step in the pipeline is the manual dependency annotation.

The dependency annotations are followed by the annotation scheme put forth by Begum et al. (2008)[8]. The scheme is based on a grammatical formalism known as Computational Paninian Grammar (CPG). The Dependency tag-set based on the scheme consists of about 43 labels, which can be broadly grouped into two categories: karaka labels and non-karaka labels [16]. The karaka labels are based on the notion of *'karaka'*, defined as the role played by a participant in an action. The karaka labels, *k1-5,7* are centered around the core meaning of a verb. Non-karaka labels like *r6, nmod, nmod_relc, etc* are, however, used to mark relations between nouns (genitives), nouns and their modifiers (adjectival modification, relativization) etc. In addition to these labels there are some special labels like *pof* and *ccof*, they do not mark dependency relations, but are used to handle special constructions like conjunct verbs (*sawaal kiyaa (question did)*), coordinating conjunctions etc.

Chapter *3*

Towards the state-of-the-art Dependency Parsing for Hindi

In Chapter 2 we got introduced to the Dependency Parsing, different parsing algorithms, tools and resources required. We also discussed about CPG framework. In this chapter we first explore these tools and resources and come up with a state-of-the-art parser. In the second Section we try to tackle the problems involved in Parsing Indian languages, as discussed in Chapter 2, using rich semantic information.

3.1 Two-stage Approach for Hindi Dependency Parsing Using MaltParser

The work presented in this section [84] was done as a part of shared task on Hindi dependency parsing in MTPIL workshop, COLING-2012[14] by a team of four members including Karan Singla, Aniruddha Tammewar, Naman Jain and Sambhav Jain. We present our approach towards improving dependency parsing of Hindi language. Our approach includes exploration using different settings available in Malt Parser following the two-step parsing strategy i.e. splitting the data into interChunks and intraChunks to obtain the best possible LAS, UAS and LA accuracies. As already explained, Hindi is a MoR-FWO language. We have seen why parsing is a challenging task for MoR-FWO languages like Turkish, Basque, Czech, Arabic, etc. because of their non-configurability. We have already seen the difficulties involved in processing Indian languages. It has been suggested that such languages can be parsed better using dependency framework rather than constituent framework [45][83][65][12]. In ICON 2010 NLP tools contest, best results were obtained by (Kosaraju et al., 2010)[55], which uses Malt parser with SVM classifier for labeling and using local morph-syntactic, chunk and automatic semantic information as features. Ambati et al. (2010)[4] has explored two-stage approach of parsing Hindi. It divided the data into two parts namely, interChunks and intraChunks. The inter chunk part of the data contains only dependency relations between chunk heads of the sentences while the intra chunk data has the dependency relations between the tokens of

a chunk. The dependency relation labels for interChunk and intraChunk are disjoint. The approach splits the problem of parsing into two parts, since labels are largely different in the two parts and also the search space is narrowed in each part thus the problem becomes simpler. This approach helps in avoiding intraChunk relations to be marked as interChunk relations and vice-versa. Following this approach, we explored different parsing algorithm parameters and learner algorithm settings of Malt Parser.

Data: For training of parser we used the treebank described in Chapter 2. A subset of the dependency annotated Hindi Treebank (HTB ver-0.5) was released as part of the Hindi Parsing Shared Task-2012 (HPST-2012). Morphological analysis, however, is not validated for errors and inconsistencies. It was released for two evaluation tracks (gold standard and automatic). In the gold standard track, the input to the system consists of lexical items with gold standard morphological analysis, part-of-speech tags, chunks and some additional features. The more detailed explanation of features can be found in Chapter 2. In the automatic track, the input to the system contains only the lexical items and the part-of-speech tags from an automatic tagger. Some sentences have been discarded due to presence of errors in the data. Table 3.1 shows the training, development and testing data sizes for Hindi. For the testing phase of the contest, the parser was trained on the entire released data(training + development).

Type	**#sentences**	**#Token**	**avg. sentence length**
Training	12,041	2,68,093	22.27
Development	1,233	26,416	21.42
Testing	1828	39,975	21.87

Table 3.1: treebank statistics

3.1.1 Experiments and results

In our experiments we have used freely available Malt Parser (version 1.6.1)[77]. In this section we give an account of experiments performed in a series. Each experiment focuses on choosing the best option for a certain parameter/feature keeping the other parameter/feature fixed. In the subsequently following experiments the best parameter chosen from previous experiment is retained. Figure 3.1 explains the work flow of the whole process. In the next few sections we will go through all the steps.

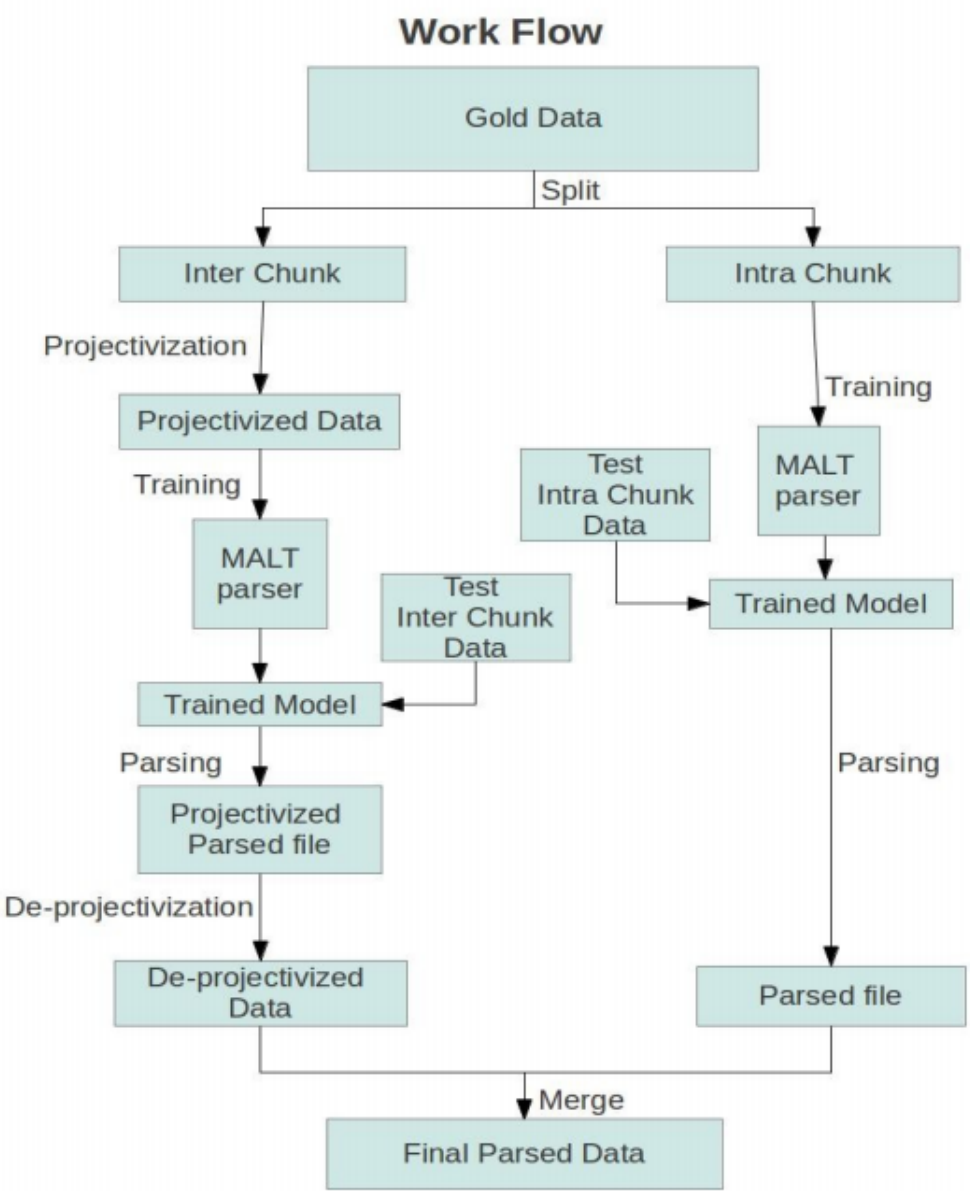


Figure 3.1: work flow of our approach for parsing

Feature Model: Feature model is the template, which governs the learning from the given training data. We explored various configuration for feature model using insight from previously used feature models from similar tasks. We observed feature model used by (Kosaraju et al., 2010)[55] performs best.

Two-stage (inter, intra chunk) approach: Every sentence in data is divided in chunks. Example:

(राम ने) (सीता को) (एक लाल किताब) (दी)।

(raam erg.) (sita dat.) (one red book) (gave).

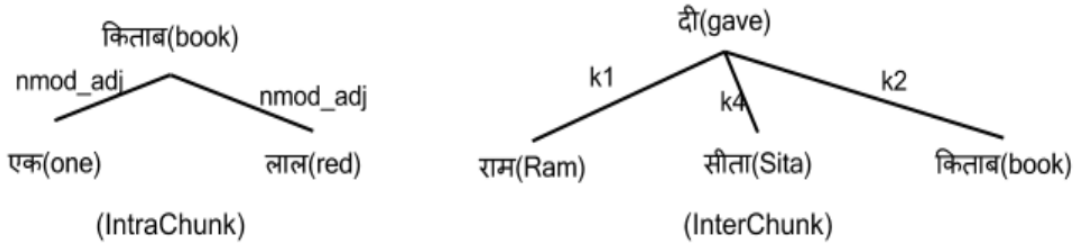


Figure 3.2: example interChunk and intraChunk

Ram gave a red book to Sita.

In the above example, the sentence is divided in 4 chunks marked by brackets. The dependency labels for intraChunk relations are different from that of interChunk, forming two disjoint sets of dependency labels interChunk labels (k1, k2, k7, etc.) and intraChunk labels (nmod_adj, lwg_psp, mod, etc.). Theoretically, the parent of a non-chunkhead token should be a chunkhead or nonchunkhead token of the same chunk and the relation should be labeled with an intra-Chunk label. (in the example, the non-chunkHead tokens एक and लाल are connected with the chunkHead token किताब also the relation label nmod_adj is an *intraChunk* label). The parent of the chunkhead must also be a chunkhead from another chunk. The relation should be marked with interChunk label. (in the given example, the chunkHead tokens दी, राम, सीता and किताब are attached with a chunkHead token also the relation labels are interChunk labels). However, in the training data, there are few noisy cases where intraChunk relations are marked as interChunk and vice-versa. The cases are very less in number and hence were ignored.
Dividing the data into inter and intra chunk, all the above constraints will be automatically handled. In the resulting intraChunk data, the chunks formed will behave as an individual sentence therefore the parser would not be able to make an inappropriate arc.

Experiment with projectivity: MaltPaser has a default constraint to give only projective output. However, in the training data we find approximately 1.1% arcs to be non-projective. To address the non-projectivity in data, we use pseudo-projective algorithm as proposed by Nivre et al, 2005 [78]. We only incorporated the pseudo-projective algorithm in case of inter-Chunk data as in intracohort we found the arcs to be always projective. There are three options available with the pseudo-projective algorithm in MaltParser. We performed intermediary six experiments on all of these and got some interesting results.

Pseudo-projective algorithm replaces all the non-projective arcs in the input data to projective arcs by applying a lifting operation. The lifts are encoded in the dependency labels of the lifted arcs. In order to apply an inverse transformation to recover the underlying (non projective) dependency graph, there is a need to encode information about lifting operations

in arc-labels. The encoding scheme can be varied according to **marking_strategy** and there are currently five of them: **none, baseline, head, path** and **head+path** [78]. We performed intermediary experiments separately on each of them and observed that **head** option gives the best result. This option projectivizes input data with head encoding for labels.

Secondly, there is an option called **covered_root**, which is mainly used for handling dangling punctuation. This option has five values: **none, ignore, left, right** and **head**. In our intermediary experiments, we found that ignore gave better results than others.

On the basis of lifting order, there are two ways to lift the non-projective arcs namely, **shortest** and **deepest**. In the deepest lifting order, most deeply nested non-projective arc is lifted first, not the shortest one. In our experiments we found that deepest has no effect in increasing the parsing accuracy rather there is a slight decrease in the accuracy as compared to shortest.

Experiment with features: We try to experiment with the types of features that can be used in FEATS column in the CoNLL-X format. We considered four ways: 1)without any information in FEATS column 2) only TAM[1] and Vubhakti information, 3)tam and vib along with chunkType and 4)with all the information present by default. The best results were obtained using all information. All this could only be done for the Gold track as such information about the features is not provided for Automatic track. This is the major reason for the difference in the parsing accuracies between gold and auto data.

Experiment with algorithms: Kolachina et al., 2010 [53] have shown that **nivre_eager** algorithm gives the best accuracy for Hindi. Our intermediary experiments also support the same. We also explored the root-handling option which can be **normal, strict** and **relaxed**. Our experiments showed that relaxed option gives the best accuracy. In relaxed option, root dependents are not attached during parsing (attached with default label afterwards) and reduction of unattached tokens is permissible.

Experiment with prediction strategy:
There are three types of **prediction strategies** available in MaltParser :

1. **combined** (default): Combines the prediction of the transition and the arc label .

2. **sequential**: Predicts the transition and continues to predict the arc label if the transition requires an arc label.

[1] tense,aspect and modality

3. **branching**: Predicts the transition and if the transition does not require any arc label then the non determinism is resolved, but if the predicted transition requires an arc label then the parser continues to predict the arc label.

We perform experiments with the above options and find that using **branching** there is an increase in the parsing accuracy.

Experiment with SVM settings: In our experiments, we use the LIBSVM learner algorithm following the SVM settings (s0t1d2g0.12r0.3n0.5m100c0.7e0.5) in experiments reported by Kolachina et al. [53] for Hindi. These settings give a better result over the default SVM settings.

Results: We train MaltParser separately using Gold and Auto training data. For gold data, we train two models, one for interChunk data with all settings obtained in the above experiments and other for intraChunk data with all the above settings except branching and projectivization. For both we use the same algorithm **"nivre_eager"** and learner **"LIBSVM"**. The final evaluation, the system demonstrates LAS is **90.99%**, UAS is **95.87%** and LA is **92.58%** respectively. For Automatic data, we don't split the data in two parts as the information on which the data is divided is missing in the testing files. Except this, all the other settings are exactly similar as for the gold data. The final LAS is **83.91%**, UAS is **91.70%** and LA is **86.77%** respectively.

3.1.2 Error Analysis

correct label	**system output label**	**frequency**
pof	k2	139
k1	k2	123
k2	pof	112
k7	k7p	95
k7p	k7	88

Table 3.2: Confusion between labels

Table 3.2 shows the top 5 most frequent errors our parser makes. The most frequent errors that the parser makes contain the confusion between marking of k2, k1 and pof dependencies. The confusion between k1 and k2 is because of the absence of the case markers for disambiguation. As pof is the verbal form of noun, it is even difficult for humans to disambiguate between

pof and k2. The confusion between k7 and k7p is also frequent because of their closeness. Some of these errors can be handled by introducing a rule based step in post-processing.

It has been previously shown that some of these errors can be handled if additional semantic information about the words is provided[13]. Motivated by this work, in the next section we try to handle the errors using additional semantic features automatically extracted from Hindi WordNet.

3.2 Exploring Semantic Information in Hindi WordNet for Hindi Dependency Parsing

This section is a compilation of different parts (which relate to the work in this thesis) of our published work [48], with Sambhav Jain being the main author of the paper. The ideas presented in this section are important to show how rich features like semantic information, help improve parsing quality. My contribution in this work is mainly, different strategies of feature extraction from Hindi Wordnet.

Last decade has witnessed several efforts towards developing robust data driven dependency parsing techniques [57]. The efforts, in turn, initiated a parallel drive for building dependency annotated treebanks [93] , which serve as a data source for training data driven dependency parsers. The annotations are often multi-layered and furnish information on part of speech category of word forms, their morphological features, related word groups and the syntactic relations. The availability of such rich resources have considerably improved the parsing performance of syntactic parsers [25]. However, the error analysis studies carried out on these parsers later revealed that certain syntactic relations are difficult to deduce and disambiguate with the syntactic information available in the annotated treebanks.

The need for richer information invoked several efforts in the direction of annotating higher order linguistic information in treebanks. It was felt that semantics can be leveraged for syntactic disambiguation and thus semantic annotation was performed in syntactic treebanks to complement the morpho-syntactic annotations [51] [71]. Fujita et al. (2007) [42] and MacKinlay et al. (2012) [60] illustrated that semantic annotation delivers a significant improvement in parsing, confirming the hypothesis that semantics can assist syntactic analysis. Among Indian languages, notable efforts on using semantic information in dependency parsing are on Hindi. Bharati et al. (2008) [13] illustrated that mere animacy (human, non-human and inanimate) of a nominal significantly improves the accuracy of the parser. Later studies on extending such information with finer semantic distinctions like time, place, abstract reconfirmed the substantial role of semantics in syntactic parsing [3]. These studies are carried out on a dataset with hand annotated semantics. Although these studies provide deep insights on the role of semantics in

parsing, they are limited in application as such information can not be automatically generated while parsing new sentences.

We make an effort to supply the aforementioned semantic information by employing concept hierarchy available in Hindi WordNet (HWN). Attempts have been made to utilize hand annotated semantic information for constituency parsing [42] [60] as well as dependency parsing [79] [13] [3]. However, acquiring such information for new sentences remains a challenge. This leads us to the exploration of lexical databases and ontologies for accessing semantic information useful for parsing. Xiong et al. (2005)[96] used two lexical resources HowNet[2] [36] and TongYiCi CiLin [64]. Agirre et al. (2008)[2] demonstrated that semantic classes obtained from English WordNet[67] help to obtain significant improvements in both PP attachment and PCFG parsing. Similarly, for dependency parsing, Agirre et al. (2011) [1] utilized the English WordNet semantic classes and improved parsing accuracies.

Challenges: In Chapter 2, we have discussed the challenges involved in parsing Indian languages. The most important of them is the *case syncretism*. The challenges include the problem of Case Ambiguity, Lack of Case Marker, Complex Predicates etc. We do experiments and see if these problems are handled.

3.2.1 Hindi WordNet and Concept Ontologies

Hindi WordNet is a lexical database developed on the lines of English Wordnet, under the Indo WordNet project [72]. For each lexical item, Hindi WordNet defines a synset which enlists its synonyms. Further, each synset is mapped to a concept ontology. The concept ontology is a hierarchical organization of concepts like entities, actions etc. which defines the semantic properties of lexical items of a given synset. The ontology consists of around 200 different concepts. The lexical item is the leaf node in this hierarchical construct. As we move up the hierarchy, the specific semantic aspects of a given lexical item are unraveled. The hierarchy terminates, immediately after capturing the syntactic category of a word, at the *TOP* node. The *TOP* acts as a *root*, holding the hierarchies of all the lexical items listed in HWN. Figure 3.3 illustrates a typical hierarchy in this ontology, where *Ape* is the most explanatory node. As we move up, it becomes more and more generic. Further, the relations between different synsets are captured based on the following paradigms :

- Semantic (hypernymy, hyponymy, meronymy etc.)
- Lexical (antonymy, synonymy etc.)
- Gradience (size, quality, manner etc.)

[2] `http://www.keenage.com`

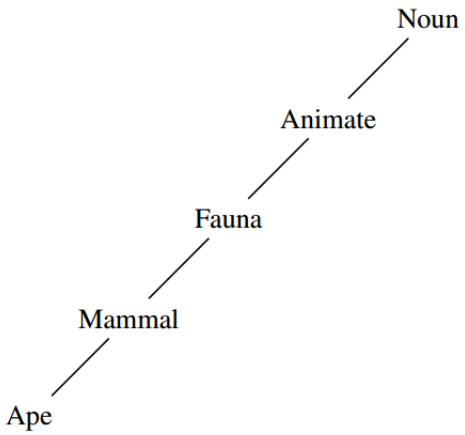


Figure 3.3: Sample Hierarchy of Concepts in Hindi Wordnet

3.2.2 Feature Extraction

In this section, we explore the extraction of features from HWN corresponding to the lexical items in our data. We also address the issues like sense selection and coverage.

1. **Sense Selection** Attributed to the phenomenon of lexical ambiguity, a lexical item can have senses varying across different contexts. Although HWN lists all the possible senses of a lexical item, to choose the contextually appropriate sense is a challenging task. Here, we discuss our approach to select the sense of a lexical item best suited in a given context.

    - *Category Based Sense Selection:* Consider a word *chaat* (चाट, it can either mean 'lick' or 'snacks'. The former corresponds to a verb while the latter is a nominal as depicted in Figure 3.4. The syntactic category of a lexical item provides an initial cue for the sense selection. Among the varied senses, we filter out the senses that do not fall into its syntactic category.
    - *Intra − Category Sense Selection:* : As a matter of fact, words are ambiguous not only across different syntactic categories but also within same category as depicted in Figure 3.5. Once the senses of a lexical item are filtered based on its syntactic category, within category senses, if many, are investigated for the best sense based on the following strategies:
        - *First Sense:* Among the varied senses, we select the first sense listed in HWN corresponding to the POS-tag of a given lexical item. The choice is motivated by our observation that the senses of a lexical item are ordered in the descending order of their frequencies of usage i.e., the first sense listed in HWN is the predominant sense of a given lexical item.

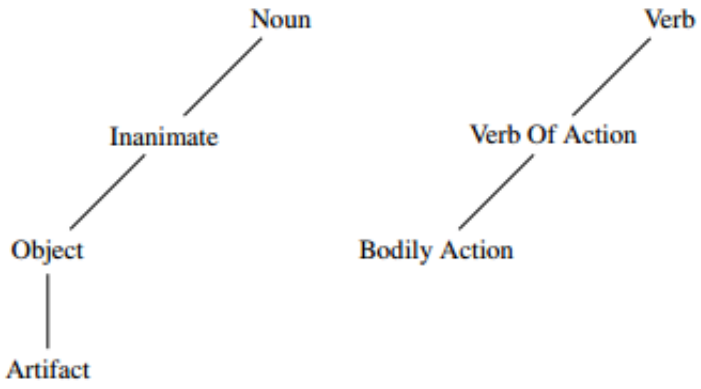


Figure 3.4: Nominal and Verb Sense of *chaat*

– *WSD:* Although first sense captures the predominant usage of a lexical item, it is inappropriate for its other infrequent usages. We, therefore, need to pick the contextually appropriate sense of a lexical item. To this end, we exercise Extended Lesk, a classical word sense disambiguation algorithm [7]

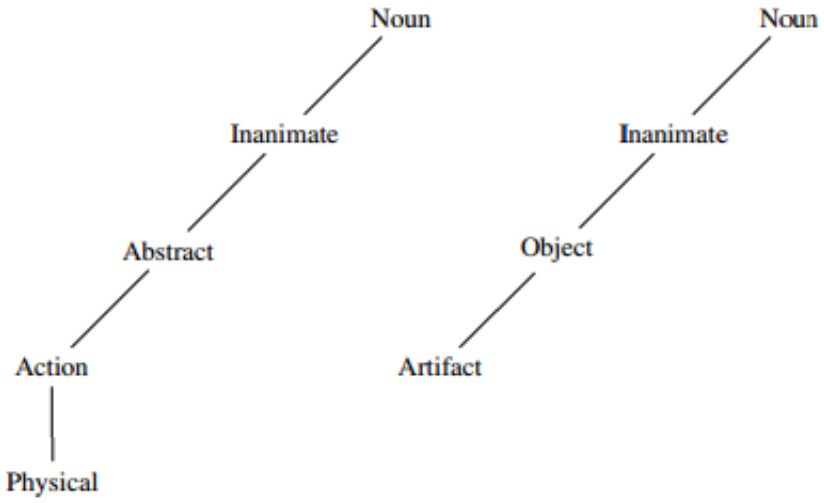


Figure 3.5: Two senses for the nominal *chaat*

2. **Numeric Expressions:** As is obvious, no lexical resource can have an exhaustive coverage because of the evolving nature of human language. In the context of HWN, the problem further intensifies as it restricts the entry to only words of open class syntactic categories. Apart from that, it also has a limited coverage for numeric expressions as

these expressions belong to an infinite set. Numerals can be used in wide range of senses. Apart from their simple ordinal or cardinal usages, they can also be used as nominals in expressions like time and measurement. In their adjectival sense, WN features can be extracted corresponding to the head word they modify e.g., the temporal sense of an expression *10 saal* (१० साल) can be identified by the head word *saal* 'year'. However, to identify the temporal sense of a numeral, used as nominal, like 2013 is challenging. We use a numeric-expression recognizer, built in-house, to identify measurement and temporal expressions. The tool makes use of regular expressions and cue words. Once identified, we assign them an appropriate HWN ontological hierarchy which either corresponds to *time, measurement* or *number*.

3. **Complex Predicate as a Feature:** Complex predicates (CPs, also known as complex verbs) are highly frequent in South Asian languages [70]. They occur in the form of nominal+verb combinations (called conjunct verbs) and verb+verb combinations (called compound verbs). For example, `शरण लेना' (refuge take) is a complex predicate composed of a nominal `शरण' and a light verb `लेना'. The constituents of a complex predicate are related by a dependency relation *pof* in HDT. In Hindi dependency parsing, the major chunk of parse errors is attributed to the low learnability of complex predicates[46]. Begum et al. (2011)[9] addressed the identification of these expressions using some linguistic rules. Fortunately, HWN has listed a finite set of these expressions in its database [20]. We first extract the multi word expressions listed in HWN if the last word in the expression is a verb. Then from the list only 2-word expressions are selected and treated as complex predicates. Instead of adding WN features to the nominal of a complex predicate, we assign a separate *CP* tag to it. The semantics of light verbs is, however, kept as such.

3.2.3 Feature Design

After the extraction of WN features, we explore possibilities of their design and incorporation in the parsing framework, as follows.

1. **Grouping Similar Features:** We observed that few concept ontological lineages are semantically similar. For example, the six lineages depicted below address the notion of time.

   - *Time*
   - *Descriptive*$\longrightarrow$*Time*
   - *Inanimate*$\longrightarrow$*Abstract*$\longrightarrow$*Time*
   - *Inanimate*$\longrightarrow$*Abstract*$\longrightarrow$*Time*$\longrightarrow$*Period*
   - *Inanimate*$\longrightarrow$*Abstract*$\longrightarrow$*Time*$\longrightarrow$*Season*

- *Inanimate⟶Abstract⟶Time⟶Mythological Period*

Since our focus is on adding representative semantic features which can assist parsing, we believe that such divergences should be grouped together. In the listed example, first, second and the last four differ in terms of their origin and belong to different branches in the hierarchy. Thus they can not be grouped by optimal depth selection (described later) and requires a manual scrutiny. We studied the possible lineages in the concept ontology and performed merging wherever necessary, furnishing a semantically well diverse set of concept lineages.

2. **Split Vs Conjoined:** The concept lineage, derived for a word from HWN concept ontology, contains diverse concepts at each level of the lineage. The choice of using each of these concepts as independent features or the complete lineage as a single feature demands exploration. In the context of parsing, each independent concept from the lineage can potentially capture a specific aspect of syntax, depending on the fineness of the concept. The down side of this proposition is the increase in the feature dimensions, as each level adds a new dimension in the feature space. Whereas, using the complete lineage as a single feature does not add any additional dimension in the feature space but captures only a specific concept. This trade off is difficult to comprehend on theoretical grounds, hence we explore both choices of feature design in our experiments

3. **Ontology Depth:** Hindi WordNet concept ontology furnishes a 'generalization hierarchy' for a lexical item, where the specificity of concepts increases as we move down the hierarchy. It may look intuitive to use fully expanded concept lineage, as it contains more detailed description of the lexical unit. However, opting for a highly fine-grained concept lineage leads to the problem of sparseness. It becomes less and less probable to find ample training examples as the feature becomes more fine-grained. At the same time, too much generalization is also unrewarding since the richer information is cast away in the excessive coarser lineage. This calls for measures to obtain an optimal depth of concept lineage for each lexical item. On one hand it should be generalized enough to give significant examples of its respective type while on the other hand, it should be fine enough to capture the rich ontological concept associated with the lexical unit. In order to quantify the trade-off we resort to statistical correlation measures and employed *Gini Coefficient*[44]. We computed the coefficient against all possible concept lineages in the training set and set a threshold. The lineages that fall below the threshold are generalized till they are above the threshold. For example, in Figure 3.3 the concept *ape* is suppressed to give the lineage till *mammal* only. So in future if a word gives the lineage as in Figure 3.3 it will be replaced with its one level up generalization i.e. *Animate⟶Fauna⟶Mammal.*

3.2.4 Experiments and Results

In our experiments, we focus on establishing dependency relations between the chunk heads i.e. *inter-chunk* parsing. The relations between the tokens of a chunk (*intra-chunk* dependencies) are not considered for experimentation. The decision is motivated by the fact that the *intra-chunk* dependencies can easily be predicated automatically using a finite set of rules [54]. We also observed the high learnability of *intra-chunk* relations in 3.1. We found the accuracies of intra-chunk dependencies to be more than 99.00% for both Labeled Attachment and Unlabeled Attachment. In this section, we present our parsing experiments incorporating the features extracted from HWN, as discussed earlier. First we setup our baseline parser followed by the detailed discussion on the impact of the individual features, extracted from HWN, on the overall parsing performance. ormance. We setup our baseline parser on the lines of [84] (as described in section 3.1 with minor modifications in the parser feature model. We employ MaltParser version-1.7. and Nivre's Arc Eager algorithm for all our experiments reported in this work. All the results reported are evaluated using eval07.pl [3]. Among the features available in the FEATS column of the CoNLL format data, we only consider *Tense, Aspect, Modality (tam)* and *postpositions* while training the baseline parser. Other columns like POS, LEMMA, etc. are used as such. After the baseline, the parsing framework is further enriched with the semantic features extracted from HWN to address the problems raised. These features are added in the FEATS column of the data, separated by '|'. In a pilot experiment split form of features, as discussed earlier, are found to perform better than conjoined form, which motivate us to use WN feature in split form in all our experiments. The experimentation proceeds in the order as listed in Table 3.3 which also presents the consolidated results of our parsing experiments using the training and testing sets. Next we discuss the impact of WN features on the accuracy of our parsing results produced on datasets of different sizes:

- *Sense Selection:* We performed two experiments to extract the WN features corresponding to the most appropriate sense of a lexical item. In the first experiment, the first sense of each lexical item is selected while in the second, WSD is used to pick the contextually most appropriate sense. These features corresponding to the chosen sense are coupled with the features already present in the baseline. As depicted in Table 3.3, though there is an increase in accuracy using first sense strategy from the baseline, using WSD the accuracy decreased. As is obvious, the fall in accuracy can be attributed to the wrong sense selection. The problem can be addressed using better WSD algorithms for Hindi.

- *Numeric Expressions:* Numeric expressions and sense grouping increases the coverage of HWN. The obvious reason is clearly depicted in the improvement in parsing results as shown in Table 3.3. More the semantic information available in the data, more will be its impact on the parsing.

[3] `http://nextens.uvt.nl/depparse-wiki/SoftwarePage/#eval07.pl`

	Ezxperiments	LAS(%)	UAS(%)	LS(%)
E1	Baseline	82.51	92.04	85.37
E2	E1 + First Sense	82.64	92.03	85.53
E3	E1 + WSD (Extended Lesk)	81.96	91.62	84.42
E4	E2 + Numeric Expressions & Grouping	82.81	92.22	85.67
E5	E4 + Ontological Depth	83.90	92.35	85.72
E6	E4 + Complex Predicate	82.90	**92.95**	**85.74**
E7	E5 + E6 (Complex Predicate + Ontological Depth)	**82.91**	92.32	85.67

Table 3.3: Results for Hindi Dependency Parsing using semantics from HWN

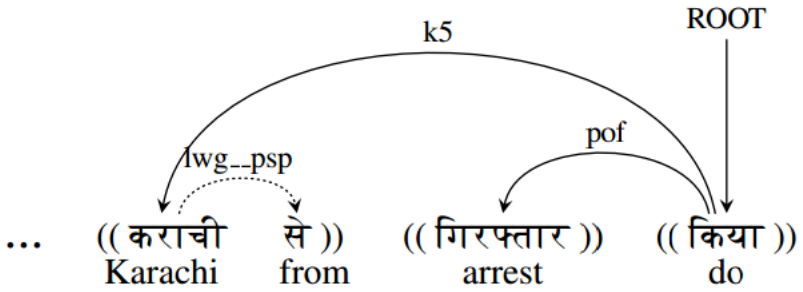
- *Depth of Information:* The optimality of feature coarseness is put to test in this experiment. This experiment is run on numeric expression data with feature pruningm done as described in *Ontology Depth*. An increment of 0.09% LAS is observed from the previous experiment. This increase in the accuracy can be accredited to the optimality achieved in the features.

- *Complex Predicate:* As pointed in [9], addressing the low learnability of complex predicates can improve the parsing results. The improvements are particularly seen in the core arguments of a verb. The similar syntactic distribution of adjectival or nominal element of a complex predicate and the syntactic arguments of a verb particularly *objects*, make these expressions highly ambiguous. Identifying these expressions beforehand, as suggested in [9], improves the parsing performance. The incorporation of this crucial information from HWN has proved rewarding. We achieved an improvement of approx. 1% in UAS.

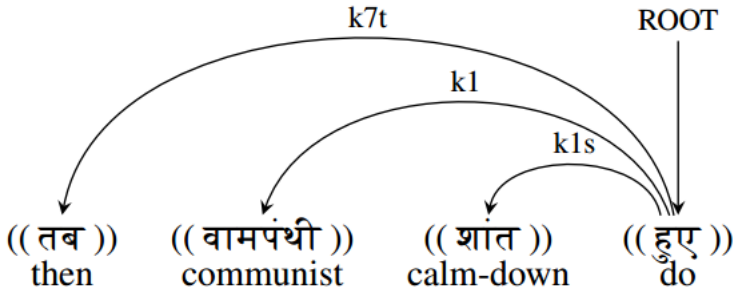
3.2.5 Discussion

In this section, we discuss further, how well the issues raised in Chapter 2 (Section 2.4.1) are handled by the incorporation of semantic information in the parsing framework of Hindi. In Chapter 2 we stated that ambiguities in morphological cases in Hindi bar their efficient exploitation while parsing. Also we noted that unmarked nominals may as well affect the performance of a parser. So we propose semantics as a complementing information that can fill these gaps. Below we discuss whether semantic information has bridged these gaps or not.

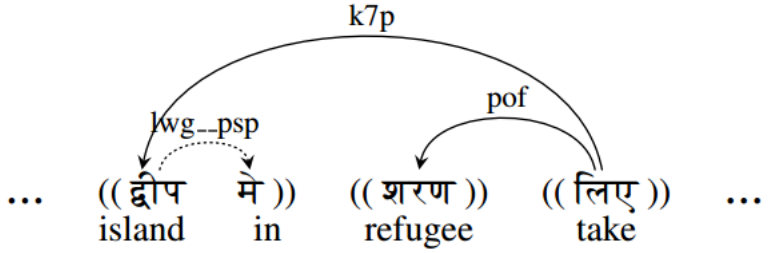
- *Case Ambiguity:* Including the semantics from HWN to help disambiguate the confusion present in a case marker, has improved parsing accuracy. Particularly confusion among



the roles of concrete vs abstract time and place, and direct vs indirect object relations has been removed. In the above example, the dependency relation between nodes *Karachi* and *do* has been corrected from *k2* 'Theme' to *k5* 'Source'. The post-position *from* can either mark a theme or a source relation. Semantics has removed this confusion.



- *Lack of Case Marker:* In absence of case marking lexical semantics acted as a complementing information. The improvement has been, as observed during error analysis, particularly for agents and patients. Thus semantics can be seen here as pseudo case markers. This is clearly visible from the above example. The dependency relation between the nodes *then* and *do* has been corrected to *k7t* 'time of action' from *k1* 'subject'.



- *Complex Predicates:* In example above, baseline incorrectly identifies *refuge* as an argument of verb *take*. 'refuge take' is a complex predicate which is correctly identified upon incorporation of complex predicates in our parsing module.

3.3 Summary

In this chapter we have seen that if resources like Treebank are available for a language, a good quality parser can be built. We have also seen that if rich information like semantic information from wordnet is provided, it helps improve parsing quality further. But what if we do not have these resources we have used, treebank with manually annotated features and dependency relations and manually created Wordnet? In the next chapter, we explore different strategies to make the task of dependency parsing feasible even with the less resources. We exploit cheap resources and make the process cheaper.

Chapter 4

Exploiting cheap resources for the cost effective dependency parsing

This chapter focuses on different strategies to reduce the overall cost of dependency parsing. Till now, we tried to improve the parsing quality for Hindi using available treebank. In Chapter 3, we set up a state-of-the-art parser for Hindi, by exploiting rich resources like Hindi WordNet. The resources we used, the treebank and the WordNet are expensive. In this chapter, we try to use cheap resources like already built treebanks of other languages, monolingual raw corpus to reduce the overall cost of the Parsing Process. In section 4.1 we try to produce treebank for Hindi using other available treebanks by the method of cross-lingual treebank transfer. In section 4.2 we try to make use of large mono-lingual corpus, which can be made easily available, to replace the costly tools used to extract features with cheap features.

4.1 Cross-Lingual Dependency Parsing using UD Framework

Cross-lingual learning has become a popular approach to facilitate the development of resources and tools for low-density languages. Its underlying idea is to make use of existing tools and annotations in resource-rich languages to create similar tools and resources for resource-poor languages. Typically, this is achieved by either projecting annotations across parallel corpora, or by transferring models from one or more source languages to a target language. We study a third strategy of using machine translation to create synthetic training data from the original source side annotations. The approach is described by Jorg Tiedemann et al. (2014) [92]. They apply this technique to dependency parsing, using a cross-lingually unified treebank for adequate evaluation. This approach draws on annotation projection but avoids the use of noisy source-side annotation of an unrelated parallel corpus and instead relies on manual treebank annotation in combination with statistical machine translation, which makes it possible to train fully lexicalized parsers. It is shown that this approach significantly outperforms delexicalized transfer parsing. We implement this technique for parsing Hindi and study if

this approach is suitable for Indian languages, it's benefits and drawbacks in our scenario. We find out that this technique is not suitable for the pair of languages having different structural properties, specifically the word order. We try to come up with different strategies to handle this problem and study another work which also tries to tackle the same problem.

In the next few sections we will briefly discuss the various Cross-Lingual Approaches, Universal Dependencies, The translation approach, Issues in the translation approach and different strategies to tackle the issues.

4.1.1 Various approaches

1. **Annotation Projection:** Annotation projection relies on the mapping of linguistic annotation across languages using parallel corpora and automatic alignment as basic resources [99] [47] [88]. Tools that exist for the source language are used to annotate the source side of the corpus and projection heuristics are then applied to map the annotation through word alignment onto the corresponding target language text. Target language tools can then be trained on the projected annotation assuming that the mapping is sufficiently correct. Less frequent, but also possible, is the scenario where the source side of the corpus contains manual annotation. This addresses the problem created by projecting noisy annotations, but it presupposes parallel corpora with manual annotation, which are rarely available, and expensive and time-consuming to produce.

2. **Model Transfer:** Model transfer relies on universal features and model parameters that can be transferred from one language to another. Abstracting away from all language-specific parameters makes it possible to train, e.g., delexicalized parsers that ignore lexical information. The Delexicalised Parsing technique can be used for multilingual syntactic parsing. In contrast to annotation projection approaches, delexicalized transfer methods do not rely on any bitext. Instead, a parser is trained on annotations in a source language, relying solely on features that are available in both the source and the target language, such as universal part-of-speech tags [100] [73] [73] [82], cross-lingual word clusters [90] or type-level features derived from bilingual dictionaries [38]. This parser is then directly used to parse the target language. For languages with similar typology, this method can be quite accurate, especially when compared to purely unsupervised methods. For instance, a parser trained on English with only part-of-speech features can correctly parse the Greek sentences, even without knowledge of the lexical items since the sequence of part-of-speech tags determines the syntactic structure almost unambiguously. Learning with multiple languages has been shown to benefit unsupervised learning [24] [85] [10]. Annotations in multiple languages can be combined in delexicalized transfer as well, as long as the parser features are available across the involved languages.

3. **Translation Approach:** The translation approach [92] is based on automatically translating training data to a new language in order to create annotated resources directly from the original source. Recent advances in statistical machine translation (SMT) combined with the ever-growing availability of parallel corpora are now making this a realistic alternative. The relation to annotation projection is obvious as both involve parallel data with one side being annotated. However, the use of direct translation brings two important advantages. First of all, using SMT, we do not accumulate errors from two sources: the tool – e.g., tagger or parser – used to annotate the source language of a bilingual corpus and the noise coming from alignment and projection. Instead, the gold standard annotation of the source language is used. Moreover, using SMT may help in bypassing domain shift problems, which are common when applying tools trained (and evaluated) on one resource to text from another domain. Secondly, SMT will produce output that is much closer to the input than manual translations in parallel texts usually are. Even if this may seem like a short-coming in general, in the case of annotation projection it should rather be an advantage, because it makes it more straightforward and less error-prone to transfer annotation from source to target. Furthermore, the alignment between words and phrases is inherently provided as an output of all common SMT models. Hence, no additional procedures have to be performed on top of the translated corpus.

4.1.2 Universal Dependency Treebank

Universal Dependencies (UD) [1] is a project that is developing cross-linguistically consistent treebank annotation for many languages, with the goal of facilitating multilingual parser development, cross-lingual learning, and parsing research from a language typology perspective. The annotation scheme is based on an evolution of (universal) Stanford dependencies [32] [33] [31], Google universal part-of-speech tags [82], and the Interset interlingua for morphosyntactic tagsets [100]. The general philosophy is to provide a universal inventory of categories and guidelines to facilitate consistent annotation of similar constructions across languages, while allowing language-specific extensions when necessary. We use the Universal Dependency Treebank v1 for annotation projection, parser training and evaluation.

As we are working on Hindi, the first task is to convert the available Hindi Treebank into UD framework[2], which in itself a tedious task. This is done using following the guidelines available in the UD Documentation for conversion of treebanks. First step is the mapping of language specific POS tags with the Universal POS tags. Universal POS categories have substantive definitions and are not necessarily just equivalence classes of categories in underlying language-particular treebanks. Hence, work to convert to UD POS tags often requires context-sensitive

[1] http://universaldependencies.github.io/docs/

[2] The Hindi UD treebank can be downloaded from http://github.com/UniversalDependencies/UD_Hindi

rules, or some hand correction. Morph features are also converted. An approximate conversion table can be created easily using Interset tool.

The most important step is to map the dependency relations with the universal dependencies. This step is more difficult in case of Indian Languages, as the Grammatical Framework used to annotate Treebanks in Indian Languages is the CPG, as seen in the Chapter 2. Indian Languages (ILs) including Hindi are morphologically rich and have a relatively flexible word order. For such languages syntactic subject-object are not able to explain the varied linguistic phenomena. In fact, there is a debate in the literature whether the notions 'subject' and 'object' can at all be defined for ILs [68]. Behavioral properties are the only criteria based on which one can confidently identify grammatical functions in Hindi [69]; it can be difficult to exploit such properties computationally. But the notion of 'subject','object' is central to most of the other languages. The aim of UD framework is to help find most common syntactic analysis of languages and thus it is very different from the CPG framework. This makes the task of mapping relations to generic relations difficult.

Once the converted treebank is ready, we can conduct various experiments to derive the treebank in Hindi using various Cross-Lingual approaches and compare it with the converted gold treebank. Next we do different experiments to get the Hindi treebank using other treebanks included in UD treebank and train parsers using the cross-linguistically generated treebanks.

4.1.3 Experiments

1. **Delexicalised Paersing:** We start experimenting with the delexicalised parsing. We use the Universal Dependency Treebank v1 for this purpose. It is a collection of data sets with consistent syntactic annotation for six languages: English, French, German, Korean, Spanish, and Swedish. First we concatenate all the six treebanks to get a single large treebank. Then all the lexical items, lemmas and other lexical level features except POS tags are removed. So now we have a large treebank which is annotated with POS tags and dependency relations only. We then train a parser on this treebank using MaltParser. We remove the features and lexical items from the converted Hindi treebank. We use the trained model to parse the Hindi treebank and to get automatic dependency relations. We evaluate the results by comparing the system output with the Gold Hindi Treebank in UD.

2. **Using Statistical Machine Translation:** Here we follow the strategy explained in "Translation Approach". As this method uses SMT system for translation, we need to have a parallel corpus with target language being Hindi. In our work we experiment with English-Hindi system, as the parallel corpus required to create the MT system is readily available and the size of English Treebank is large enough for the purpose of studying cross lingual treebank transfer. So, first we train a SMT system using English-Hindi parallel

	UAS	LS	LAS
Delexicalised	65.65	70.08	54.11
SMT based	74.2	72.8	61.5

Table 4.1: Results for Cross Lingual parsing

corpus. We use MOSES toolkit [52] for SMT. We first translate all the sentences in English treebank to Hindi using the SMT system. We also acquire the automatic word alignment produced during the translation of sentences. We then transfer the dependency relations in source sentence to the target sentence using the approach described by Jorg Tiedemann et. al.(2014) [92], which makes use of the automatic word alignment to map the source and target words. This approach helps in handling the minor structural differences between languages, as the SMT also re-orders the words in the local context. Another benefit of this approach over the delexicalised approach is that, here we also have the word forms as a lexical level information (along with POS tags), which is not present in the delexicalised parser.

For evaluation of this strategy, we keep the lexical items as it is, in the automatically generated treebank, unlike the delexicalised parsing approach. We train a parser on this treebank and run the parser on the original Hindi treebank sentences. We compare the produced results with the original tree to get the evaluation scores.

4.1.4 Results

Table 4.1 shows the results for different experiments. We can see that the results are not quite satisfactory and seems unusable for the real world applications.(Note: the results are on the full data. We did not break the data in interChunk and intraChunk). The problem is not in the approach but the choice of approach for the given language. In the next section, we study why it didn't work for Hindi.

4.1.5 Discussion

One important thing to notice here is that all the languages which are included in the UD treebank V1 have S-V-O word order whereas Hindi is relatively free word order with S-O-V being preferred. If we talk about the Delexicalised Parser, only the POS tags can not capture this structural divergence between the languages. And all the languages in training data being SVO, the parser gets biased towards the SVO order and hence can not efficiently parse the Hindi POS sequence, which are in SOV order. Even the SMT system can only handle the short distance re-ordering of words, so the translation method also does not work very well for us.

One method to overcome the problem, of structural differences in case of Delexicalised Parsing is to choose the source languages wisely. Instead of just concatenating all the treebanks with equal preference, more weight could be given to the languages which are syntactically more similar to the target language. There are different ways to weigh the source languages. Søgaard and Wulff (2012)[87] introduced weighting into the method, using a POS n-gram model trained on a target POS-tagged corpus to weight source sentences in a weighted perceptron learning scenario[19]. Rosa and Zabokrtsky (2015), ported the method to a crosslingual setting by combining delex parsers for different languages, weighted by src-tgt language similarity. They introduced KLcpos3 – a language similarity measure for delexicalized parser transfer.

Another method is "Target Language Adaptation of Discriminative Transfer Parsers"[89] which involves selective parameter sharing based on typological and language-family features, applying to a discriminative parser by carefully decomposing its model features. Another method we suggest, is to reorder the source language trees to make the it more similar to target language structure, using some re-ordering rules based on dependency relations.

As we have already seen, for Indian Languages, we use CPG formalism for dependency annotations, which is very different from the other grammar formalisms and hence using UD framework for Indian Languages we wouldn't gain much information even if we get great accuracies. But as we have stated, for almost all the Indian languages we follow same formalism, moreover the structural differences between Indian languages is less compared to other languages, so it would be a good idea to come up with a common representation like UD, specifically for Indian languages.

4.2 Exploring Vector Space Models for Hindi Dependency Parsing

Word Embeddings have shown to be useful in wide range of NLP tasks. We explore the methods of using the embeddings in Dependency Parsing of Hindi, a MoR-FWO (morphologically rich, relatively freer word order) language and show that they not only help improve the quality of parsing, but can even act as a cheap alternative to the traditional features which are costly to acquire. In this chapter, we demonstrate that if we use distributed representation of lexical items instead of features produced by costly tools such as Morphological Analyzer, we get competitive results. This implies that only mono-lingual corpus will suffice to produce good accuracy in case of resource poor languages for which these tools are unavailable. We also explored the importance of these representations for making parser domain independent. The work is driven by the insight that word embeddings from vector space models capture some real world aspects of lexical items which is quite distinct and unlikely to be deduced from morpho-syntactic information such as morph, POS-tag and chunk.

As explained in earlier chapters, Hindi is a MoR-FWO language. It exerts a relatively free word order with SOV being the default configuration. Due to the flexible word order, dependency representations are preferred over constituency for its syntactic analysis (Bharati and Sangal, 1993) [15]. The dependency representations do not constrain the order of words in a sentence and thus are better suited for flexible ordering of words. The dependency grammar formalism used for Hindi is *Computational Paninian Framework*(CPG) (Begum et al., 2008; Bharati et al., 2009) [8, 16]. The dependency relations in CPG formalism are closer to semantics and hence are also reffered as *syntactico-semantic relations*.

In Chapter 3, we have shown how efforts towards building dependency annotated treebanks started (Tsarfaty et al., 2013) [93], which serve as a data source for training data driven dependency parsers. The annotations are often multi-layered and furnish information on part of speech category of word forms, their morphological features, chunking related words and syntactic relations. But error analysis show that the marked information is not enough for good quality parsing. As experimented in Chapter 3, the efforts towards adding richer information to the treebank started to help parser disambiguate syntactic relations and suggested that semantic information could help parsing (Jain et al., 2013) [48].

Annotating treebank with rich information improves the results but annotation in itself is a very costly task in terms of both, time and manual work. Hence, manually annotated data can not be made available for all the languages. Even if treebanks with rich information are available, it is difficult to accurately generate these features automatically in real time parsing. The tools used to generate these features such as morph analyzer, WordNet (in Case of semantic information), etc. are also costly to build. Resource poor languages do not have these tools readily available. The only thing very easily available for any language is the mono-lingual text corpus. Efforts are being made to exploit the use of mono-lingual corpus to capture all this information in word-embeddings using continuous vector space models. Using word-embeddings as features have shown to be useful and an easy replacement to the costly features in different NLP tasks. We try to exploit the same for the task of Hindi Dependency Parsing.

We show that the word embeddings learned from vector space models can replace almost all the features and hence costly tools and using it as a stand-alone feature can produce better results. We also show that these features when used as complementary features instead of replacement, improve the results further. Another outcome of our experiments is the improvement in the parsing quality when the training and testing data are from different domains.

4.2.1 Related Work

(Chen and Manning, 2014) [22] have recently proposed a way of learning a neural network classifier for use in a greedy, transition-based dependency parser. They used small number of dense features, unlike most of the current parsers which use millions of sparse indicator features. The small number of features makes it work very fast. Use of low dimensional dense

word embeddings as features in place of sparse features has recently been used in many NLP tasks and successfully shown improvements in terms of both, time and accuracies. The recent works include POS tagging (Col- lobert et al., 2011) [26], Machine Translation (Devlin et al., 2014) [35] and Constituency Parsing (Socher et al., 2013) [86]. These dense, continuous word-embeddings give strength to the words to be used more effectively in statistical approaches. The problem of data sparsity is reduced and also similarity between words can now be calculated using vectors.

4.2.2 Data & Tools

4.2.2.1 Hindi TreeBank

In these experiments we use a newer version of Hindi treebank, which is different from the treebank that we described in Chapter 2. This newer treebank is larger in size and some of the errors have been corrected. Here, we give an overview of Hindi Treebank (Hindi DTB). Pre-release version of Hindi Dependency Treebank data has been made available for download for researchers [3]. It is a multi-layered dependency treebank with morphological, part of speech and dependency annotations based on the CPG. CPG provides an essentially syntactico-semantic dependency annotation, incorporating karaka (e.g., agent, theme, etc.), non-karaka (e.g. possession, purpose) and other (part of) relations. A complete tag-set of dependency relations based on CPG can be found in (Bharati et al.,2009) [16].

	Training	Testing
# sentences	14321	1799
# Token-Count	299690	41514
#Chunk-Count	167910	20992

Table 4.2: Hindi TreeBank Statistics

4.2.2.2 Hindi Mono-Lingual Data

We used news corpus of Hindi(distributed as a part of WMT'14[4] translation task) for vector space modeling. The data is about 3 million sentences comprising of about 5.3 million words with unique vocabulary size of 430,379 words.

[3] http://ltrc.iiit.ac.in/treebank_H2014

[4] http://www.statmt.org/wmt14/translation-task.html

4.2.2.3 Tools Used

We used MaltParser(version 1.8)[5] [77] for training the parser. Here we have used a newer version of MaltParser unlike the previous experiments in chapter 2 and chapter 3. We use Word2Vec[6] for vector space modeling.
Word2Vec provides an efficient implementation of continuous bag-of-words and skip-gram architectures for computing vector representations of words. We give more information about the working of vector space modeling in the next section.

4.2.3 Background and Experimental setup

In this section, we will explain the setup required for our experiments.

As usual, in our experiments, we focus on establishing dependency relations between the chunk heads i.e. inter-chunk parsing.

4.2.3.1 Available Features

Table 4.3 describes all the features and their properties annotated for each token, in Hindi treebank. All the features listed may not be applicable to all the words/categories. For example, GNPC is not applicable for conjunctions. In treebank, all the features are manually annotated and thus have good quality. We have already discussed about the available features in treebank in Chapter 2, but here we try to show how these features are helpful in parsing and how costly is it to extract them automatically using tools. Almost all of the features provided contain rich information, useful for parsing. But, at parsing time, it is unrealistic to work with gold features. Instead these features should be derived automatically using the available tools. We divide the features used in parsing into two sets based on how efficiently they can be learned and produced. A feature is *trivial* if the tool used to obtain it can be made easily available and provides good quality. In case of Hindi, the widely used Paradigm Based Analyzer morph analyzer (Bharati et al., 1995) [12] is very costly to build and inefficient in handling OOV words. So we consider features generated by it like G,N,P,C to be *Non-Trivial*. Similarly, we consider POS tag as a trivial feature.

We try out different combinations of features in gold tagged data as well as on the auto-data (In auto-data, features are generated using tools). We combine these strategies along with the different strategies of incorporating word embeddings to see the effect and find out most suitable combination for the auto-data.
Here we show the different combinations of features we tried.

[5] `http://www.maltparser.org/`

[6] `https://code.google.com/p/word2vec/`

Feature	Tool used	Trivial or Non-Trivial	Col
word-form	none	Trivial	2
Lemma	Morph Analyzer	Non-Trivial	3,6
CPOS	Morph Analyzer	Non-Trivial	4,6
POS	POS Tagger	Trivial	5
G,N,P,C	Morph Analyzer	Non-Trivial	6
TAM,Vibhakti	Vibhakti Computation	Trivial	6
Chunk ID, Chunk Type	Chunker	Trivial	6
stype, voice-type	Manual	Non-Trivial	6

Table 4.3: All the features available in Hindi DTB. G,N,P,C refers to gender, number, person and case. TAM: Tense, Aspect, Modality

1. All features available in treebank for gold data & all features derivable from tools for auto-data (all features except stype, voicetype).

2. Only trivial features which includes word-form + POS (part of speech tag)+ TVC (TAM, Vib, ChunkType, ChunkID)

3. Only word-form and POS

Note: To make evaluation feasible for output of auto-data, we have used gold features for chunking to handle the problem of chunk-heads mismatch between gold test-data and output of auto test data. As the accuracy of the chunker[7] we have used is very high, this does not affect much to the system.

[7] `http://ltrc.iiit.ac.in/showfile.php?filename=downloads/shallow_parser.php` we are using shallow parser for automatic features

4.2.3.2 Learning Vector Space model

In recent years, there has been a trend in the NLP research community of learning distributed representations for different natural language units, from morphemes, words and phrases, to sentences and documents. Using distributed representations, these symbolic units are embedded into a low dimensional and continuous space, thus it is often referred to as *embeddings*.

Here, we give a brief idea about the working of Word2Vec tool. The word2vec tool takes a text corpus as input and produces the word vectors as output. It first constructs a vocabulary from the training text data and then learns vector representation of words.

Various approaches have been studied for learning word embeddings from large-scale plain texts. Most commonly used approaches are, Continuous bag-of-words model (CBOW) and Skip Gram NNLM. CBOW being faster than Skip Gram approach, in this study, we consider the Continuous Bag-of-Words (CBOW)[66] model.

Continuous bag-of-words model (CBOW):
The basic principle of the CBOW model is to predict each individual word in a sequence given the bag of its context words within a fixed window size as input, using a log-linear classifier. This model avoids the non-linear transformation in hidden layers, and hence can be trained with high efficiency.

With large window size, grouped words using the resulting word embeddings are more topically similar; whereas with small window size, the grouped words will be more syntactically similar. So we set the window size to 5 for our parsing task.

Figure 4.1 represents a typical CBOW model. The model learns compressed, continuous representations of words. We call the vectors in the matrix between the input and hidden layer, *word vectors*. Each word is associated with a real valued vector in N-dimensional space (usually N = 50 - 1000). These word vectors can be subsequently used as features in many NLP tasks. As word vectors can be trained on huge text datasets, they provide generalization for systems trained with limited amount of supervised data. Word vectors capture many linguistic properties (gender, tense, plurality, even complex semantic concepts)

Skip Gram NNLM:
This model tries to predict the surrounding words using the current word. It is slower than CBOW architecture and better for infrequent words. Figure 4.2 represents a typical Skip Gram NNLM architecture.

Word2Vec training
First of all, we train a vector space model using the raw data mentioned in section 4.2.2.2. We keep the size of word vectors to be 100 and use other default settings. So now we have word-embeddings ready to be used.

We perform K-means clustering over the vectors formed to group similar words into one cluster.We keep models ready with different values of 'K'.

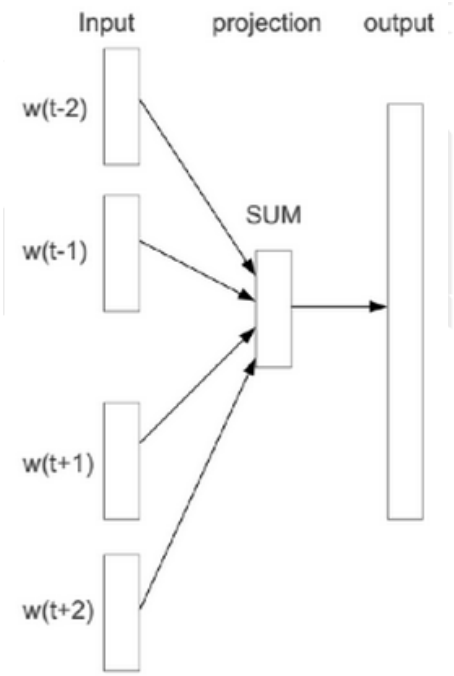


Figure 4.1: CBOW model

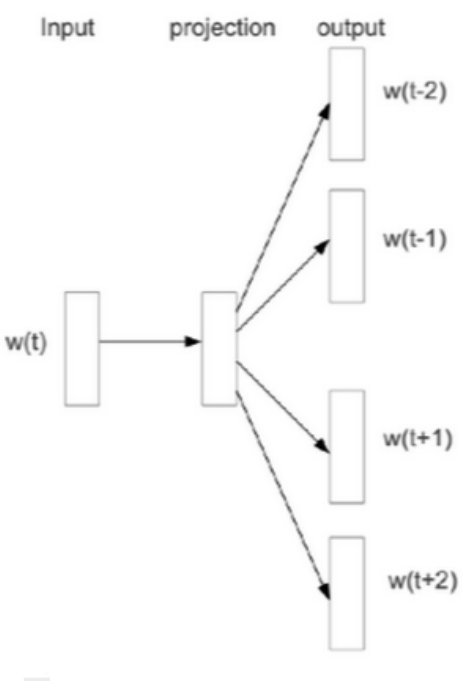


Figure 4.2: Skip Gram NNLM

4.2.3.3 Incorporating Embeddings in Conll Data

Architecture of CoNLL data format is described in Chapter 2. In the FEATS column (in a CoNLL file) of commonly used template of Malt Parser, the features are generally discrete and are separated by pipes ("|"). Whereas the features generated by Word2Vec are Numeric (Continuous). So, we created new columns for each numeric-feature and added them in the template/Conll X data-format file to consider them as numeric features. Similarly in the 'feature model specification file' where, the features to be considered are defined, we added top of stack and current input into consideration for every new dimension/feature from the vector. So, in the present state, the FEATS column is not affected and new columns are added to the feature templates.

4.2.3.4 Different Strategies To Gather More Information In Word Embeddings

Just like different combinations of features as described in section 4.2.3.1, we also try out different strategies to produce different word embeddings. Every strategy used, produces vectors containing different types and amount of information. To understand different strategies, we take a dummy example and apply all the strategies to it.
Let's say "a b c" is a chunk consisting of 3 tokens a, b and c, where 'b' is the chunk head. Let's say our word vectors consist of 3 dimensions.

word vectors for a, b and c:
a: 1, 2, 1
b: 2, 1, 2
c: 0, 3, 3

1. **Chunk-Head Word Vector:** In this strategy we directly add word embeddings for chunk heads as features in the inter-chunk data. So in our dummy example, the word vector features of chunk-head will look like:
   b: 2, 1, 2

2. **Concatenate Window-1:** To capture the context information which we miss-out during inter-chunk parsing, we concatenate vectors of previous, current and next word (from full data) to get a combined vector of size 300. We add this new vector to the chunk head as features. In our dummy example size of word vector would become 9.
   b: 1, 2, 1, 2, 1, 2, 0, 3, 3

3. **Average-out vectors in chunk:** When we are dealing with inter-chunk parsing, we generally focus only on the information provided by the chunk-heads and ignore other information from other words in the chunks. To include this information, we take vectors of all the words present in the chunk and take average of each feature, to get a new vector of the same length. We use this vector as a feature for the chunk-head. Now, the features in chunk head also capture entire information provided by the words in that chunk. In our example,
   b: (1+2+0)/3, (2+1+3)/2, (1+2+3)/3
   b: 1, 2, 3
   We know that concatenation of vectors works better than averaging them, but we can not concatenate the vectors in a chunk, as the number of tokens in a chunk vary from chunk to chunk. If we do concatenation in a chunk, we would get vectors of different dimensions for different chunk-heads, which would not be consistent with the feature template.

4. **Cluster ID:** In this strategy we add Cluster ID of word (or lemma in case of word not found) from the clusters formed as described in section 4.2.3.2 in the FEATS column as an additional feature.This strategy can also be combined with the other three strategies.

4.2.4 Experiments and Results

4.2.4.1 Baseline

We set up two baselines, one for gold-data and another for auto-data. For learning and parsing of baseline systems, we use MaltParser with all the default settings. Here, we do not

use optimized settings as described by [84] in Chapter 3. The settings used in that system is optimized for the features in FEATS column and because of the complex SVM settings, the model takes a lot of time to learn a model and also to parse new sentences. We can not use these settings as those are not optimized for the new features we are adding (word-embeddings), so it would be difficult to compare the results.

In the baseline systems, we use all the available features in the gold and auto data. The baseline for gold data gives LAS of 82.23% & for auto data the baseline LAS comes out to be 76.55%. As expected, the baseline for auto data is much lower (about 6% LAS) than gold data because the features in auto data are obtained using automatic tools and are worse in quality than gold features.

4.2.4.2 Combinations of different features and word embedding strategies

In our first experiment, we try to find out optimal number of clusters to be used while clustering the vectors to get best results. We try out different values of 'K' ranging from 25 to 700 and come to conclusion that K=200 gives the best results when we use cluster ID as a feature in the parsing. So, we fix the value of 'K' to be 200 for further experiments.

- **Gold Data Experiments**
  In experiments with gold data, we first try out different strategies of incorporating word embeddings, and taking all features from FEATS column. The results can be seen in table 4.4. We observe that the strategy of concatenation of context vectors works best for gold data, giving an increment of 0.6% in LAS Score. We can see that all the strategies of word embeddings produced some increment in accuracy. In further experiments on gold data, we show results for the technique of concatenation only as this strategy performed best in all the experiments. From these experiments we can say that, word-embeddings capture some *additional real world aspects* of lexical items, which are quite distinct and unlikely to be deduced from the morpho-syntactic information like morph, POS-tag and chunk. This complementing semantic information is helping improve parse quality. We also performed combination of both the strategies (cluster ID along with concatenation) and did not find it helpful. Then we apply all the strategies of word embeddings to the data where we take only trivial features. The baseline accuracy (where no word vectors are added) is decreased in this case. This shows that not only trivial but all the features (if provided correctly) are important for parsing. This time also, we observe similar pattern as seen in above experiments. Once again the concatenation performs better than chunk average and it in turn performs better than normal adding of word vectors. Here we observe that combination of concatenation with Cluster ID improves the results further. In Table 4.5, we show the improvements for the strategy of concatenation. We observe that cluster ID in itself does not improve results much but when used with other strategies, it helps

Experiment (Features Used)	**LAS**	**UAS**	**LS**
Baseline / All feats	82.23	90.33	84.54
All feats+vectors(1)	82.51	90.43	84.84
All feats+context(2)	**82.83**	**90.73**	**85.18**
All feats+chunk avg.(3)	82.61	90.46	84.95
All feats+cluster ID(4)	82.57	90.58	84.84
All feats+context(2) +clusterID(4)	82.76	90.66	85.14

Table 4.4: Results on Gold Data, All features, different Word Embedding strategies

Experiment (Features Used)	**LAS**	**UAS**	**LS**
trivial features only	81.95	90.19	84.18
trivial+vectors(1)	82.39	90.31	84.62
trivial+context(2)	82.73	90.76	84.95
trivial+context(2) +clusterID(4)	**82.74**	**90.76**	**85.00**

Table 4.5: Trivial features with different word embedding strategies

improve the results further. So, to conclude, we get a maximum LAS score of 82.83% using the context concatenation strategy. And we have shown that using chunk ID as a feature may sometimes help improve the scores.

- **Auto Data Experiments**
  Our main focus is to improve the results of auto-data as real time systems only have access to automatically extracted features. We do the same experiments with auto-data to see the results. In our first set of experiments, we use all the features in FEATS column and see effect of different embedding strategies.

  Here, we see maximum improvement of 1.52% over the baseline. In case of auto-data we observe a little change in pattern. We see that here, the averaging of all the chunk vectors performs better than the concatenation of context vectors, unlike the gold-data. One of the reasons behind this might be that, "vibhakti" feature captures somewhat information

Experiment (Features Used)	**LAS**	**UAS**	**LS**
Baseline	76.55	87.45	79.02
All + vectors(1)	77.09	87.62	79.62
All + context(2)	77.34	87.67	80.00
All+chunk avg.(3)	**78.07**	**87.98**	**80.63**
All + Chunk ID(4)	77.05	87.63	79.57

Table 4.6: Results on Auto Data, All features, different Word Embedding strategies

in the chunk. Gold data being manually annotated, this information is accurate in it but in auto data, it may not be accurate. So, to gather this information more accurately, need of help from other words in chunk arises and hence helps in improving results for auto-data. We observed that averaging of chunk performs better in all further set of experiments. Similar to the gold-data experiments, we will now show results only for this strategy in the further experiments.

Experiment (Features Used)	**LAS**	**UAS**	**LS**
trivial features only	77.64	88.26	79.87
trivial + vectors(1)	78.04	88.34	80.36
trivial + chunk avg.(3)	79.24	88.79	81.66
trivial+chunk avg.(3) + cluster ID(4)	**79.30**	**88.83**	**81.67**

Table 4.7: Trivial features with different word embeddings

The second set of experiments (table 4.7) is performed similar to gold-data, taking only trivial features. Unlike gold data, we find that the results improve over that of the experiments with all features. This shows that the non-trivial features, we skipped in these experiments are not accurate and hence produce sub-optimal results.

The maximum improvement of 2.52% is observed over baseline in the combination of two strategies Cluster ID and Chunk avg.

In third set of experiments on the auto-data we try to use only the most basic feature "POS", which is very easy to obtain for any language.(table 4.8)

Experiment (Features Used)	LAS	UAS	LS
POS	67.99	82.16	69.75
POS + vectors(1)	69.14	82.76	70.84
POS + chunk avg.(3)	**79.16**	**88.45**	**81.77**
POS+chunk avg.(3) + cluster ID(4)	79.10	88.43	80.97
POS + cluster ID(4)	47.18	57.73	55.22

Table 4.8: Only POS feature in feats column

Here we can see that using just one additional feature (POS) along with word embeddings provides very good results.

On the conrtary, Cluster ID as a feature had a negative effect on these experiments which is based on the fact that there is no feature which now captures the information about the context which was earlier captured by *V*ibhakti feature.

4.2.4.3 Cross Domain Parsing

Another problem we try to tackle is of domain difference between training and testing data. When we have parser trained on data of certain domain (in our case news articles and heritage) and we want to use it for parsing data from another domain, we need to bridge the gap between diverse vocabulary of different domains, which makes parsing difficult. For this task, we group words with similar semantics into clusters.(as mentioned in section 4.2.3.2). These clusters help in handling new words by treating them similar to other words from the same cluster.

We experiment different approaches on the data of four domains: box-office, cricket, gadget and recipe. The data statistics are shown in table 4.9. The data we have, for different domains, is manually annotated for dependency relations, but the features provided in the FEATS column are automatically obtained using tools. So, we also tried out using different combination of features, as we did in previous experiments.

Table 4.10 shows LAS for different experiments we tried.

As expected, we can see that removing the non trivial and less accurate features, improves the results of parsing by large amount as compared to the baseline. For each domain we can see that using cluster ID as a feature improves the quality significantly.

One strange thing we observed is, using word embeddings as features has adverse effects on parsing. One reason behind this might be, by using **100** dense features while training, parser

Counts Domain	sentences	chunks	word vector not found
Box Office	509	3986	20
Cricket	508	4487	36
Gadget	527	4340	141
Recipe	544	4082	47

Table 4.9: Domain Data statistics

might have learned more accurately for the domain of training data and got biased toward the specific domain. We can see that the same pattern is followed by each domain.

Domain features	Box -office	Cricket	Gadget	Recipe
all	69.84	65.73	63.00	60.31
all + vect	60.59	51.74	53.25	49.56
all+cluster	**70.92**	**67.13**	**64.40**	**61.02**
triv	**77.32**	**72.17**	**71.36**	**66.88**
triv + vect	67.89	58.58	62.67	52.72
triv+cluster	**77.47**	**74.11**	**72.88**	**68.57**

Table 4.10: Domain data LAS (all: all features in FEATS, triv: only trivial features).

4.2.5 Observations and Discussion

From the above experiments on gold-data and auto data, we observe that using trivial features (which can be made easily available for any language) along with the word embeddings, the gap between auto-data accuracy and gold data accuracy is reduced to almost half compared to original. We also observed that using merely one feature "POS", which is very trivial to obtain, we can reach close to the maximum auto-data accuracy. Let us look at an example.
(जब)(when) (1911 में)(1911 in) (क्वीन मैरी)(queen Mary) (यहां)(here) (आई थीं ,)(came aux ,) (NULL)(then) (ब्रिटिश शासकों ने)(British Rulers-Erg) (यहां के)(here's) (पानी के)(water's) (हौज के ऊपर)(reservoir above) (एक छत)(one roof) (बनवाई थी)(built aux) (।)(.)

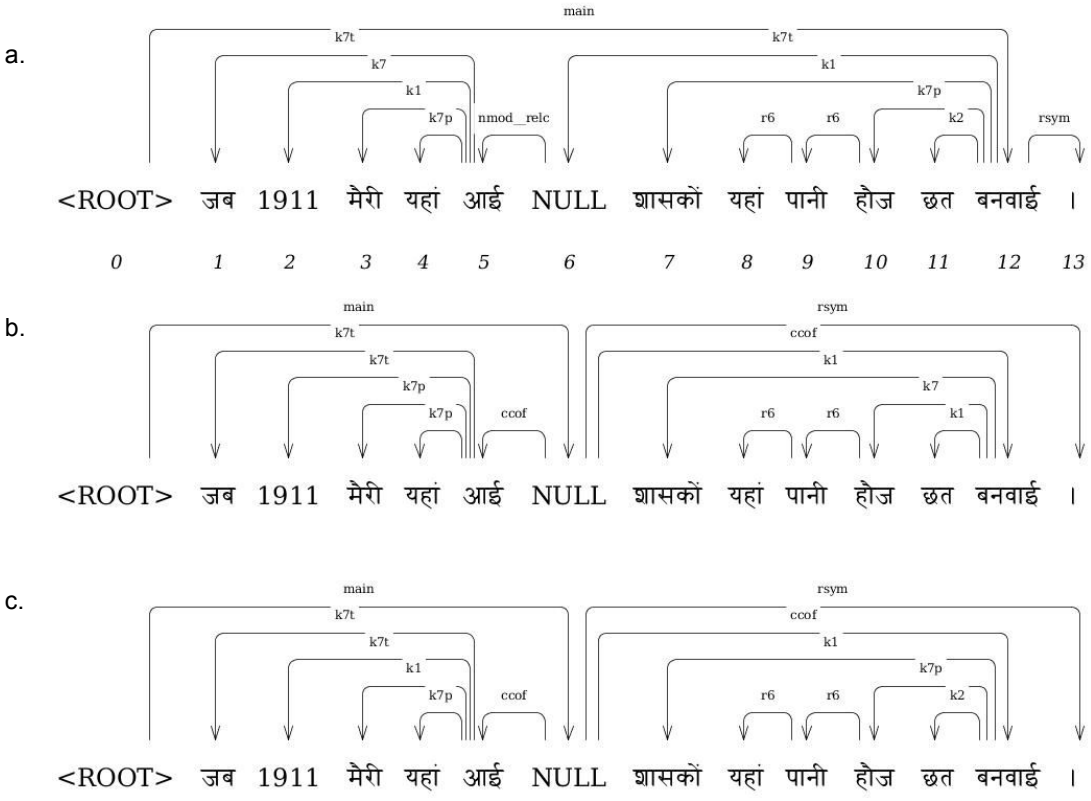


Figure 4.3: Gold tree(a), Auto tree(b), Auto+chunk Avg.+clusterID tree(c)

Translation: In 1911, when Queen Mary came here, British rulers had built here a roof above the water reservoir.
Figure 4.3(a) shows the gold tree, 4.3(b) the trees produced by auto trivial features and 4.3(c) the tree produced by auto trivial + chunk avg + clusterID (the trees are interChunk, i.e., between chunk-heads). Improvement can be clearly seen in the tree by using the chunk avg. technique (strategy 3 under section 4.2.3.4). 'Mary' should be marked as 'k1' (doer) of 'came' but it is wrongly marked as 'k7p' (place of action) as the information provided by trivial features is not sufficient to identify that the word 'Mary' is name of a person (the POS tag 'NNP' (proper noun) is assigned). The problem is resolved when we use the technique of chunk avg. and clusterID.

We found that most of the words in the cluster containing word 'Mary' are British names as the treebank data is from news and heritage domain. Data from Indian heritage domain contains description of British people at various places. The information that Mary is a person and not a place is captured by the clusterID and the word-vector. In similar way the case of 'k7p' for 'reservoir' and 'k2' for 'roof' is handled.

From above observation we can say that some semantic properties of words are getting captured in the word vectors which also include some morphological characteristics.

4.2.6 Conclusions

Experiments on gold data have shown that use of word embeddings and cluster ID as features, is helpful for the resource rich languages for which accurate tools for obtaining features are available.

The main outcome of our experiments is that, we can achieve good results for parsing of resource-poor languages (for which tools and treebanks are not available) by using simple features or even just the POS tags and having a large mono-lingual corpus in hand.

The strategy of clustering is most useful in the case of cross domain parsing. It helps in reducing difficulty faced in parsing because of vocabulary mismatch across different domains.

4.2.7 Future Work

In our experiments, we have neglected very low frequency words to efficiently learn a vector space model, which tends to loss of important vocabulary words. In Morphological rich languages like Dravidian Languages, where same root-words inflect to have many word forms with different suffixes and prefixes, it becomes extremely difficult to model all words efficiently while learning word-embeddings for them. Therefore it might be a good idea to treat suffixes as separate words and learn embedding for them too. According to various works on compositional semantics (Krishnamurthy and Mitchell, 2013; Kalchbrenner et al., 2014; dos Santos and Gatti, 2014) [56, 49, 37], it has been shown that we can efficiently learn embedding of group of words by knowing embedding of individual words. In this case, a word can have multiple suffixes and prefixes along with one root-word. So this word will be formed with compositional semantics of all those word-embedding.

Chapter *5*

Conclusions and Future Work

Most of the Indian languages are resource poor in terms of tools and the data required to create high quality tools. Indian languages are MOR-FWo languages. Dependency Parsing is a challenging task for such languages. We have presented different approaches for handling the problem of resource availability. First we studied the technique, "Cross Lingual Dependency Parsing" which makes use of the already built treebanks, which is the most important resource required for data driven parsing, of other languages to create treebanks in the desired language. This saves a lot of human effort required for creating the resource, hence making the process cost effective.

To make the parsing quality better at real time and to make the process cost effective, we exploit cheap mono-lingual raw corpus and try to remove the costly linguistic features by replacing them with word-embeddings learned on this raw data.

To further improve the quality of parsing of morphologically rich Indian languages at real time, we are planning to make use of sandhi-splitting as already explained in the future-work section of the Chapter 4.

We have used the default settings of Malt Parser in the experiments using word-embeddings, but as shown in the Chapter-3, we can find the optimal settings, parameters, options of the Malt Parser to improve the Parse quality significantly. The strategies used in Chapter 2 like, pseudo-projectivization can also be applied to the parser developed using word embeddings. This would result in a state of the art parser using word-embeddings as features. The Malt Optimizer tool [6] is nowadays used to automatically find out the best parameter settings, feature template and other options in Malt Parser, but we found that the tool works only with the standard conll-x format. In our experiments with word embeddings, we modified our data format in order to include the additional numeric features, so to find out optimal settings, we need to find another way. As the new feature set includes both numeric and discrete features, the SVM settings also, matter a lot.

Most of the statistical, rule based or constraint based parsers require shallow parse features, thus the accuracies of modules such as chunker, pos-tagger, morph-analyzer etc. affect parsing

accuracy. To reduce this problem, we are working on a discrete optimization model based on a linear programming formulation as an alternative to the cascade of classifiers. The cascade of classifiers introduce the problem of error propagation. Using this approach, will not only reduce this error but also make the modules work in parallel hence, reducing the time complexity for parsing, again making it cost-effective in terms of time taken to parse the input.

Related Publications

1. **Aniruddha Tammewar**, Karan Singla, Bhasha Agrawal, Riyaz Bhat and Dipti Misra Sharma. 2015. Can distributed word embeddings be an alternative to costly linguistic features: A study on parsing hindi. In *Proceedings of the 6th Workshop on Statistical Parsing of Morphologically Rich Languages*, (SPMRL 2015) (pp. 21-30), Bilbao, Spain.

2. Sambhav Jain, Naman Jain, **Aniruddha Tammewar**, Riyaz Bhat and Dipti Misra Sharma. 2013. *Exploring Semantic Information in Hindi WordNet for Hindi Dependency Parsing. In The sixth international joint conference on natural language processing*, (IJCNLP 2013), Nagoya, Japan.

3. Karan Singla, **Aniruddha Tammewar**, Naman Jain and Sambhav Jain. 2012. Two-stage Approach for Hindi Dependency Parsing Using MaltParser. In *24th International Conference on Computational Linguistics. Vol. 12041. No. 268,093.*, (COLING 2012), Mumbai, India.

Bibliography

[1] E. Agirre, K. Bengoetxea, K. Gojenola, and J. Nivre. Improving dependency parsing with semantic classes. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 699–703. Association for Computational Linguistics, 2011.

[2] E. Agirre, B. C. Donostia, T. Baldwin, and D. Martinez. Improving parsing and pp attachment performance with sense information. *ACL-08: HLT*, page 317, 2008.

[3] B. R. Ambati, P. Gade, G. Chaitanya, S. Husain, and I. LTRC. Effect of minimal semantics on dependency parsing. In *RANLP*, pages 1–5, 2009.

[4] B. R. Ambati, S. Husain, S. Jain, D. M. Sharma, and R. Sangal. Two methods to incorporate local morphosyntactic features in hindi dependency parsing. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 22–30. Association for Computational Linguistics, 2010.

[5] Authors. The frobnicatable foo filter. 2006. ECCV06 submission ID 324. Supplied as additional material `eccv06.pdf`.

[6] M. Ballesteros and J. Nivre. Maltoptimizer: an optimization tool for maltparser. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 58–62. Association for Computational Linguistics, 2012.

[7] S. Banerjee and T. Pedersen. Extended gloss overlaps as a measure of semantic relatedness. In *IJCAI*, volume 3, pages 805–810, 2003.

[8] R. Begum, S. Husain, A. Dhwaj, D. M. Sharma, L. Bai, and R. Sangal. Dependency annotation scheme for indian languages. In *IJCNLP*, pages 721–726. Citeseer, 2008.

[9] R. Begum, K. Jindal, A. Jain, S. Husain, and D. M. Sharma. Identification of conjunct verbs in hindi and its effect on parsing accuracy. In *Computational Linguistics and Intelligent Text Processing*, pages 29–40. Springer, 2011.

[10] T. Berg-Kirkpatrick and D. Klein. Phylogenetic grammar induction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1288–1297. Association for Computational Linguistics, 2010.

[11] A. Bharati. Anncorra: Annotating corpora guidelines for pos and chunk annotation for indian languages. 2006.

[12] A. Bharati, V. Chaitanya, R. Sangal, and K. Ramakrishnamacharyulu. *Natural language processing: a Paninian perspective*. Prentice-Hall of India New Delhi, 1995.

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

[14] A. Bharati, P. Mannem, and D. M. Sharma. Hindi parsing shared task. In *Proceedings of Coling Workshop on Machine Translation and Parsing in Indian Languages, Kharagpur, India*, 2012.

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

[16] A. Bharati, D. M. Sharma, S. Husain, L. Bai, R. Begam, and R. Sangal. Anncorra: Treebanks for indian languages, guidelines for annotating hindi treebank, 2009.

[17] R. A. Bhat and D. M. Sharma. A dependency treebank of urdu and its evaluation. In *Proceedings of the Sixth Linguistic Annotation Workshop*, pages 157–165. Association for Computational Linguistics, 2012.

[18] R. Bhatt, B. Narasimhan, M. Palmer, O. Rambow, D. M. Sharma, and F. Xia. A multi-representational and multi-layered treebank for hindi/urdu. In *Proceedings of the Third Linguistic Annotation Workshop*, pages 186–189. Association for Computational Linguistics, 2009.

[19] G. Cavallanti, N. Cesa-Bianchi, and C. Gentile. Linear algorithms for online multitask classification. *The Journal of Machine Learning Research*, 11:2901–2934, 2010.

[20] D. Chakrabarti, V. Sarma, and P. Bhattacharyya. Complex predicates in indian language wordnets. *Lexical Resources and Evaluation Journal*, 40(3-4), 2007.

[21] C.-C. Chang and C.-J. Lin. Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[22] D. Chen and C. D. Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, 2014.

[23] Y.-J. Chu and T.-H. Liu. On shortest arborescence of a directed graph. *Scientia Sinica*, 14(10):1396, 1965.

[24] S. B. Cohen and N. A. Smith. Shared logistic normal distributions for soft parameter tying in unsupervised grammar induction. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 74–82. Association for Computational Linguistics, 2009.

[25] M. Collins, L. Ramshaw, J. Hajič, and C. Tillmann. A statistical parser for czech. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 505–512. Association for Computational Linguistics, 1999.

[26] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

[27] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[28] M. A. Covington. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th annual ACM southeast conference*, pages 95–102. Citeseer, 2001.

[29] K. Crammer and Y. Singer. Ultraconservative online algorithms for multiclass problems. *The Journal of Machine Learning Research*, 3:951–991, 2003.

[30] D. Damljanovic, M. Agatonovic, and H. Cunningham. Natural language interfaces to ontologies: Combining syntactic analysis and ontology-based lookup through the user interaction. In *The semantic web: Research and applications*, pages 106–120. Springer, 2010.

[31] M.-C. De Marneffe, T. Dozat, N. Silveira, K. Haverinen, F. Ginter, J. Nivre, and C. D. Manning. Universal stanford dependencies: A cross-linguistic typology. In *Proceedings of LREC*, pages 4585–4592, 2014.

[32] M.-C. De Marneffe, B. MacCartney, C. D. Manning, et al. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454, 2006.

[33] M.-C. De Marneffe and C. D. Manning. The stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8. Association for Computational Linguistics, 2008.

[34] R. Debusmann, D. Duchier, and G.-J. M. Kruijff. Extensible dependency grammar: A new methodology. In *Proceedings of the COLING 2004 Workshop on Recent Advances in Dependency Grammar*, pages 70–76, 2004.

[35] J. Devlin, R. Zbib, Z. Huang, T. Lamar, R. Schwartz, and J. Makhoul. Fast and robust neural network joint models for statistical machine translation. In *52nd Annual Meeting of the Association for Computational Linguistics, Baltimore, MD, USA, June*, 2014.

[36] Z. Dong and Q. Dong. Hownet chinese-english conceptual database. Technical report, Technical Report Online Software Database, Released at ACL. http://www. keenage. com, 2000.

[37] C. N. dos Santos and M. Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of the 25th International Conference on Computational Linguistics (COLING), Dublin, Ireland*, 2014.

[38] G. Durrett, A. Pauls, and D. Klein. Syntactic transfer using a bilingual lexicon. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1–11. Association for Computational Linguistics, 2012.

[39] J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71(4):233–240, 1967.

[40] J. M. Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 340–345. Association for Computational Linguistics, 1996.

[41] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

[42] S. Fujita, F. Bond, S. Oepen, and T. Tanaka. Exploiting semantic information for hpsg parse selection. *Research on language and computation*, 8(1):1–22, 2010.

[43] L. Georgiadis. Arborescence optimization problems solvable by edmonds' algorithm. *Theoretical Computer Science*, 301(1):427–437, 2003.

[44] C. Gini. Variabilità e mutabilità. contributi allo studio dele relazioni e delle distribuzioni statistiche. *Studi Economico-Giuridici della Università di Cagliari*, 1912.

[45] R. Hudson. Word grammar: Blackwell oxford. 1984.

[46] S. Husain and B. Agrawal. Analyzing parser errors to improve parsing accuracy and to inform tree banking decisions. *Linguistic Issues in Language Technology*, 7(1), 2012.

[47] R. Hwa, P. Resnik, A. Weinberg, C. Cabezas, and O. Kolak. Bootstrapping parsers via syntactic projection across parallel texts. *Natural language engineering*, 11(03):311–325, 2005.

[48] S. Jain, N. Jain, A. Tammewar, R. A. Bhat, and D. M. Sharma. Exploring semantic information in hindi wordnet for hindi dependency parsing. 2013.

[49] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[50] F. Karlsson, A. Voutilainen, J. Heikkilae, and A. Anttila. *Constraint Grammar: a language-independent system for parsing unrestricted text*, volume 4. Walter de Gruyter, 1995.

[51] P. Kingsbury, M. Palmer, and M. Marcus. Adding semantic annotation to the penn treebank. In *Proceedings of the Human Language Technology Conference*, pages 252–256. Citeseer, 2002.

[52] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics, 2007.

[53] S. Kolachina, P. Kolachina, M. Agarwal, and S. Husain. Experiments with malt parser for parsing indian languages. *Proc of ICON-2010 tools contest on Indian language dependency parsing. Kharagpur, India*, 2010.

[54] P. Kosaraju, S. Husain, B. R. Ambati, D. M. Sharma, and R. Sangal. Intra-chunk dependency annotation: expanding hindi inter-chunk annotated treebank. In *Proceedings of the Sixth Linguistic Annotation Workshop*, pages 49–56. Association for Computational Linguistics, 2012.

[55] P. Kosaraju, S. R. Kesidi, V. B. R. Ainavolu, and P. Kukkadapu. Experiments on indian language dependency parsing. *Proceedings of the ICON10 NLP Tools Contest: Indian Language Dependency Parsing*, 2010.

[56] J. Krishnamurthy and T. M. Mitchell. Vector space semantic parsing: A framework for compositional vector space models. *ACL 2013*, page 1, 2013.

[57] S. Kübler, R. McDonald, and J. Nivre. Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127, 2009.

[58] T. Kudo and Y. Matsumoto. Japanese dependency analysis using cascaded chunking. In *proceedings of the 6th conference on Natural language learning-Volume 20*, pages 1–7. Association for Computational Linguistics, 2002.

[59] M. Kuhlmann and M. Möhl. Mildly context-sensitive dependency languages. In *ANNUAL MEETING-ASSOCIATION FOR COMPUTATIONAL LINGUISTICS*, volume 45, page 160, 2007.

[60] A. MacKinlay, R. Dridan, D. McCarthy, and T. Baldwin. The effects of semantic annotations on precision parse ranking. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*, pages 228–236. Association for Computational Linguistics, 2012.

[61] H. Maruyama. Structural disambiguation with constraint propagation. In *Proceedings of the 28th annual meeting on Association for Computational Linguistics*, pages 31–38. Association for Computational Linguistics, 1990.

[62] R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics, 2005.

[63] A. Meena and T. Prabhakar. *Sentence level sentiment analysis in the presence of conjuncts using linguistic analysis*. Springer, 2007.

[64] J.-j. Mei and Y. Gao. Tongyi cilin (a chinese thesaurus). *China: Shanghai Lexicographical Publishing House*, 1996.

[65] I. Mel'c. Dependency syntax: theory and practice. *State University of New York Press, Albany, New York, USA*, 1988.

[66] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[67] G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[68] K. P. Mohanan. Grammatical relations and clause structure in malayalam. *The mental representation of grammatical relations*, 504:589, 1982.

[69] T. Mohanan. *Argument structure in Hindi*. Center for the Study of Language (CSLI), 1994.

[70] T. Mohanan. Multidimensionality of representation: Nv complex predicates in hindi. *Complex predicates*, pages 431–471, 1997.

[71] S. Montemagni, F. Barsotti, M. Battista, N. Calzolari, O. Corazzari, A. Lenci, A. Zampolli, F. Fanciulli, M. Massetani, R. Raffaelli, et al. Building the italian syntactic-semantic treebank. *Treebanks*, pages 189–210, 2003.

[72] D. Narayan, D. Chakrabarti, P. Pande, and P. Bhattacharyya. An experience in building the indo wordnet-a wordnet for hindi. In *First International Conference on Global WordNet, Mysore, India*, 2002.

[73] T. Naseem, H. Chen, R. Barzilay, and M. Johnson. Using universal linguistic knowledge to guide grammar induction. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1234–1244. Association for Computational Linguistics, 2010.

[74] J. Nilsson, S. Riedel, and D. Yuret. The conll 2007 shared task on dependency parsing. In *Proceedings of the CoNLL shared task session of EMNLP-CoNLL*, pages 915–932. sn, 2007.

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

[76] J. Nivre. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553, 2008.

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

[78] J. Nivre and J. Nilsson. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 99–106. Association for Computational Linguistics, 2005.

[79] L. Øvrelid and J. Nivre. When word order and part-of-speech tags are not enough–swedish dependency parsing with rich linguistic features. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP)*, pages 447–451, 2007.

[80] K. Owczarzak. Depeval (summ): dependency-based evaluation for automatic summaries. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 190–198. Association for Computational Linguistics, 2009.

[81] M. Palmer, R. Bhatt, B. Narasimhan, O. Rambow, D. M. Sharma, and F. Xia. Hindi syntax: Annotating dependency, lexical predicate-argument structure, and phrase structure. In *The 7th International Conference on Natural Language Processing*, pages 14–17, 2009.

[82] S. Petrov, D. Das, and R. McDonald. A universal part-of-speech tagset. *arXiv preprint arXiv:1104.2086*, 2011.

[83] S. Sheiber. Evidence against the context-freeness of natural languages. *Linguistics and Philosophy*, 8:333–343, 1985.

[84] K. Singla, A. Tammewar, N. Jain, and S. Jain. Two-stage approach for hindi dependency parsing using maltparser. 2012.

[85] B. Snyder, T. Naseem, J. Eisenstein, and R. Barzilay. Adding more languages improves unsupervised multilingual part-of-speech tagging: A bayesian non-parametric approach. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 83–91. Association for Computational Linguistics, 2009.

[86] R. Socher, J. Bauer, C. D. Manning, and A. Y. Ng. Parsing with compositional vector grammars. In *In Proceedings of the ACL conference*. Citeseer, 2013.

[87] A. Søgaard and J. Wulff. An empirical etudy of non-lexical extensions to delexicalized transfer. In *COLING (Posters)*, pages 1181–1190, 2012.

[88] O. Täckström, D. Das, S. Petrov, R. McDonald, and J. Nivre. Token and type constraints for cross-lingual part-of-speech tagging. *Transactions of the Association for Computational Linguistics*, 1:1–12, 2013.

[89] O. Täckström, R. McDonald, and J. Nivre. Target language adaptation of discriminative transfer parsers. 2013.

[90] O. Täckström, R. McDonald, and J. Uszkoreit. Cross-lingual word clusters for direct transfer of linguistic structure. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 477–487. Association for Computational Linguistics, 2012.

[91] P. Tapanainen and T. Järvinen. A non-projective dependency parser. In *Proceedings of the fifth conference on Applied natural language processing*, pages 64–71. Association for Computational Linguistics, 1997.

[92] J. Tiedemann and J. Nivre. Treebank translation for cross-lingual parser induction. *CoNLL-2014*, page 130, 2014.

[93] R. Tsarfaty, D. Seddah, S. Kübler, and J. Nivre. Parsing morphologically rich languages: Introduction to the special issue. *Computational Linguistics*, 39(1):15–22, 2013.

[94] M. Wang, N. A. Smith, and T. Mitamura. What is the jeopardy model? a quasi-synchronous grammar for qa. In *EMNLP-CoNLL*, volume 7, pages 22–32, 2007.

[95] S. Wann, M. Dras, R. Dale, and C. Paris. Improving grammaticality in statistical sentence generation: Introducing a dependency spanning tree algorithm with an argument satisfaction

model. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 852–860. Association for Computational Linguistics, 2009.

[96] D. Xiong, S. Li, Q. Liu, S. Lin, and Y. Qian. Parsing the penn chinese treebank with semantic knowledge. In *Natural Language Processing–IJCNLP 2005*, pages 70–81. Springer, 2005.

[97] P. Xu, J. Kang, M. Ringgaard, and F. Och. Using a dependency parser to improve smt for subject-object-verb languages. In *Proceedings of human language technologies: The 2009 annual conference of the North American chapter of the association for computational linguistics*, pages 245–253. Association for Computational Linguistics, 2009.

[98] H. Yamada and Y. Matsumoto. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, volume 3, pages 195–206, 2003.

[99] D. Yarowsky, G. Ngai, and R. Wicentowski. Inducing multilingual text analysis tools via robust projection across aligned corpora. In *Proceedings of the first international conference on Human language technology research*, pages 1–8. Association for Computational Linguistics, 2001.

[100] D. Zeman and P. Resnik. Cross-language parser adaptation between related languages. In *IJCNLP*, pages 35–42, 2008.