Advancements in Dependency Parsing for Indian Languages

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

MS in Computational Linguistics
by Research

by

Juhi Tandon
201225032
juhi.tandon@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
July 2018

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

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

Date

Adviser: Prof. Dipti Misra Sharma

To My Family

Acknowledgements

I would like to extend my sincere gratitude to my adviser Professor Dipti Misra Sharma. She has always encouraged me to strive to be better. Her advice, that has always stressed to strengthen the fundamemtals and to be guided by linguistic intuition in all the problem solving, has helped me immensely. I draw great inspiration from her not only for her academic excellence but also for making a mark as a strong, successful, kind hearted woman we all look upto.

Thank you to all my instructors Dr. Manish Shrivastava, Dr. Radhika Mamidi and Dr. Soma Paul for the highly interactive classes and laying strong fundamentals for research in Computational Linguistics and Natural Language Processing.

Riyaz Sir thank you for mentoring me and giving me great insights about life. I would like to thank all my lab mates Irshad, Prathyusha Jwalapuram, Nishant, Darshan and others for making LTRC a fun environment to work in. There always has been dynamics of collaboration, of mutual learning and of encouraging each other. Senior members of the lab like Pruthwik, Himanshu, Aniruddha, Naman, Karan deserve a special mention for having patience and resolving my doubts.

Aditya, Raveesh, Snigdha and Agni thank you for always supporting me, guiding me, sticking through the rough times. Thank you for all those light hearted enjoyable moments and meaningful discussions. I would like to thank my younger brother Rishi for stepping up like the older one sometimes. You my brother, are my best critic, thank you for your constructive feedback.

Finally, I cannot thank my parents enough for making innumerable sacrifices for giving me little moments of joy, for giving me a brilliant education and making me able to tread on the path to my dreams. Taking difficult life choices have been easier with your support. I love you.

Abstract

Indian languages are morphologically rich and free word order languages. There are many other distinguished characteristics possessed by them. Keeping these in mind Computational Paninian Grammar formalism was chosen to represent the syntacto-semantic relations between different words in a sentence and establish their relationship with the verb, for these languages. The syntactico-semantic dependency relations and their labels defined in the CPG formalism are very fine grained to account for the rich grammatical functions. The number of distinct dependency labels are 82 as per the scheme (both interchunk and intrachunk). It has been observed that the more semantically oriented annotation schemes make labeled parsing more difficult than the schemes based on more surface-oriented grammatical functions. These relations can be organised in the form of a hierarchical structure with each level representing a degree of granularity which can be underspecified. We aim to explore whether reducing granularity of these labels can help bridge the gap between between Labeled Score and Unlabeled Attachment Score. Universal Dependencies (UD) on the other hand have a coarser scheme of grammatical representation. We extend UD to Indian languages through conversion of Paninian Dependencies to UD for the Hindi Dependency Treebank (HDTB) and observe the effects of a relatively sparse taxonomy. We discuss the differences in annotation in both the schemes, present parsing experiments for both the formalisms and empirically evaluate their weaknesses and strengths for Hindi. We produce an automatically converted Hindi Treebank conforming to the international standard UD scheme, making it useful as a resource for multilingual language technology.

Indian language treebanking was undertaken to create resources for facilitating data driven syntactic analysis. 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. Rich syntactic features are very important for building state-of-the-art syntactic analysers but they require lot of feature engineering expertise. These indicative features pose the problem of data sparsity, incompleteness and expensive extraction. Building expensive tools to automatically generate these features is an expensive task. Keeping this in mind our work proposes to apply non linear neural network for parsing five resource poor Indian Languages belonging to two major language families - Indo-Aryan and Dravidian. Bengali and Marathi are Indo-Aryan languages whereas Kannada, Telugu and

Malayalam belong to the Dravidian family. The non linear architecture elegantly addresses all the problems mentioned above.

While little work has been done previously on Bengali and Telugu linear transition-based parsing, we present one of the first parsers for Marathi, Kannada and Malayalam. All the Indian languages are free word order and range from being moderate to very rich in morphology. Therefore in this work we propose the usage of linguistically motivated morphological features (suffix and postposition) in the non linear framework, to capture the intricacies of both the language families. We also capture chunk and gender, number, person information elegantly in this model. We put forward ways to represent these features cost effectively using monolingual distributed embeddings. Instead of relying on expensive morphological analyzers to extract the information, these embeddings are used effectively to increase parsing accuracies for resource poor languages. Our experiments provide a comparison between the two language families on the importance of varying morphological features. Part of speech taggers and chunkers for all languages are also built in the process.

Contents

List of Figures

List of Tables

Chapter *1*

Introduction

Natural Language Processing is a methodology that aims to deliver to computers the human like ability to process what another human said and comprehend what he / she meant. An understanding of the place of language in human cognition is our best bet for understanding the difference between human and nonhuman minds. While we are still on slippery grounds in learning how humans acquire language. To be able to transfer that capability to a machine is a complex task. Language being a combinatorial system of communication makes the task even more difficult. 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). For more than half a century, efforts have been taken to build systems where machines can complement human capabilities. To converse with humans, a program must understand syntax (grammar), semantics (word meaning), morphology (tense) and pragmatics (conversation).

Natural language can manifest itself as forms of speech, text or gesture. Computationally speaking, processing of all these forms of communication are quite different from one another. In our work as described in this dissertation, we focus on natural language text which is one of the most important and widely used component of any natural language. Processing of text is a complex task and can be broken down into smaller sub tasks. These sub tasks can be organized in the form of a hierarchy as shown in Figure 1.1. i.e. starting from word level analysis at the bottom to discourse analysis at the top most level. The complexity of tasks increases as move up the hierarchy.

Significant work has been done to develop tokenizers, part of speech taggers and morph analyzers for widely used languages like English, Spanish etc. These components find real world applications and meet industry standards. A lot of focus in the last decade has been given to syntactic parsers as they lie at the heart of Natural Language Processing. Now is a transitional time where parsers are being used in downstream applications like machine translation, question

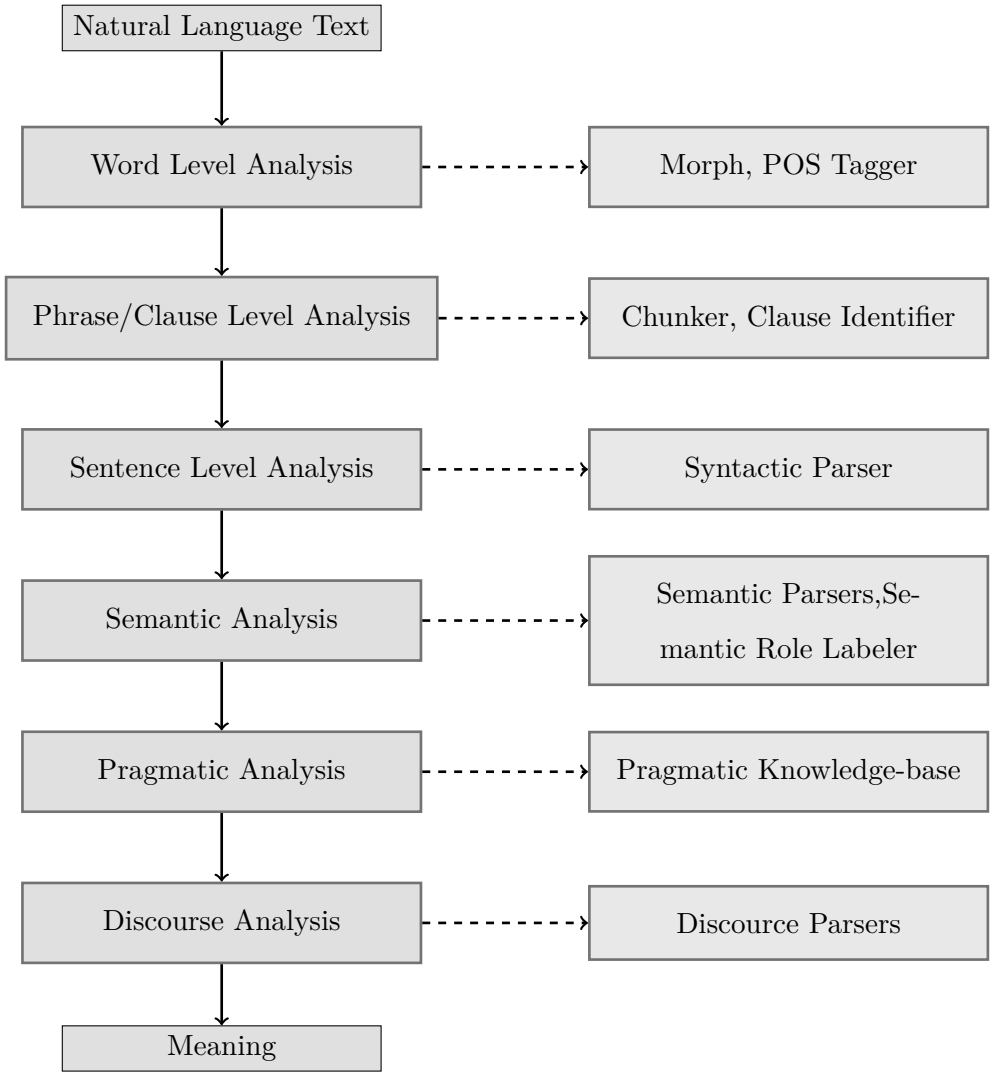


Figure 1.1: Overview of Natural Language Analysis of Text

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

answering, sentiment analysis, natural language generation. Thus the aim of the research community is to develop high quality systems that can establish relationship between words of a sentence. Although sincere efforts have been done in the past to extend the advances of research in parsingto underrepresented languages, majority of Indian languages still lag behind. Hindi (the most spoken Indian language with about 544 million speakers [1]) has seen development of state-of-the art parsers. The creation of a multi layered and multi representational treebank for

[1] https://en.wikipedia.org/wiki/List_of_languages_by_total_number_of_speakers

Hindi facilitated the process. The main problem with the statistical approaches is the tagged data. In the wake of maiden steps taken by the research community to develop treebanks for other Indian languages, in our dissertation we propose data driven syntactic parsers for other resource poor languages.

Syntactic parsing utilizes information gained at lower levels of text analysis, to establish relationships, which can be both syntactic (grammatical) or syntactico-semantic (partially conveying meaning also), between words of a sentence. The tools like morph analyzers, chunkers etc, which give these lower levels of analysis are expensive to build. Research in mainstream NLP has slowly improved moving from a cumbersome rules based to a pattern learning based computer programming methodology. The traditional approaches require a lot of manual intervention for annotation, cleaning of data, devising task specific features. We thus explore the new school frameworks like neural network and vector space modeling to reduce the overall cost of syntactic parsing. Cheap resources like mono lingual data are exploited to get the distributed representations which can replace the features extracted from expensive tools. The reduced cost enables easy integration of parsers in real time NLP applications. But as it is noticed in most cases they perform clumsily because of domain confinement - they are trained on data from finite and very specific domains. Our proposed method of using large mono lingual data which covers sentences from a large variety of domains would help to make the system domain unbound.

1.1 Problem Statement

This disseration titled "Advancements in Dependency Parsing for Indian Languages" highlights our efforts to bring Indian languages at par with other widely represented languages in the world like English, Chinese, Spanish etc which have a lot of labeled resources and state of the art Natural language Processing tools. Indian languages are resource poor and to make them more available for tasks like parsing we present a mapping from the Computational Paninian Grammar to an internationally accpeted scheme - Universal Dependencies. Next we build state of the art dependency parsers using neural networks for five underrepresented languages - Bengali, Marathi, Kannada, Telugu and Malayalam.

1.2 Motivation

Dependency Parsing has been a long standing problem in NLP. Universal Dependencies (UD) are gaining much attention of late for systematic evaluation of cross-lingual techniques for cross-lingual dependency parsing. The effort to convert the existing Hindi treebank to an internationally consistent scheme was driven by the need to bring Hindi at par with other widely used languages of the world, and to construct a mapping between the Computational

Paninian Grammar (which underlie Indian language treebakning) and UD. This would make Hindi a more available resource for multilingual parsing. It would also facilitate building of cross lingual NLP tools for Hindi and other Indian languages and transfer of knowledge from resource rich languages to underrepresented ones. Most work in NLP has been done on languages which already have large scale data like English or Chinese. Among Indian languages, Hindi has received significant attention and has a considerable amount of monolingual data, a large dependency treebank, mapping to a universal scheme of annotation (now as a result of our work), thus enabling building of supervised NLP tools. However for most other Indian languages there exists a little or even no labeled training data for parsing. Keeping in line with the development of Hindi Dependency treebank, Treebanks for Indian Languages (TBIL) - a project funded by DeiTy, Government of India was started for the development of treebanks in Marathi, Bengali, Kannada and Malayalam[2]. We believe that the advances in NLP must deliver widespread benefits. With that motivation in mind we work to apply non linear neural network for parsing five resource poor Indian Languages belonging to two major language families - Indo-Aryan and Dravidian. Bengali and Marathi are Indo-Aryan languages whereas Kannada, Telugu and Malayalam belong to the Dravidian family.

1.3 Contributions

Below are the main contributions of this thesis :

- **Conversion of Computational Paninain Grammar formalism to Universal Dependencies Universal Dependencies (UD) :** We extend UD to Indian languages through conversion of Pāṇinian Dependencies to UD for the Hindi Dependency Treebank (HDTB) and observe the effects of a relatively sparse taxonomy. We discuss the differences in annotation in both the schemes how existing dependency scheme and POS tags for Hindi map onto the universal taxonomy, the issues that were faced and how they have been resolved. We present parsing experiments for both the formalisms and empirically evaluate their weaknesses and strengths for Hindi. We produce an automatically converted Hindi Treebank conforming to the international standard UD scheme, making it useful as a resource for multilingual language technology.

- **Non linear neural network based parser for resource poor Indian languages:** The contribution of this work is manifold. Firstly we try to build the first parsers for languages like Marathi, Kannada and Malayalam. Secondly this is a maiden attempt to propose a non linear framework involving neural network for parsing languages Bengali, Telugu and the above mentioned languages as opposed to traditional computationally

[2]The organizations involved in this project are Jadavpur University-Kolkata (Bengali), MIT-Manipal (Kannada), C-DIT,Trivandrum (Malayalam), IIT-Bombay (Marathi), IIIT-Hyderabad (Hindi).

expensive parsers. Thirdly we try to reduce the cost and error propagation in parsing by replacing he features extracted using costly-to-build tools by their distributed representations. Keeping in mind the varying degree of morphological richness of languages belonging to two major language families, we propose a unified feature set that uses rich syntactic features in neural network parser albeit in a cost effective manner. We empirically gauge the importance of these distributed features for parsing different languages and show how it is consistent with their typology.

- **Exploring granularity of Paninian dependency relations:** Computational Paninian Grammar dependencies are highly fine grained. It is conjectured that the more semantically oriented nature of the Paninian annotation scheme makes labeled parsing more difficult than in schemes based on more surface-oriented grammatical functions and such deep granularity is not suitable for all tasks. Many applications may not require finer dependency labels and running a full parser with large set of fine dependency labels (82) is too expensive. The hierarchical tree for dependency labels has different layers that can be considered as levels of granularity. We proposes parsing with a varied number of labels (levels of granularity) and analyse accuracies and loss of crucial information if any.

- **Auxillary contributions:** Our main aim is to improve parsing accuracies for a real time setting where gold features are not provided. In such a scenario all features must be automatically extracted. For this purpose we built neural network based POS tagger and Chunker for part of speech and chunk information we need in our experiments. These tools are built for all 5 languages Kannada, Malayalam, Telugu, Marathi and Bengali. They are faster than traditional linear based models and uses the same neural network architecture liek that of our parser.

1.4 Outline

- **Chapter 2:** In this chapter we provide necessary background for the thesis. We discuss the formal description of parsing and its frameworks - Constituency and Dependency. We explain why Dependency framework is more suited for our choice of languages. We go on to discuss about the major schemes used worldwide to mark the dependency relations - the Computational Paninian Grammar formalism (that lies at the heart of treebanking and parsing Indian languages), Stanford Dependencies and an internationally accepted scheme for all languages called Universal Dependencies.

- **Chapter 3:** Different approaches for dependency parsing are discussed. This is followed by discussing two paradigms for data driven parsing. We talk about MaltParser a very popular tool used for transition based dependency parsing and also the new school neural network based parsers.

- **Chapter 4:** In this chapter we discuss in detail about one of the most important resource Hindi Dependency Treebank. We discuss about its rich and granular representation. We experimentally try to see the effect of reducing the granularity of the labels on parsing accuracies. We present our work to produce an automatically converted Hindi Treebank conforming to the international standard UD scheme, making it useful as a resource for multilingual language technology. We discuss in detail the suitability of the scheme for Hindi, the differences in CPG and UD scheme for representing different constructions, and the conversion process.

- **Chapter 5:** Traditional parsers rely on millions of hand-crafted features, that limit their generalization ability and slows down the parsing speed. Neural dependency parsing is attractive for several reasons: distributed representation generalizes better than sparse representations of specially engineered features. It's fast training and parsing mechanism enables ease of integration of parsers in the pipeline of other NLP modules. In this chapter we propose neural network based parser for languages from two language families - Indo Aryan and Dravidian.

- **Chapter 6:** In this chapter we present our concluding remarks and outline directions for possible future work.

Chapter 2

Grammar Formalisms and Parsing Annotation Schemes

This thesis revolves around application of transition based dependency parsers to unrestricted natural language text. In this chapter we provide the necessary background about the different parsing formalisms, stating which is suitable for our purposes. We also talk about the different annotation schemes used to mark dependency relations.

2.1 Parsing

Parsing may be defined as the process of assigning structural descriptions to sequences of words in a natural language (or to sequences of symbols derived from word sequences). This assignment of structural description is defined by the grammar of the language and set of structural constraints - according to which the parser attempts to analyse the symbol sequences presented to it. 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.

In this disseration we talk about two major grammar formalisms that underlie the syntactic analysis of a sentence - Context Free Grammar or Constituentcy Grammar and Dependency Grammar.

2.1.1 Phrase Strucuture Grammar

[1] We first talk about the notion of a *constituent*. The fundamental idea of constituency is that groups of words may behave as a single unit or phrase, called a constituent. For example we will see that a group of words called *noun phrase* often acts as a unit; noun phrases include

[1] The contents of the sections below are based on the book Speech and Language Processing. Daniel Jurafsky & James H. Martin.

single words like she or Michael and phrases like the house, Russian Hill, etc. One piece of evidence for why they are called as a single unit is that they can all appear in similar syntactic environments, for example before a verb. We would now talk about context-free grammars, a formalism that allows us to model these constituency facts. It is the most commonly used mathematical system for modeling constituent structure in English and other natural languages. A context-free grammar consists of a set of rules or productions, each of which expresses the ways that symbols of the language can be grouped and ordered together, and a lexicon of words and symbols. The symbols that are used in a CFG are divided into two classes. The symbols that correspond to words in the language ("the", "nightclub") are called *terminal* symbols; the lexicon is the set of rules that introduce these terminal symbols. The symbols that express clusters or generalizations of these are called *non-terminals*. Phrase structure formalism aims to break a sentence into its constituents known as syntactic categories - both phrasal and lexical. The structures given by this scheme are nested and often recursive.

2.2 Dependency Grammar

We now talk about another class of grammar formalisms called dependency grammars where constituents and phrase-structure rules do not play any fundamental role. Instead, the syntactic structure of a sentence is described purely in terms of words and binary asymmetrical semantic or syntactic relations between these words. Grammatical relations exist between individual words with directed, la- beled arcs from heads to dependents. We call this a typed dependency structure because the labels are drawn from a fixed inventory of grammatical relations. It does not propose a recursive structure like constituency grammar. It also includes a root node that explicitly marks the root of the tree, the head of the entire structure. 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.*'

In general, for languages with rather fixed word order patterns a phrase structure representation may be found more suitable. Dependency representations, in contrast, may be found more adequate for languages which allow greater freedom of word order and in which linearisation is controlled more by pragmatic than by syntactic factors. For example, word order in Czech can be much more flexible than in English; a grammatical object might occur before or after a location adverbial. A phrase-structure grammar would need a separate rule for each possible place in the parse tree where such an adverbial phrase could occur. A dependency-based approach would just have one link type representing this particular adverbial relation. Thus, a dependency grammar approach abstracts away from word-order information, representing only the information that is necessary for the parse. An additional practical motivation for a dependency-based approach is that the head-dependent relations provide an approximation

to the semantic relationship between predicates and their arguments that makes them directly useful for many applications such as coreference resolution, question answering and information extraction.

Indian languages are morphologically rich and free word order. It is shown that dependency formalism can handle these better than constituency based [45] [83] [12]. Learning in Dependency Grammar is more straightforward, the fragments are interpretable and directly give predicate-argument structure. It is not dependent on the word order of a sentence like Constituency Grammar.

In the sections below we discuss the major annotation schemes that are adopted worldwide to mark dependency relations for natural language texts.

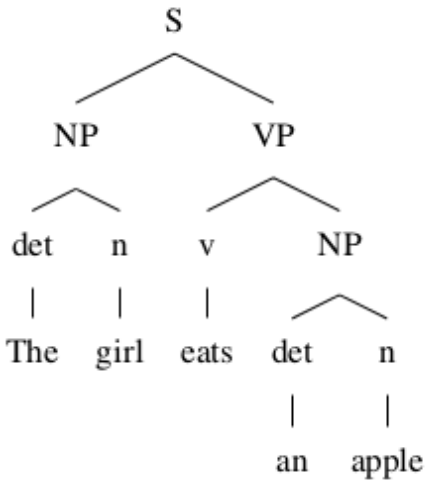


Figure 2.1: Example of constituency based representation

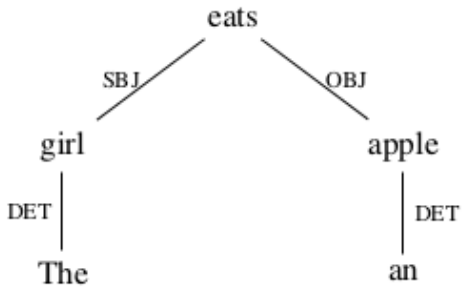


Figure 2.2: Example of dependency based representation

2.3 Computational Paninian Grammar

Computational P¯an.inian Grammar (CPG) is a grammar formalism that takes concepts and insights from P¯an.inian Grammar (PG) for computational processing of a natural language text [20]. Pāṇini was an Indian grammarian who is credited with writing a comprehensive grammar of Sanskrit. The underlying theory of his grammar provides a framework for the syntactico-semantic analysis of a sentence. The grammar treats a sentence as a series of modified-modifier relations where one of the elements (usually a verb) is the primary modified. This brings it close to a dependency analysis model as propounded in Tesnière's Dependency Grammar [80]. The syntactico-semantic relations between lexical items provided by the Pāṇinian grammatical model can be split into two types[2]:

1. **Kāraka**: These are semantically related to a verb as the direct participants in the action denoted by a verb root. The grammatical model has six 'kārakas', namely **'kartā'** (the doer), **'karma'** (the locus of action's result), **'karaṇa'** (instrument), **'sampradāna'** (recipient), **'apādāna'** (source), and **'adhikaraṇa'** (location). These relations provide crucial information about the main action stated in a sentence.

2. **Non-kāraka**: These relations include reason, purpose, possession, adjectival or adverbial modifications etc.

Both the **Kāraka** and **Non-kāraka** relations in the scheme are represented in Figure 2.3; glosses of these relations are given in Table 2.1 . The purpose of choosing a hierarchical model for relation types is to have the possibility of underspecifying certain relations.

2.4 Stanford Dependencies

The Stanford typed dependencies (SD) representation was designed to provide a simple description of the grammatical relationships in a sentence that can easily be understood and effectively used by people without linguistic expertise who want to extract textual relations; as mentioned in the Stanford Dependency Manual [33]. The style of the SD representation is based on the design principle of the framework of Lexical Functional Grammar [20]. It draws inspiration from the grammatical relations and the naming defined in two representations that follow an LFG style: the GR [22] and PARC [54] schemes. These were used as a starting point for developing the Stanford dependencies [32]. The grammatical relations of SD are arranged in a hierarchy, containing 56 relations. However, five variants of the typed dependency representation are available in the dependency extraction system provided with the Stanford parser. The differences between the five formats are that they range from a more surface-oriented representation, where each token appears as a dependent in a tree, to a more semantically interpreted

[2]The complete set of dependency relation types can be found in [6]

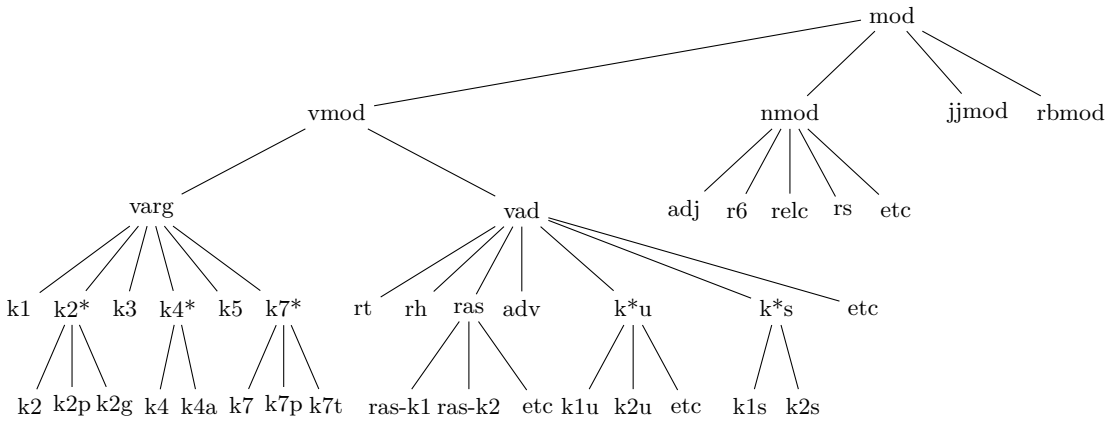


Figure 2.3: Inter-chunk dependency labels

representation where certain word relationships, such as prepositions, are represented as dependencies, and the set of dependencies becomes a possibly cyclic graph [33]. The design principles of SD put forward by them which on the surface level, seem in contrast to CPG are:

- Underspecified relations should be available to deal with the complexities of real text.
- Where possible, relations should be between content words, not indirectly mediated via function words.
- The representation should be spartan rather than overwhelming with linguistic details.

Dependency schemes for some of the other languages such as the Chinese Stanford Dependencies [24]has been based on the Standford Dependency Relations. There has been some efforts in defining a consistent scheme of cross-linguistic set of relations in the style of Stanford Dependencies [65], [83]. A reformulation of the existing scheme was proposed in the form of Universal Stanford Dependencies [31]. Further refinement with a team of collaborators has led to a new synthesis spanning tokenization, morphological features, parts of speech, and dependencies, known as Universal Dependencies (http://www.universaldependencies.org).

2.5 Universal Dependencies

As described by [31] when they first introduced Universal Dependencies it was proposed as an improvement over the Standford Dependencies as discussed above. This was done to capture cross linguistic typology by being able to represent grammatical relations across all languages including morphologically rich ones. They tried to achieve an appropriate parallelism between

Relation	Meaning
k1	Agent / Subject / Doer
k2*	Theme / Patient / Goal
k3	Instrument
k4*	Recipient / Experiencer
k5	Source
k7*	Spatio-temporal
rt	Purpose
rh	Cause
ras	Associative
k*u	Comparative
k*s	(Predicative) Noun / Adjective Complements
r6	Genitives
relc	Modification by Relative Clause
rs	Noun Complements (Appositive)
adv	Verb modifier
adj	Noun modifier

Table 2.1: Some major dependency relations depicted in Figure 2.3.

expressing grammatical relations versus morphology. They suggested a two-layered taxonomy: a set of broadly attested universal grammatical relations, to which language-specific relations can be added. The goal of the new universal version was to add or refine relations in the Stanford Dependencies to better accommodate the grammatical structures of typologically different languages and to clean up some of the relations which were more English-specific. Hence, the new taxonomy has less relations than the original SD. Several efforts have been made for different languages to show how their existing dependency schemes could map onto the universal taxonomy proposed. About 102 treebanks in 60 languages have been converted and released [3]

[3]as on November 15, 2017

with the goal of facilitating multilingual parser development, cross-lingual learning, and parsing research from a language typology perspective.

Below are listed the changes to the representation that affect the structure of dependency trees[4].

- **Treatment of Copular Constructions**

  "The first difference between between Universal Dependencies and Stanford Dependencies is the treatment of copular constructions. While only nominal and adjectival predicates were treated as heads of copular constructions in the Stanford Dependencies representation, we also treat prepositional and adverbial predicates as the head of copular constructions in the Universal Dependencies representation.""

- **Treatment of Prepositional Phrases**

  "In the Stanford Dependencies representation the preposition is always being treated the head of the prepositional phrase while in the Universal Dependencies representation the head of a prepositional phrase is always a content word – either a noun phrase or the main predicate of a clause introduced by a preposition. The preposition itself is attached to the head of the prepositional phrase with the case case or a marker relation. If the prepositional phrase only consists of a preposition followed by a noun phrase we use the `nmod` relation between the head of the prepositional phrase and the noun or predicate which it modifies. If the prepositional phrase consists of a preposition followed by a clause we either use the `acl`or `advcl` relation between the head of the prepositional phrase and the noun or predicate it modifies.""

- **Multi-Word Expressions**

  "There are two differences regarding the treatment of multi-word expressions in Universal Dependencies as compared to Stanford Dependencies. First, multi-word expressions are now always head-inital and all other words in the expression depend on the first word. Second, the list of expressions which are being treated as multi-word expressions changed.""

The UD representation also introduces the following new relations which did not exist in SD like `dislocated, foreign, list, remnant, vocative`. The gloss of these relations and the complete list can be found in Table 2.2.

[4]As described originally in http://universaldependencies.org/docs/en/overview/migration-guidelines.html

Relation	Meaning
acl	clausal modifier of noun (adjectival clause)
advcl	adverbial clause modifier
advmod	adverbial modifier
amod	adjectival modifier
appos	appositional modifier
aux	auxiliary
case	case marking
cc	coordinating conjunction
ccomp	clausal complement
clf	classifier
compound	compound
conj	conjunct
cop	copula
csubj	clausal subject
dep	unspecified dependency
det	determiner
discourse	discourse element
dislocated	dislocated elements
expl	expletive
fixed	fixed multiword expression
flat	flat multiword expression
goeswith	goes with
iobj	indirect object
list	list
mark	marker
nmod	nominal modifier
nsubj	nominal subject
nummod	numeric modifier
obj	object
obl	oblique nominal
orphan	orphan
parataxis	parataxis
punct	punctuation
reparandum	overridden disfluency
root	root
vocative	vocative
xcomp	open clausal complement

Table 2.2: The dependency relations of the Universal Dependecies scheme [version 2].

Chapter *3*

Approaches to Dependency Parsing

Let us look at the formal description of Dependency Parsing and the different approaches adopted to achieve it. (The following section is based on Prashant Mannem's slides[1] and the book Speech and Language Processing. Daniel Jurafsky & James H. Martin [2]). We talk about the two techniques used - grammar driven and data driven. We further go on to discuss different ways of performing data driven parsing like transition based and graph based.

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 ∈ 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*: There exists only one head for wach word.
- *Acyclic*: There are no cycles in the graph.
- *Connected*: The graph should be a single component 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).

In general, dependency parsing can be broadly divided into grammar-driven (where grammatical rules are used for parsing) and data-driven (where annotated/un-annotated data is used for training of the model) dependency parsing. These approaches are not mutually exclusive, though. We talk about these approaches in the next sections.

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

[2] https://web.stanford.edu/ jurafsky/slp3/14.pdf

3.0.1 Grammar driven

The grammar-driven constraint based dependency parsing is based on the notion of eliminative parsing, where analysis of sentences take place 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.

One of the hardest problems for the grammar-driven approach has traditionally been to achieve robustness that is whether the system can analyze any unrestricted input text. This formalism makes a crucial assumption that the formal grammar defined for a language is a reasonable approximation of the language. Where as in practice, there are many instances when a language input cannot be derived by the formal grammar of the language. Natural languages are not much restricted to certain rules. Secondly as a result of constraint satisfaction technique, there may exist more than one analysis for an input sentence, that can lead to the problem of disambiguation for the correct parse.

3.0.2 Data driven

This parsing technique uses supervised machine learning methods over annotated data in order to give a syntactic analysis of an input sentence. This technique gives an analysis by breaking the problem into two subproblems (a) the task of using a training data that is a representative sample of structured sentences for a language to learn a parsing model, called the *learning problem* (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. Eisner (1996) [38] was the first to establish data driven dependency parsing technique using graph based methods. Kudo and Matsumoto (2002) [59] and Yamada and Matsumoto (2003)[91] were among the first ones to explore transition based approach.

As opposed to its grammar driven counterpart, data driven parsing ensures that for unrestricted natural language text there is an output, thereby making it a more robust approach. These parsers also do not create the problem of disambiguation for the user as they generally give a single best output.

3.1 Approaches to Data Driven Parsing

3.1.1 Graph based parsing

Graph-based approaches to dependency parsing search through the space of possible trees for a given sentence for a tree (or trees) that maximize some score. The search space encoded as directed graphs by these methods, that use techniques from graph theory to search the space for optimal solutions. Formally speaking, given a sentence S we look for the best dependency tree in G, the space of all possible trees for that sentence, that maximizes some score. The overall score for a tree can be viewed as a function of the scores of the parts of the tree. One of the ways to achieve it through edge-factored approaches where the edge-factored score for a tree is based on the scores of the edges that comprise the tree. There are several motivations for the use of graph-based methods. First, unlike transition-based approaches, these methods are capable of producing non-projective trees. Although projectivity is not a significant issue for English, it is definitely a problem for many of the world's languages, for our purposes all the Indian languages. A second motivation concerns parsing accuracy, particularly with respect to longer dependencies. Empirically, transition based methods have high accuracy on shorter dependency relations but accuracy declines significantly as the distance between the head and dependent increases [63]. Graph-based methods avoid this difficulty by scoring entire trees, rather than relying on greedy local decisions. We will now look at the widely-studied approach based on the use of a maximum spanning tree (MST) algorithm for weighted, directed graphs.

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 [28],[37].

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 [40].

3.1.2 Transition based parsing

A key element in transition-based parsing is the notion of a configuration which consists of a stack, an input buffer of words, or tokens, and a set of relations representing a dependency tree. Given this framework, the parsing process consists of a sequence of transitions through the space of possible configurations. The goal of this process is to find a final configuration where all the words have been accounted for and an appropriate dependency tree has been synthesized. To implement such a search, there are a defined set of transition operators, which when applied to a configuration produce new configurations. The operation of a parser can now be viewed as a search through a space of configurations for a sequence of transitions that leads from a start state to a desired goal state.

In the last two decades, a number of incremental parsing algorithms have been proposed to parse natural language text. In this thesis, we restrict our choice to arc-eager system. 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)[73].

3.1.2.1 MaltParser

One of the most popular tools to perform transition based parsing is the MaltParser. MaltParser is an implementation of inductive dependency parsing, where the syntactic analysis of a sentence amounts to the derivation of a dependency structure, and where inductive machine

learning is used to guide the parser at nondeterministic choice points (Nivre, 2006) [69]. It uses no grammar but relies completely on inductive learning from treebank data. The parsing methodology is based on three essential components as mentioned in (Nivre, 2007) [72]:

- Deterministic parsing algorithms for building labeled dependency graphs ([59],[91],nivre2003efficient)
- History-based models for predicting the next parser action at nondeterministic choice points (Black et al., 1992; Magerman, 1995; Ratnaparkhi, 1997; Collins, 1999)
- Discriminative learning to map histories to parser actions ([59],[91],[68], [46])

. MaltParser implements nine deterministic parsing algorithms - Nivre arc-eager, Nivre arc-standard, Covington non-projecive to name a few. MaltParser allows users to define feature models of arbitrary complexity. The features used in Malt are all symbolic and extracted from the following fields of the CoNLL data representation (Buchholz and Marsi, 2006) [21]: FORM, LEMMA , CPOSTAG , POSTAG , FEATS , and DEPREL. Symbolic features are converted to numerical features using the standard technique of binarization. Feature optimization in MaltParser is achieved by defining a base model and using forward and backward feature selection algorithms for language-specific feature selection. It uses LIBSVM - A Library for Support Vector Machines[23] and LIBLINEAR – A Library for Large Linear Classification [39] to build a classifier to predict the correct transition.

3.1.2.2 Neural Networks

For a long time NLP tasks used linear machine learning approaches like Support Vector Machines, logistic regression that were trained over very sparse indicator features. These features required a lot of expertise to design and manual labour for annotation of training data. With the advent of nerual networks, that is a non linear architecture we can easily exploit dense representations of these features and use them for our classification or labeling task. People in the research community were quick to adopt them for syntactic parsing. Chen and Manning were the first to present a feed forward neural network for classification of actions as per the configuration in a transition based parser. They use dense low-dimensional vector for core features which are concatenated and fed into a non- linear classifier (multi-layer perceptron) which can potentially capture arbitrary feature combinations. Weiss et al. (2015) [88] used the same approach while improving the set of core features. They used a more involved network architecture with skip-layers, beam search-decoding, and careful hyper-parameter tuning. While neural-network classifiers alleviates the need for hand-crafting feature combinations, defining core features remain to be an important task. The above mentioned efforts are in the direction of coming up with effective feature combinations by hand-crafting them. The feature-engineering problem was later attacked by coming up with novel neural-network architectures for encoding the parser state, representing intermediately-built subtrees in the form of vectors which are

then fed to non-linear classifiers. Dyer et al. (2015) [36], Zhu et al. (2015) [93], Kiperwasser and Goldberg (2016) [55] have worked in this direction using convolutional neural networks, LSTMs and Bidirectional LSTMs. For our work in this thesis we use the simple feed forward neural network.

3.2 Summary

We have laid the background for our work by talking about the two major grammar formalisms, techniques used to perform dependency parsing, the linear and non linear frameworks in transition based parsing. The different schemes used to annotate dependency relations we also discussed. We would now present in the next chapter, one of the main contributions of our thesis. It is the conversion of the Hindi Dependency treebank from Computational Paninian Grammar to Universal Dependencies to represnt it the consistent annotation that is being adopted by major languages of the world.

Chapter 4

Conversion from Pāṇinian Kārakas to Universal Dependencies for Hindi Dependency Treebank

In this chapter we provide a mapping to convert one of the most important resources in Indian Language Treebanking - the Hindi Dependency Treebank from Computational Paninai Grammar to Universal Dependencies. CPG is highly fine grained whereas UD is a coarser scheme. In this chapter we also present some experiments to explore the granularity of the Paninian scheme. We see if reducing the granularity of the labels has any effect on parsing accuracies or results in a lot of information loss.

4.1 Introduction

Universal Dependencies is a project undertaken to develop an inventory of languages that have treebanks annotated in a consistent scheme [65]. The UD annotation has evolved by reconstruction of the Standford Dependencies [33] and it uses a slightly extended version of Google universal tag set for part of speech (POS) [75]. This is done with the motivation to facilitate the efforts in building of cross-linguistic tools such as parsers, translation systems, search engines, etc.

The efforts in building similarly structured or annotated treebanks have invoked a lot intreset from researchers around the world. The first release of UD treebanks included six languages where English and Swedish were created by automatic conversion. Thereafter several other treebanks have been developed automatically such as Italian [19], Russian [60], and Finnish [76]. Several treebanks have also been created using manual annotation procedures. For languages where a treebank is already available, automatic conversion process is more suitable than manual annotation which is expensive and time consuming. It should be noted here that while for some languages conversion between the original and the UD representations can be accurate, for others it may introduce too much noise.

There have been few attempts that have tried to convert the annotation scheme used for Indian languages to other schemes such as the annotation style of Prague Dependency Treebank [92]. Our work, instead, aims to convert HDTB annotation scheme to UD.

Keeping in line with the ongoing efforts in this direction, our work is a volunteer effort to harmonize the Hindi Dependency Treebank according to the UD formalism, making it a more available resource for multilingual parsing. In doing so, we have converted the dependency relations in Pāṇinian framework and the POS tag set followed by Hindi to the Universal Dependency scheme. This conversion had its challenges, as many language specific phenomena had to be addressed in the process. However, there was no requirement to develop a new, language specific UD-scheme, unlike some other treebanks, for instance Russian [60].

4.2 The Two Schemes

4.2.1 Computational Paninian Grammar and Hindi Dependency Treebank

Manually annotated data is a necessity for data driven basic and advanced NLP applications, therefore triggering efforts for the creation of these linguistic resources. Specifically, syntactic treebanking projects have generated a lot of interest in the community due to their manifold usage. A syntactic treebank is, by definition, a set of syntactic trees capturing the syntactic or semantic structure of sentences.

Creation of these treebanks has caught the interest of linguists and computational linguists alike. For the former, they provide insights about the linguistic theory the treebanks ave been built upon, and the later use them for the development of data driven parsers. One such brilliant effort which led to the development of state-of-the-art syntactic parsers for Hindi was the Hindi Treebanking project.

The Hindi Treebank contains text from news articles and heritage domain. It consists of 434,856 tokens in 20,783 sentences with an average of 20.92 words per sentence as can be seen in Table 4.1. It is multi-layered and multi-representational [16, 89, 74, 15]. It contains three layers of annotation namely **dependency structure (DS)** for annotation of modified-modifier relations, **PropBank-style annotation** for predicate-argument structure, and an independently motivated **phrase-structure annotation**. Each layer has its own framework, annotation scheme, and detailed annotation guidelines.

Dependency Structure−the first layer in these treebanks−involves dependency analysis based on the Pāṇinian Grammatical framework [5, 4] as discussed in Chapter 2. Computational Paninian Grammar formalism that underlie the dependency annotation scheme used for Indian language treebanking. It uses different types of relations to signify syntacto-semantic relations between different words and the verb. In our work we will focus on the scheme adopted for the depedency layer and provide a mapping from this scheme to Universal Dependencies.

Count of	HDTB
Types	22,171
Tokens	434,856
Chunks	233,864
Sentences	20,783
Avg. Tokens/Sentence	20.92
Avg. Chunks/Sentence	11.25

Count of	Training	Testing
Tokens	3,47,744	87,112
Chunks	1,87,029	46,835
Sentences	16,629	4,154

Table 4.1: General Treebank Statistics and training-testing split for all the experiments reported in this work.

A generic tree-banking pipeline which consists of a series of steps was devised for Indian languages. Annotation begins with the tokenisation of the raw text. These tokens are then marked with morphological and POS tag information. After the word level annotations, the next step in the pipeline is chukning - the grouping of correlated, inseparable words into chunks (non-recursive phrase). These markings are done using automated tools. 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.

4.2.2 Universal Dependencies

Universal dependencies differ from the CPG framework. As mentioned by [71] and also discussed by [52], the driving principles of UD formalism are:

1. **Content over function:** Content words form the backbone of the syntactic representation. Giving priority to dependency relations between content words increases the probability of finding parallel structures across languages, since function words in one language often correspond to morphological inflection (or nothing at all) in other languages. Functional heads are instead represented as specifying features of content words, using dedicated relation labels.

2. **Head-first:** In spans where it is not immediately clear which element is the head (the content-over-function rule does not apply straightforwardly), UD takes a head-first approach: the first element in the span becomes the head, and the rest of the span elements attach to it. This applies mostly to coordinations, multiword expressions, and proper names.

3. **Single root attachment:** There should be just one node with the root dependency relation in every tree, attached to the artificial root governor.

4.3 Differences in Design

While mapping the two annotation schemes we found that most of the tags entailed multiple correspondences, either one-to-many or many-to-one mappings between their tags. Below, some of the differences are discussed in detail.

4.3.1 Part of Speech Tags

The UD POS tag-set comprises 17 different tags only, whereas the POS tag set developed for Indian Languages [11], has 32 tags.

4.3.1.0.1 One to many (HDTB to UD) The POS tag `WQ` used in the Hindi treebank for question words maps to `DET`, `PRON` and `ADV` in the UD.

4.3.1.0.2 Many to one (HDTB to UD) Similarly several tags on the Hindi treebank side `RB`, `WQ` etc. map with the UD POS tag `ADV`. Though we have a POS tag `RB` which directly maps with the grammatical category `Adverb`, our POS tagging scheme being more granular, we have various tags to annotate different kinds of adverbs. Tags such as `WQ` for question words ('kaha.N' कहाँ (where), 'kab' कब (when), 'kaise' कैसे (how)), `NN` (for words such as 'kal' कल (yesterday/tomorrow), 'Aj' आज (today)), `NST`, `INTF`, etc. are covered by UD under the umbrella tag `ADV`.

Hindi as compound conjunctions like 'aur to aur' और तो और (all the more) and 'jaisA ki' जैसा कि (like/as) etc. In HDTB these are tagged as और_CCC तो_CCC और_CC.
Multiword names are also marked by POS tags `NNPC` and `NNP`. However in UD compounding is marked at the level of dependency relations by three tags: `compound`, `mwe` and `name`.

The Hindi tag set does not tag subordinate and coordinate conjunct differently. Our tags `CC` and `CCC` map with both, `CONJ` and `SCONJ` of UD for all simple and compound conjunctions.

There is not always a straight forward equivalence class mapping from HDTB to UD. The correct mapping of some tags requires the knowledge of lemma or the syntactic context. For

UD	HDTB
ADJ	JJ, JJC, QO
ADP	PSP, PSPC
ADV	RB, RBC, INTF, INTFC,
	NST, WQ, PRP, NN
AUX	VAUX, VAUXC
AUX	VAUXC
CONJ	CC, CCC
DET	DEM, QF, QFC, WQ
INTJ	INJ
NOUN	NN, NNC
NUM	QC, QCC
PART	RP, RPC, NEG
PART	NEG
PRON	PRP, PRPC, WQ
PROPN	NNP, NNPC
PUNCT	SYM
SCONJ	CC, CCC
VERB	VM, VMC
X	UNK

Table 4.2: Mappings of HDTB and Universal Dependencies POS tags.

example, the ambiguity in `WQ` and `CC` is resolved by using a word list corresponding to each Universal POS mapping and a few heuristics derived from the dependency tree structure.

In Indian Languages, there is a phenomenon called reduplication that involves the doubling of a lexical item to convey a grammatical function, such as plurality or intensification. The first word in such reduplicative construction is tagged by its respective lexical category and the second word is tagged as `RDP` to indicate that it is a case of reduplication, thus distinguishing it from a normal sequence [11]. UD does not have a corresponding tag for `RDP` which marks reduplication.

A mapping of all the 17 UD POS Tags to HDTB POS Tag set can be seen in the Table 4.2.

4.3.2 Dependency Relations

In the above section, we found profound ambiguity in mapping the Hindi POS tags to their corresponding UD tags. In case of dependency relations, we also witnesses many cases of `many-to-one` and `one-to-many` mappings. For example dependency relations such as `k3` (instrument of an action), `k7t, k7p` (location in time and space respectively), `r6` (possession relation between two nouns), are all mapped to the label `nmod` (denoting nominal modifiers) in the UD scheme. Likewise, the Pāṇinian relation label `k2` maps to `xcomp, ccomp` and `dobj`, based on 'Chunk[1] condition' described in section 4.4. The relation labels `k1, k4a, pk1` (loosely corresponding to agent, experiencer, causer respectively) all map to the label `nsubj` of the UD scheme.

4.3.3 Dependency Structure

In the Pāṇinian scheme there are about 82 kāraka and non kāraka relations. However, in UD there are only 40 dependency relations now, as opposed to 42 which were mentioned in [31]. The mapping between the two scheme can be seen in Table 4.3. One of the major challenges we came across during the conversion process was the conversion of elliptical constructions.

4.3.3.1 Copula

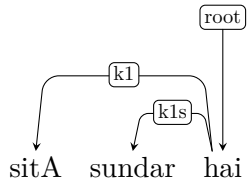
Currently in our scheme, a copular verb is considered as the head of a copula construction. During the conversion to UD, predicative nominal in the copula construction is marked as the head instead, while the 'be' verb becomes a `cop` dependent. For example in sentence (1) Sita is beautiful, 'sundar' सुंदर (beautiful) is treated as the head, while 'sItA' सीता (Sita) and the be verb 'hai' है (is) are its dependents of the type `nsubject` and `cop`, respectively.

(4.1) `सीता सुंदर है.'
'sItA sundar hai.'

sItA *sundar* *hai*
sita beautiful is
'Sita is beautiful.'

For conversion to UD, these relations must be reversed, not just relabeled, which in turn may cause structural changes of other kinds. For example, a reanalysis must be done for dependents of the previous governor and decision be made whether they should attach to the new governor or remain as they were. Thus, for conversion to UD these relations are reversed though it leads to structural changes as can be seen in Figure 4.1.

[1]A chunk (with boundaries marked), in HDTB, by definition, represents a group of adjacent words in a sentence, that are in dependency relation with each other, and where one of these words is their head [25]



(a)

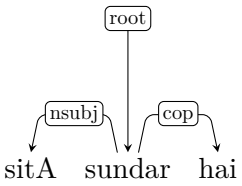


Figure 4.1: Dependency tree for a) HDTB and
b) UD copula constructions.

4.3.3.2 Conjunctions

Another type of constructions we handle are those with conjunctions. In HDTB annotation scheme a conjunction, either coordinating or subordinating is the head of the clause and the other elements of the clause are its dependents. In the sentence such as in Example (2), 'aur' (and) is annotated the head with 'rAm' (Ram), 'mohan' (Mohan), 'sItA' (Sita), and 'mIrA' (Meera) as its dependents.

(4.2) `राम, मोहन, सीता और मीरा आज आए थे.'
'rAm, mohan, sItA aur mIrA Aj Aye the.'

rAm, mohAn, sItA aur mIrA Aj Aye-the
rAm mohAn sItA and mIrA today came-PAST
'Ram, Mohan, Sita and Meera came today.'

Whereas in UD the first element of the coordinated construction is taken as the head. The conjunct and the other coordinated elements are annotated as its dependents. Given this sentence, in UD, 'rAm' is the head of the construction while 'mohan', 'sItA', 'mIrA' and 'aur' depend on it. Further, while 'mohan', 'sItA', 'mIrA' are annotated with the label `conj`, 'aur' is annotated with the label `CC`, since it is a coordinating conjunction, as can be seen in Figure 4.2. Also, UD annotates subordinating conjunction as `mark`, which is a dependent of the head of the subordinate clause. For the sake of conversion from HDTB to UD we distinguish between coordinating and subordinating conjunctions annotating them as `conj` and `mark`. For this we have enlisted them as two separate classes.

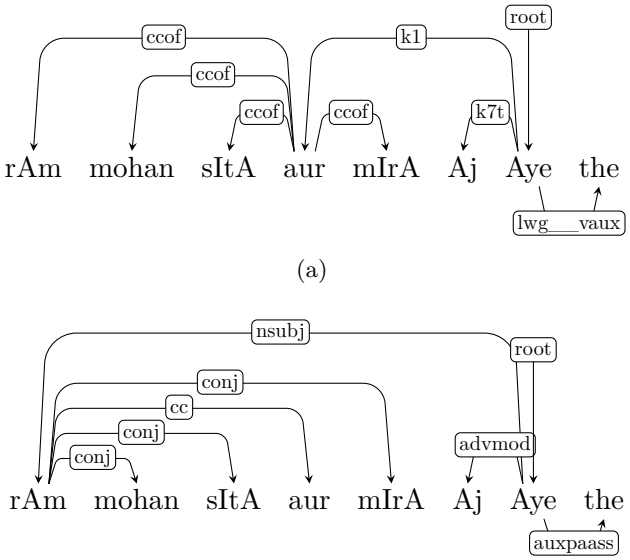


Figure 4.2: Dependency tree for a) HDTB and b) UD conjunctions constructions

4.3.3.3 Multiword names

As has been observed by [52], "In spans where it is not immediately clear which element is the head, UD takes a head-first approach: the first element in the span becomes the head, and the rest of the span elements attach to it. This applies mostly to coordinations, multiword expressions, and proper names." For example, in a name such as 'Atal Bihari Vajpayee', in UD, the first word in a compound name 'Atal', becomes the head and the rest its dependents. Whereas in HDTB, 'Vajpayee' is annotated the head and 'Atal' and 'Bihari' its dependents.

4.3.3.4 Ellipsis

Instances of ellipsis are abundant in the Hindi Treebank. While we are able to handle some in our current conversion, there are others we still need to work on. One such type which we have addressed is the 'yah-ki' यह-कि(this-that) complement constructions which follow the pattern: yah (*यह)-its property-VP-ki* कि*clause'* [61]. In cases of ellipsis, a `NULL` node is introduced to facilitate annotation, since the entire 'ki' (that) clause is annotated as the child of 'yaha' (this) /`NULL'` node here.

In Hindi, sentences such as in Example (3):

(4.3) `गौरतलब है कि गोपाल को नासा आमंत्रित कयिा गया था.'
'gaurtalab hai ki gopAl ko nAsA Amantrit kiyA gayA thA.'

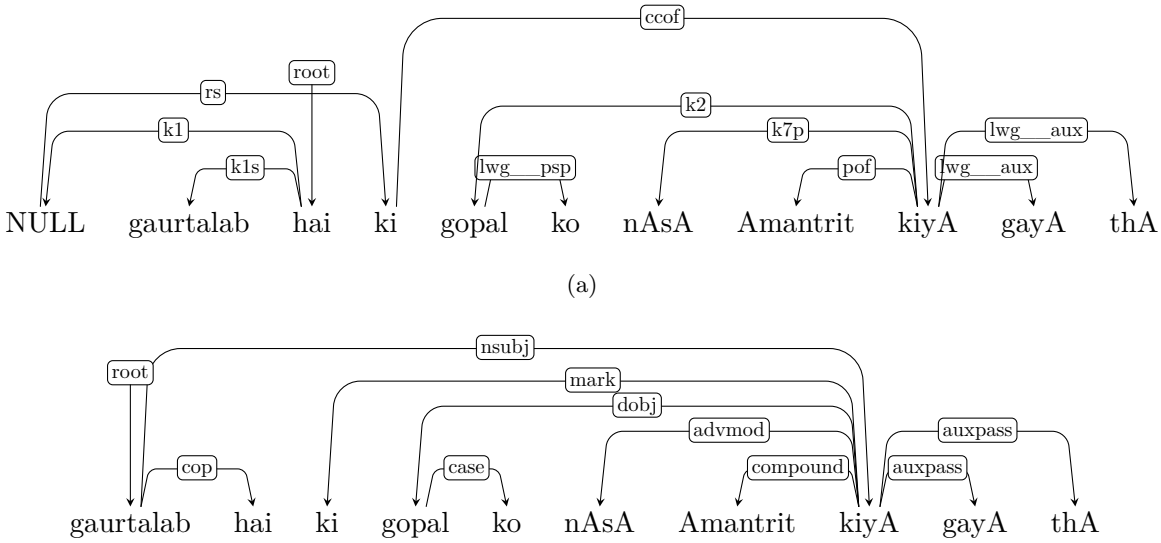


Figure 4.3: Dependency tree for a) HDTB and
b) UD ellipsis constructions.

gaurtalab	*hai*	*ki*	*gopAl-ko*	*nAsA*	*Amantrit-kiyA-gayA-thA.*
to-be-noted	is	that	Gopal	to-NASA	invited-was

'Is to be noted that Gopal was invited to NASA.

'gaurtalab hai ki gopAl ko nAsA Amantrit kiyA gayA thA' (Is to be noted that Gopal was invited to NASA.) can come with the referent 'yah' यह (this) elided (a case of Pronoun drop) or explicitly manifested in the sentence. The 'ki' कि(that) clause and its referent are both modifiers (child) of the verb. However, in HDTB annotation the 'ki' clause is annotated a modifier of its referent which in turn is marked as the child of the verb. For the sake of consistency in cases where 'yah', the referent, does not manifest explicitly, a 'NULL' node is introduced in its stead. However, the UD scheme does not introduce `NULL` tokens to represent elided elements. Therefore to map all 'ki' complement constructions, with the UD scheme, we drop the 'NULL' node, and the 'ki' complement clause is annotated a dependent of the head of the removed `` `NULL' `` node (usually the verb) as illustrated in Figure 4.3.

Universal	**Pāṇinian**
acl	nmod___k1inv, nmod___k2inv, nmod___relc, rs, k2g, k2s, rbmod___relc
neg	nmod___neg
dislocated	fragof
iobj	k4
nmod	k2u, jk1, k1u, k3, k3u, k2p, k4u, k5, k7, k7a, k7p, k7pu, k7t, k7tu, k7u, r6, r6-k1, r6-k2, r6v, ras-k1, ras-k1u, ras-k2, ras-k4, ras-k4a, ras-k7, ras-k7p, ras-neg, ras-pof, ras-r6, ras-r6-k2, ras-rt, nmod___emph
punct	rsym
vocative	rad
advmod	rd, rsp, lwg___intf, vmod___adv, jjmod___intf, jjmod, adv, rbmod
dep	lwg___rp, lwg___unk, undef, enm
compound	pof___cn, pof___redup, lwg___rdp, lwg___vm, nmod___pofinv, pof, pof___inv
case	lwg___psp, lwg___nst, psp___cl
neg	lwg___neg
det	mod___wq
dobj	k2, k1s, mk1
amod	nmod___adj, nmod
parataxis	vmod
ccomp	k2
xcomp	k2
aux	lwg___vaux
auxpass	lwg___vaux
nsubj	k1, k4a, pk1
nsubjpass	k1
advcl	rh, rt, rtu, sent-adv, vmod

Table 4.3: Universal mapping of Pāṇinian Dependencies used in HDTB.

4.4 Conversion Process

During conversion it must be noted that we are moving from a very detailed and granular format to a format which is under-specified. The implementation of the conversion was based on the mapping between schemes described above. The conversion script executes as a pipeline of three components, each of which takes as input data in CONLL format and outputs data in the same format. During conversion, structure is handled before labels. The first module harmonizes the structural differences from HDTB to UD by handling ellipsis (and thereafter aligning nodes in the tree after NULL removal), copula constructions, mutliword names and conjunctions. It updates the nodes as modifier-modified relations have been changed. The second and third module converts POS and Dependency relations from HDTB to UD, respectively. The conversion is based on certain heuristics which involve conditions specified in terms of lexical, structural, morphological information and Part of Speech tags. Examples for the different types of conversion conditions are as follows:

- Lexical condition: The POS tag `WQ` of HDTB is converted to `ADV` of UD when expressed by word form or lemma 'kab' (कब), 'kahA.N' (कहाँ), 'kaisA' (कैसा), 'kyun' (क्यों). Else if the node has chunk type as child in its features, it is converted to `DET` of UD; otherwise to `PRON` of UD.

- Morphological condition: If the dependency relation is any of `k1`, `pk1`, `k4a` and the current node's parent has TAM (Tense, Aspect, Modality) feature as *'yA_jA'* (या_जा), the relation is converted to `nsubjpass`; if dependency relation is `lwg__vaux` and there is a presence of the TAM feature, it is converted to `auxpass`. In the absence of this morph feature `lwg__vaux` is converted to `aux`, while `k1`, `pk1`, `k4a` are converted to `nsubj`.

- Chunk condition: If the dependency relation is `k2` and the current node's chunk id is `VGF` (Finite Verb Chunk), the relation is converted to `ccomp`; else if chunk id is `VGNN` (Verb Chunk - Gerund) it is converted to `xcomp`; otherwise to `dobj`.

- POS condition: If the dependency relation is any of `nmod__adj` or `nmod` and the node's POS Tag is `DET` or `DEM` the relation is converted to `det`; if POS is `NUM` it is converted to `nummod`; else if POS is any of `NNP, NNPC, PRP, NN` or `NNC`, it is converted to `nmod`.

During conversion from HDTB to UD we lose 3852 sentences that cannot be restructured according to our current scheme, they are mostly cases of ellipsis (gapping).

4.5 Parsing Experiment

Our motive of conversion from HDTB to UD was not keeping the increase/decrease of parsing accuracy in consideration. The design choices taken, such as the head-first approach,

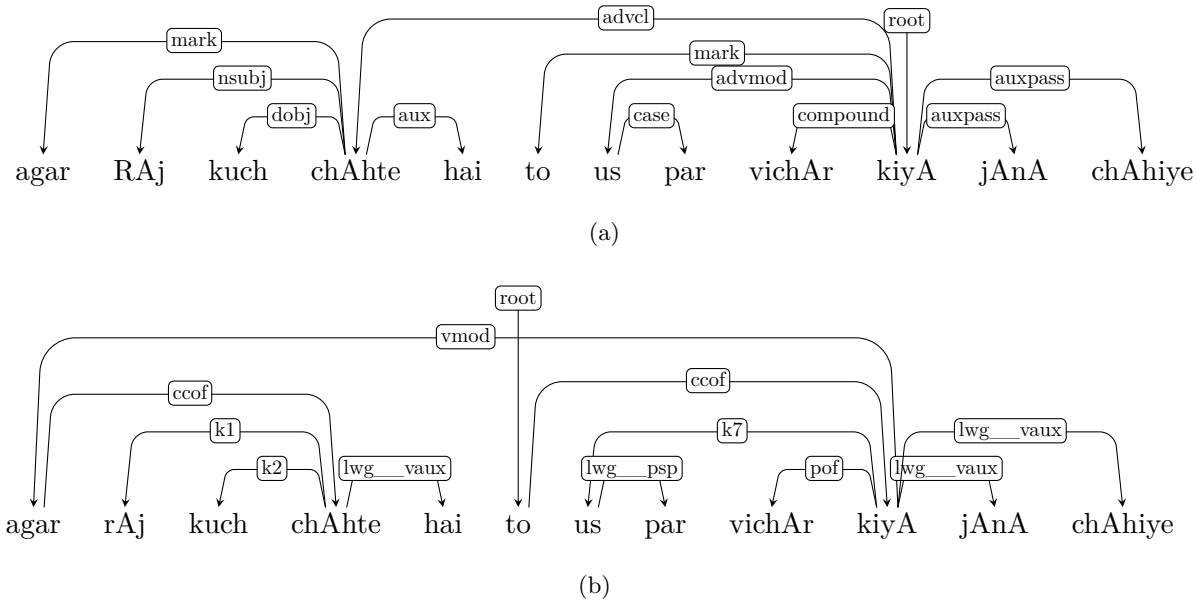


Figure 4.4: Dependency tree for paired connective 'agar-to' for a) UD and b) HDTB.

as described in Section 4.2.2, led to changes in a lot of attachments like that of Copula and decreased the learnability of the parser for the syntactic structure of Hindi. We conduct parsing experiments to quantify these effects and also do a manual error analysis to point out the constructions which are not learnt efficiently by the parser.

For experiment purposes, we are using MALT [2] with parser settings from [3]. The metrics used for evaluation are Labeled and Unlabeled attachment score (LAS, UAS) and Label accuracy score (LS). The average accuracy of 10 fold cross validation is reported in Table 4.4.

We experience an accuracy drop of ∼2% in UAS in conversion from HDTB to UD. This is not surprising as the two are now quite different treebanks. The drop in accuracy can be attributed to the numerous changes in attachment of edges while conversion. However the increase in accuracy of LS is intuitive because of the reduced number of classes of classification for dependency relations.

On doing a manual error analysis we observe the following patterns:

- The parser is not able to learn copula constructions properly, the 'be' verb is not recognised as 'cop' in most cases, it is made the root of the sentence or head of the phrase, instead. This is at odds with the general structure where verb is a root of a dependency tree as

[2]MALT version 1.8.1 , http://www.maltparser.org/

	LAS	UAS	LS
Pāṇinian	90.97	95.206	92.908
UD	90.237	93.193	94.053

Table 4.4: Average accuracy of 10-fold cross validation using Pāṇinian and UD framework.

it is the primary modified. These structures also cannot be learnt efficiently based on lexicalism as the 'be' verb is also used as an auxillary in most cases.

- For control constructions which have more than one verb, the first non-finite verb is marked as the head instead of the finite verb.
- Sentences having paired connectives like 'agar-to' (अगर-तो), 'yadi-to' (यदि-तो) corresponding to 'if-then', do not have their governors and dependents correctly marked. This is because they are handled differently in both the schemes. In HDTB 'agar'and 'to' are clausal heads. The 'to'-clause is modified by the 'agar'-clause. Whereas in UD 'agar'and 'to' must be dependents of the main verb of their respective clauses as can be seen in Figure 4.4.

Thus we briefly described the process of conversion of Hindi Treebank to UD annotation scheme. It was an attempt to release the resource in a widely accepted international format so that it becomes more usable for a variety of multilingual NLP tasks. The conversion was a challenging task and there are constructions which are yet to be addressed to be fully compliant with the UD scheme like that of ellipsis etc. As a part of future work, we propose to come up with better techniques to resolve empty nodes in the absence of predicational or verbal heads. Also a few attachment schemes must be reanalyzed and revised to handle long distance dependencies efficiently. We performed experiments using MALT parser on both the source treebank HDTB and the converted UD Hindi Treebank, to find that the performance is slightly deteriorated after conversion.

4.6 Granularity of the Paninian Scheme

CPG dependencies are highly fine grained. They tell us about the subtle semantic relationships between words in a sentence. This is in contrast to the Standford Dependencies (SD) and its variants such as Universal Dependencies (UD). The Stanford Dependencies representation is

guided by the principle that "The representation should be spartan rather than overwhelming with linguistic details." [33] Thus their dependency types, mainly corresponding to traditional grammatical functions such as subject,object and adverbial. The hierarchy contains 56 grammatical relations as compared to Hindi dependency tagset which contains 82 labels which is comparatively larger than the tag-set for other languages, like Arabic(10), English(55), German(45) etc.

The labeled attachment score is influenced by the complexity of the dependency type classification. With the increase in the number of distinct dependency types the gap between labeled and UAS[3] grows as reported by Nivre and Hall[1] and this drives us to focus on improving LS[4], to boost the overall accuracy LA[5] of the parser. It is conjectured that the more semantically oriented nature of the Paninian annotation scheme makes labeled parsing more difficult than in schemes based on more surface-oriented grammatical functions and such deep granularity is not suitable for all tasks.

Many applications may not require finer dependency labels and running a full parser with large set of fine dependency labels (82) is too expensive. The hierarchical tree for dependency labels has different layers that can be considered as levels of granularity. This section proposes parsing with a varied number of labels (levels of granularity) and analyzing accuracies and factors of time/computational complexity.

In order to understand the effect of granularity we calculate label error among sub-groups of label from the hierarchical scheme. It is seen that our parsers can accurately identify `lwg` and non dependency relations such as `root, conjunction, fragof` and `part-of`.`lwg` or local word group dependency labels intra-chunk relations. Intra-chunk dependencies are mostly syntactic in nature and capture information like case for nouns, tense and aspect for verbs. However `varg` and `vad` seem to be predicted erroneously. `varg` and `vad` represent dependency relations involving a verbal predicate. One of the frequent labeling errors that the parser makes is observed to be between closely related dependency tags, for eg. `k7` (abstract location) and `k7p` (physical location) are often interchangeably marked [77]. We have reasons to believe that such a decision is comparatively tougher for an automatic parser to disambiguate than human validation. The CPG dependency relations are highly fine-grained and have been shown to be strongly correlated with PropBank labels [86]. These finer distinctions in the dependency layer that would nonetheless be captured in PropBank may be dropped off.

4.6.1 Experiments with reducing the granularity

We have seen how the finer dependency relations are difficult to correctly classify for a parser. We make the rich semantic scheme coarser and empirically observe the impact of

[3]Unlabeled Acuuracy Score

[4]Label Accuracy Score

[5]Labeled Attachment Score

Dependencies	Hindi		
	UAS	LS	LAS
CPG	95.88	92.95	91.13
vad (1)	95.84	92.93	91.08
vad (2)	95.77	92.9	91.01
vad (3)	95.78	92.96	91.08
varg (1)	95.7	93.75	91.81
varg (2)	95.85	93.77	91.84
varg (3)	95.89	93.96	92.03
vad (4)	95.89	93.96	92.03
varg (4)	95.91	94.42	92.44

Table 4.5: Impact of annotation granularity on parsing Hindi

granularity of CPG-based dependencies on parsing. We conduct experiments on the Hindi treebank. The granularity of hierarchical dependency relations are reduced by moving up in the tagset hierarchy (Figure 2.3) in right to left bottom-up fashion. We derive the coarse-grained relations for verb argument `varg` and adjunct relations `vad` relations only. At each tree level, first the labels in `vad` are merged and then the labels `varg`. We do not merge all the `varg` relations, keeping the distinctions related to core arguments of a verb namely `k1`, `k2` and `k4` intact, whereas all relations in vad are merged. The procedure is graphically depicted in Figure 4.5

MALT which is linear transition-based parser to conduct the experiments. The results of these experiments are reported in Table 4.5 Surprisingly, reducing granularity in first level `vad` relations such as `ras*` has no positive impact on parsing accuracy. Some improvements in LS is observed by merging of second-level `vad` relations but not as much as expected. Substantial improvements in LS at both levels of granularity is observed on coarsening `varg` relations. Thus it can be concluded that the finer distinctions in `varg` relations at both levels are harder to differentiate. The merging of labels leads to an overall improvement of 1.5% LS across the dataset. It is interesting to note that UAS seems invariant to change in granularity of dependency labels.

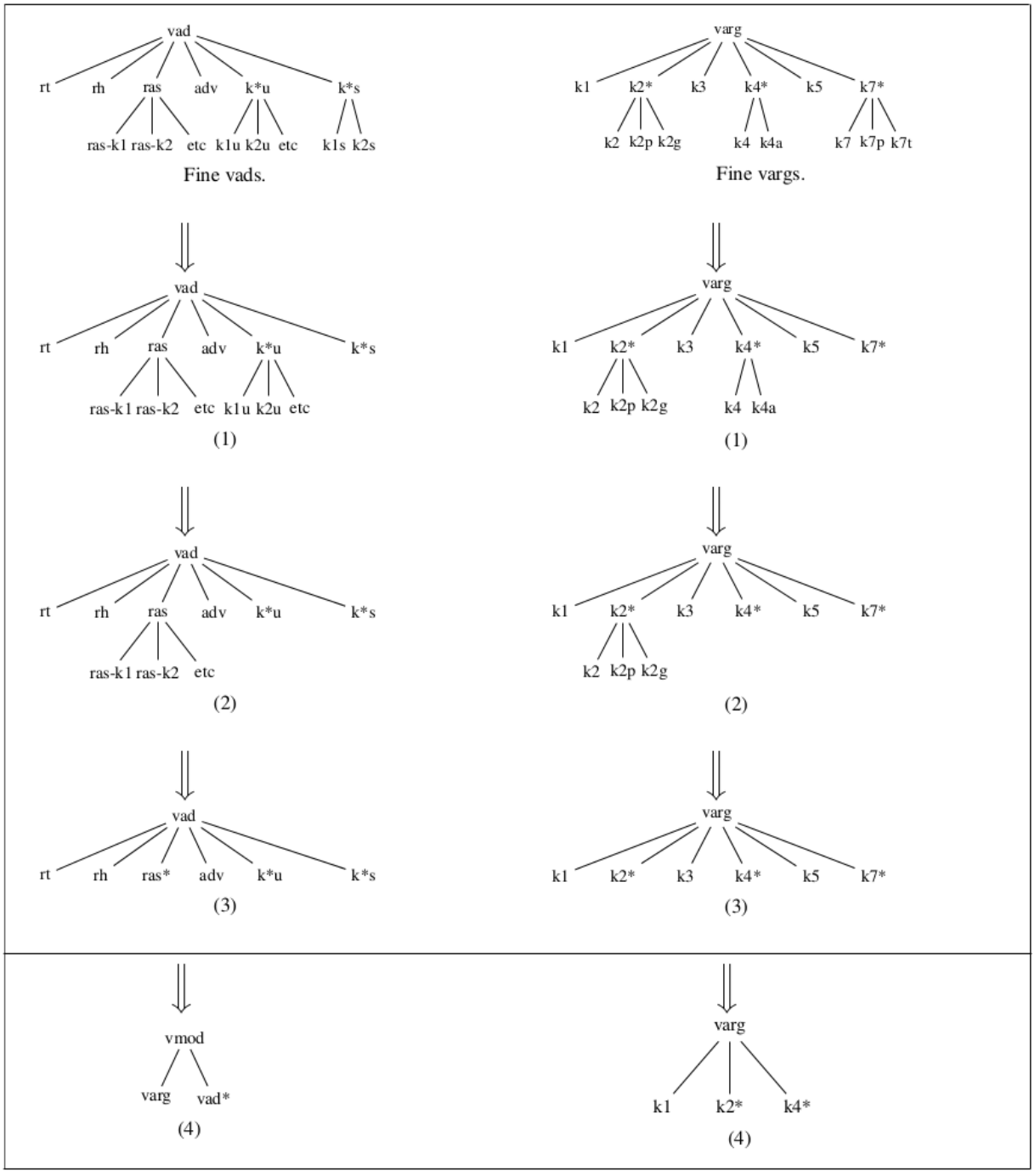


Figure 4.5: Fine-grained and coarse-grained dependency labels for verb arguments and adjuncts.

4.7 Summary

We successfully converted the Hindi Dependency Treebank to the Universal Dependencies scheme taking care of the linguistic phenomenon specific to Indian languages. Universal Dependencies is a coarser scheme as compared to the CPG scheme. Thereafter we experimented by reducing the granularity of the Paninian scheme too to observe the effects. On doing so were able to bridge the performance gap between LS and UAS, on reducing the annotation granularity but may lose crucial information that could be important for downstream applications. As a part of future work the dependencies marked in the Computational Paninian Grammar could be revised for inclusion of more semantically driven choices for marking dependency relations. In the presence of such a richer scheme which takes sound linguistically driven decisions, mapping to any other scheme of annotation would become fairly easy and the issues faced in addressing structural differences might be dealt with efficiently.

Continuing with our efforts to make Indian languages available across the world for building sophisticated NLP tools, we come up with the first neural network parsers for five underrepresented languages from two major language families in the next chapter. We continue to use the Computational Paninian Grammar for representing our dependency trees in the upcoming work as it captures the Indian language intricacies most powerfully.

Chapter *5*

Unity in Diversity: A unified parsing strategy for major Indian languages

We got introduced to different parsing algorithms and tools used in Chapter. We saw how MaltParser that uses SVM was being popularly used as an indispensable tool for transition based parsing. In this chapter we come up with a state-of-the-art parser using non-linear feed forward neural network. We talk about the importance of distributed representations for having non sparse features and reason why neural networks using them is a better choice over the traditional linear framework for dependency parsing. Our work puts forward a cost effective and unified parsing strategy for resource poor languages from two major language families - Indo Aryan and Dravidian. Neural network based part of speech taggers and chunkers for all languages are also built in the process.

5.1 Old school transition based parser vs New school neural network parsers

Over the years there have been several successful attempts in building data driven dependency parsers using rich feature templates [58] requiring a lot of feature engineering expertise. In the conventional transition based parsing approaches indicator features are extracted by the conjunction of 1 to 3 elements (words, POS tags, arc labels etc) from the stack and buffer. Though these indicative features brought enormously high parsing accuracies, they required manual designing of feature templates which may remain incomplete and it is unavoidable to miss the conjunction of every useful word combination. As noted by Chen and manning the conjunction of first element in stack s_1 and second element in buffer b_2 is omitted in almost all commonly used feature templates, however it could indicate that we cannot perform a RIGHT-ARC action if there is an arc from s_1 to b_2. From a statistical perspective these parsers suffer from the use of millions of mainly poorly estimated feature weights. There is usually scarce data to correctly weight high order interaction of these features. They are also computationally

expensive to extract. We have to concatenate some words, POS tags, or arc labels for generating feature strings, and look them up in a huge table containing several millions of features. As reported by Bohnet (2010) [17], his baseline parser spends 99% of its time doing feature extraction, despite that being done in standard efficient ways. Apart from aforementioned problems of incompleteness and expensive extraction, these features also suffer from data sparsity. Lexical features which give cue for word to word interactions and are indispensable for dependency parsing are highly sparse.

5.1.1 Why shall we use Dense Vectors over One hot representations ?

In order to capture meaningful information in the features templates, it is important that they are well represented. Feature templates can be one of the most important factors determining parsing accuracy. In the traditional NLP approaches each feature is represented as a unique dimension (the so called one-hot representation). The number of distinct feature give the dimensionality of the one-hot vector which is usually very high. Feature combinations receive their own dimensions. All values are binary based on the presence or absence of a feature. All the features are completely independent. The trigram "dog drinks milk" is as dissimilar to "cat drinks milk" as it is to "table is broken". Where as in dense representations a feature is represented as a vector of dimension 'd' which is usually low and thus provides computational ease. Similar features have similar vectors. Thus major benefit of dense representations lie in its generalisation power. Earlier occurences of the word 'dog' said nothing about occurences of 'cat'. However now the learned vector for 'dog' may be similar to the learned vector from 'cat', allowing the model to share statistical strength between the two events. Another benefit is that feature combinations need not be encoded explicitly, only core features must be represented.

5.1.2 Why must we prefer neural network classifiers over linear ones for NLP tasks ?

Thus to address the problem of discrete representations of words, distributional representations became a critical component of NLP tasks such as POS tagging [29], constituency parsing [78] and machine translation [34]. These symbolic natural language units such as morphemes, words, phrases, sentences and documents are embedded into a low-dimensional and continuous space such that they solve the problem of sparsity by sharing statistical strength between similar units.

In general, there are two major ways of applying distributed representations to NLP tasks. First, they can be fed into existing supervised NLP systems as augmented features in a semi-supervised manner. This approach has been adopted in a variety of applications [84]. But these simple and effective generalized linear models that are adopted in many NLP applications are not able to fully exploit the potential of distributed representations. The distributed represen-

tations are shown to be more effective in non-linear architectures compared to the traditional linear classifier [87].

Apart from enabling dense representations to be more effective, another advantage of neural network parser is that it does not need to compute conjunction features and look them up in a huge feature table. The feature extraction stage in the neural-network settings deals only with extraction of core features. This is in contrast to the traditional linear-model-based NLP systems in which the feature designer had to manually specify not only the core features of interests but also interactions between them (e.g., introducing not only a feature stating "word is X" and a feature stating "tag is Y" but also combined feature stating "word is X and tag is Y" or sometimes even "word is X, tag is Y and previous word is Z"). The combination features are crucial in linear models because they introduce more dimensions to the input, transforming it into a space where the data-points are closer to being linearly separable. On the other hand, the space of possible combinations is very large, and the feature designer has to spend a lot of time coming up with an effective set of feature combinations. One of the promises of the non-linear neural network models is that one needs to define only the core features. The non-linearity of the classifier, as defined by the network structure, is expected to take care of finding the indicative feature combinations, alleviating the need for feature combination engineering. [1]

Keeping in line with this trend, Chen and Manning [26] introduced a compact neural network based classifier for use in a greedy, transition-based dependency parser that learns using dense vector representations not only of words, but also of part-of-speech (POS) tags, dependency labels, etc. The predictor in transition-based dependency parser is replaced with a well-designed neural network classifier. This model handles the problem of `sparsity, incompleteness and expensive feature computation`. Some work has been done to show why neural network based parsers are better than standard transition based parsers [30]. The authors have done a comparative parser evaluation to show the difference in learning capabilities of the "old school" transition based parsers and their "new school" counterparts that use neural networks. They chose Maltparser and UDPipe[2] for their experiments which are thought to be representative of the two schools. More specifically, they compare MaltParser with Parsito[3] that uses neural networks to learn vector representations of words, POS tags and dependency relations in a similar way to Chen and Manning, it additionally has a search-based oracle. A robust sampling of trees from UD data set was done to cover languages from typologically different categories. They provide their analysis on a variety of typologically different languages and empirically illustrate that neural network models outperform classical models with only a small amount of data. However as the size of the training samples increase, the learning rate of UDPipe start to flatten out and increase similarly as that of Maltparser. Their experiments further show that longer dependencies in sentences pose more difficulty to Maltparser for learning than to neural

[1] This paragraph has been taken from Yoav Goldberg's article on NNlp [42].

[2] http://ufal.mff.cuni.cz/udpipe

[3] https://ufal.mff.cuni.cz/parsito

network models. MaltParser is better, although not by far, for dependencies that are expected to be short.

5.2 Dependency parsing for Indian Languages

Indian languages are mostly morphologically rich and free word order. Because of their non-configurationality parsing is a challenging task. The syntactic cues that are necessary to identify various relations in such languages are complex and distributed. One major issue in these languages is that of argument scrambling. The case markers carry information about relations between the words, so words can freely change their positions in the sentence. This often leads to long distance dependencies non-projective strucutres, which poses problem to parsing. The problem worsens in the context of data-driven dependency parsing due to non-availability of large annotated corpus.

The last decade has seen quite a few attempts at parsing Indian languages. Constraint based methods for parsing have been explored for Hindi by Bharati et al., 2008b [9],Bharati et al., 2009b [8] and by Kesidi, 2013 [53] for Telugu. [45] proposes a semi-supervised graph-based dependency parser for Indian languages. Kolachina et al., 2010 [56] have performed experiments on applying the MaltParser for parsing Indian languages Bengali, Hindi, Telugu by using unexplored features like valency and arc-directionality. They were also the first in exploring ensemble approaches such as blending for parsing these languages. Bharati et al., 2008a [7] noted how information related to gender, number, person (GNP) and basic semantic human-animate-inanimate distinction like are indeed crucial features for disambiguating subject-object in Hindi. Later [**?**] showed how properly incorporating syntactically relevant information like case marking, complex predication and grammatical agreement in an arc-eager parsing model can significantly improve parsing accuracy. It was observed that the rich morphological nature of a language can prove challenging for a statistical parser as in noted by [82]. High performance for morphologically rich, free word order languages can be achieved using vibhakti[4] and information related to tense, aspect, modality (TAM). These syntactic features related to case and TAM marking have been found to be very useful in previous works on dependency parsing of Hindi [3, 48, 49, 14]. Previous works have also shown the addition of chunk tags[5] [2] and grammatical agreement [7, 12] has been proven to help Hindi and Urdu.

While little work has been done for Hindi, Bengali, Telugu, the research in this direction majorly focused on data driven transition-based parsing using MALT [72], MST parser [62]. Other Indian languages do not have been unexplored for syntactic parsing. Only recently Bhat et al., 2016 [13] have used neural network based non-linear parser to learn syntactic

[4] vibhakti is a generic term for postposition and suffix that represent case marking

[5] a chunk is a set of adjacent words which are in dependency relation with each other, and are connected to the rest of the words by a single incoming arc to the chunk

representations of Hindi and Urdu. Following their efforts, we present a similar parser for parsing five Indian Languages namely Bengali, Marathi, Telugu, Kannada, Malayalam. We present the first data dfriven parsers for Marathi, Kannada and MalayalamThese languages belong to two major language families, Indo-Aryan and Dravidian. The Dravidian languages - Telugu, Kannada and Malayalam are highly agglutinative whereas Bengali and Marathi are moderately rich in morphology.

We have seen the rich syntactic and semantic features used in previous works for a few Indian languages. We try to use a part of those features for other Indian languages too as all of them follow more or less the same typology. We decided to experiment with these features too. We propose an efficient way to incorporate the syntactic features of case marking and TAM information in the aforementioned neural network based parser. We use these linguistically motivated morphological features (suffix and postposition) in the non linear framework, to capture the intricacies of both the language families. We put forward ways to represent these features cost effectively using monolingual distributed embeddings. Instead of relying on expensive morphological analyzers to extract the information, these embeddings are used effectively to increase parsing accuracies for resource poor languages. In our model, these features are included as suffix (last 4 characters) embeddings for all nodes. For case and TAM markers occurring in all the chunks their lexical embeddings are included. We also include chunk tags and gender, number, person information as features in our model. Our experiments test the effectiveness of these features for other 5 languages in concern. Computationally, obtaining chunk tags can be done with ease. However, acquiring information related to gender, number, person for new sentences remains a challenge if we aim to parse resource poor languages for which sophisticated tools do not exist. We show that adding both these features definitely increases accuracy but we are able to gain major advantage by just using the lexical features, suffix features and POS tags which can be readily made available for low resource languages.

Finally our experiments provide a comparison between the two language families on the degree of importance of these rich syntactic and morphological features. As an auxilalry contribution, Part of speech taggers and chunkers for all languages are also built in the process, to report (auto) accuracies on real time data.

5.3 Data and Background

5.3.1 Dependency Treebanks

There have been several efforts towards developing robust data driven dependency parsing techniques in the last decade [58]. The efforts, in turn, initiated a parallel drive for building dependency annotated treebanks [83]. Development of Hindi and Urdu multi-layered and multi-representational [16, 89, 74] treebanks was a concerted effort in this direction. In line with these

efforts, treebanks for Kannada, Malayalam, Telugu, Marathi and Bengali are being developed as a part of the Indian Languages - Treebanking Project. The process of treebank annotation for various languages took place at different institutes[6]. These treebanks are manually annotated and span over various domains, like that of newswire articles, conversational data, agriculture, entertainment, tourism and education, thus making our models trained on them robust. The treebanks are annotated systematically with part of speech (POS) tags, morphological features (such as root, lexical category, gender, number, person, case, vibhakti, TAM (tense, aspect and modality) label in case of verbs, or postposition in case of nouns), chunking information and syntactico-semantic dependency relations. There has been a shift from the Anncorra POS tags [11] that were initially used for Indian Languages to the new common tagset for all Indian languages which we would refer to as the Bureau of Indian Standards (BIS) tagset [27]. This new POS tagging scheme is finer than the previous scheme. The dependency relations are marked following the Computational Paninian Grammar [5, 4]. Partial corpus of all the languages containing 25,000 tokens has been released publicly in ICON 2017 [7], the rest is still being annotated with multi layered information and sanity-checked. The Telugu treebank data corresponding to BIS tagset is still being built so we used the data from ICON10 parsing contest [50]. It was cleaned and appended with some more sentences. We automatically converted this data from Anncorra tagset to BIS tagset against some word lists and rules. Since 149 sentences are lost in automatic conversion we report results on both the datasets. The statistics of the treebank data in this work can be found in the Table 5.1. Previous work has been done to convert the Hindi Treebank to Universal Dependencies (UD) [79]. These new treebanks which are built on the same underlying principle, could also be converted to UD by the same process as a future work.

5.4 Preprocessing

The process of treebank annotation for various languages took place at different institutes[8]. Owing to which there were many inconsistencies in the scheme followed. We wrote scripts to automatically make them consistent by having same label names and preservation of granularity all across. We removed erroneous sentences which gave rise to forests or cycles. Only proper acyclic trees were retained. Instances of sentences having multiple roots and unmarked dependencies were removed. Morphological analysis is the most time consuming part in the pipeline

[6]The organizations involved in this project are Jadavpur University-Kolkata (Bengali), MIT-Manipal (Kannada), C-DIT,Trivandrum (Malayalam), IIT-Bombay (Marathi), IIIT-Hyderabad (Hindi)

[7](http://kcis.iiit.ac.in/LT)

[8]The organizations involved in this project are Jadavpur University-Kolkata (Bengali), MIT-Manipal (Kannada), C-DIT,Trivandrum (Malayalam), IIT-Bombay (Marathi), IIIT-Hyderabad (Hindi).

	Types	Tokens	Chunks	Sentences	Avg. tokens / per sentence
Kannada	36778	188040	143400	16551	11.36
Malayalam	20107	65996	54818	5824	11.33
Telugu BIS	4079	11338	8203	2173	5.21
Telugu Ann.	4582	13477	8363	2322	5.80
Bengali	18172	87321	69458	8209	10.64
Marathi	24792	94844	69214	7983	11.88

Table 5.1: Treebank statistics for the 5 languages used in the experiments

of treebanking both for annotators and automatic extraction. 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). Many a times, one or many of these 8 fields were missing or noisy, requiring manual intervention for correction. The developemnt of these treebanks is however an ongoing project and we proposed the corrections that must be made in the data. The Telugu treebank data corresponding to BIS tagset is still being built so we used the data from ICON10 parsing contest [50]. It was cleaned and appended with some more sentences. We automatically converted this data from Anncorra tagset to BIS tagset against some word lists and rules. Since a lot of noise must have been introduced we report results on both the datasets.

5.5 Getting the best Features

We first describe the rationale behind choosing each feature, why it is important for each language and report a series of experiments by adding them one by one to observe their effects. It is a known fact that language specific features play a crucial role in robust dependency parsing, but their generation may require expensive tools.

5.5.1 Part of Speech Tags

POS tags are very important for dependency parsing, as a purely lexical parser may lead to sparseness but adding POS tags provides a coarser grammatical category. This generalization of words help as words belonging to the same part-of-speech are expected to have the same syntactic behavior. [64] have shown in their delexicalised parser that most of the information is captured in POS tags and just using them as features provides high unlabeled attachment score

(UAS). However, for labeled dependency parsing, especially for semantic-oriented dependencies like Paninian dependencies these non-lexical features are not predictive enough.

5.5.2 Word

It is an indispensable unit for labeled dependency parsing. It is important for resolving ambiguous relationships for dependency parsing. But lexical units are sparse and difficult to learn given a limited training data set. This sparsity is observed more in morphologically rich languages. For instance in Indian langusges, based on the grammatical information the words carry they take various lexical forms. In case if nouns, they reflect for case and number, while verbs inflect for TAM and agree in gender, number, person with of one of its arguments. Such multiple forms of words increase the vocabulary size of the language, which causes problem of data sparsity.

5.5.3 Vibhakti (Suffix and Postpositions)

In a relatively fixed word order language like English the position of a word or phrase relative to the verbal head, gives cues for grammatical relations. On the other hand free word order and morphologically rich languages change the morphological form of the dependent word, the head word, or both in order to represent grammatical relations. This information about grammatical relations thus remains available irrespective of the position of words. The morphemes (suffixes) in Dravidian languages explicitly represent grammatical and semantic relations in a sentence. This is in contrast to Indo-Aryan languages where case marking can also be expressed lexically as postpositions to establish relations between nominals and verbal predicates, the degree of which depends on their varying morphological richness. Hindi and Urdu are relatively sparse in morphology when compared to Bengali, which in turn is less rich than Marathi. These units called vibhakti that exhibit case marking are important surface cues that help identify various dependency relations. Also are important the units that mark Tense, Aspect, Modality (TAM) of a verb. There exists a direct mapping between many TAM labels and the nominal case markers because TAMs control the case markers of some nominals. Different languages tend to encode syntactically relevant information in different ways. It has been shown in previous works for Hindi[3] that the integration of morphological and syntactic information boosts the accuracy for treebanks that are syntacto-semantic in nature. We experiment to see the extent to which it helps the other Indian languages.

5.5.4 Chunk Tag

Previous work on Hindi [2] has shown that considerable improvement in parsing could be achieved using the local morphosyntactic features like chunk tags. In analytical languages,

where information about finiteness or non finiteness of verbs is not captured in the chunk head alone but is also indicated by postpositions and auxiliaries following the head, the different chunk level tags[9]can help the parser identify different syntactic behavior of these verbs. For example a finite verb can become the root of the sentence, whereas a non-finite or infinitival verb cannot. [2] used a coarser POS tag scheme so the improvement observed on addition of chunk was major. But in the new tagset that we are using, the finiteness information for verbs is marked at the POS level too. Therefore we experiment to see how far the chunk information helps us in this setting.

5.5.5 Gender, Number, Person

We want to capture the agreement between verb and its arguments in all languages by the addition of other morphological features such as gender, number and person (GNP) for each node. The verb agrees in GNP with the highest available karaka `k1` usually. But agreement rules can be complex, it may sometimes take default feature or agree with karaka `k2` in some cases. The problem worsens when there is a complex verb. Similar problems with agreement features have also been noted by [43]. So we experiment to see if the parser can learn selective agreement pattern for different languages.

Kannada and Malayalam have a three gender system - gender marking is based on semantics. Human males and females are masculine and feminine gender respectively, whereas all things and animals are neuter gender. Telugu also has a three-gender system but human females are grouped with neuter nouns in singular, and human males in plural. The verb in Malayalam is not marked for number, gender person. Similarly in Bengali, the verb changes according to the person information only, it exhibits no grammatical gender phenomena at all. Marathi also has a three gender system - masculine, feminine and neuter.

As marked in the treebanks, number can be singular or plural. Person can be be 1st person, 2nd person or 3rd person.

5.6 Experimental Setup

In our experiments, we focus on establishing dependency relations between the chunk heads which we henceforth denote as inter-chunk parsing. The relations between the tokens of a chunk (intra-chunk dependencies) are not considered for experimentation as they can easily be predicted automatically using a finite set of rules [57]. Moreover we also observed the high learnability of intra-chunk relations from an initial experiment. We found the accuracies of intra-chunk dependencies to be more than 99.00% for both Labeled Attachment and Unlabeled Attachment. The treebanks available to us are in the SSF format [10]. We use in house built

[9]finite, non-finite, infinitival and gerundial [11]

tool to convert from SSF to CoNLL format. This tool uses head and vibhakti computation tools as its dependencies. The head computation tool finds the head of a chunk based on certain rules written using POS tag information of nodes. The vibhakti computation module is again a simple, rule based tool that uses POS tag information to decide whether a lexical unit qualifies as a postposition or not. It then augments the head of the chunk with its postpositional features in the SSF format. Our parser uses data in the converted CoNLL format.

We use the arc-eager parsing model for parsing sentences containing projective arcs only, discarding the non-projective sentences. The data set is split in the ratio of 80-10-10 for training, testing and tuning the parsing model. Baseline for parsing is set using a delexicalised model having only POS tags as features. We explore with different feature sets by adding features like words, suffix, chunk tags and GNP information one by one. These features are represented as described below. In order to parse in more realistic settings, we also show parsing results using predicted POS and chunk tags obtained from the models discussed below. We report auto accuracy of the parsing model on the same training, development and testing sets that are used for parsing with gold tags.

5.6.1 Parsing Model

We have used a non-linear neural network based greedy transition-based parser [70] that is similar in structure to [26]. We use an arc-eager transition system [67]. The arc-eager system defines a set of configurations for a sentence w_1 ,...,w_n where each configuration C = (S, B, A) consists of a stack S, a buffer B, and a set of dependency arcs A. For each sentence, the parser starts with an initial configuration where S = [ROOT], B = [w_1 ,...,w_n] and A = ϕ and terminates with a configuration C if the buffer is empty and the stack contains the ROOT. The parse trees derived from transition sequences are given by A. The arc-eager system defines four types of transitions (t): 1) Shift, 2) Left-Arc, 3) Right-Arc, and 4) Reduce to derive the parsed tree. A non-linear standard feed-forward neural network is used for the oracle to predict the transitions for the parser configurations. The network uses a single layer of hidden units. RelU activation function is applied over these hidden units. In the output layer for probabilistic multi-class classification, softmax function is used. The model is trained by minimizing cross entropy loss with an l2-regularization over the entire training data. We also use mini-batch Adagrad for optimization [35] and apply dropout [47]. The parameters like number of iterations, learning rate, embedding size are tuned on the development set.

From each parser configuration, we extract features related to the top four nodes in the stack, top four nodes in the buffer and leftmost and rightmost children of the top two nodes in the stack and the leftmost child of the top node in the buffer. In addition to POS tags, dependency arc labels and sentence tokens used by Chen and Manning, we have introduced a few new features in the input layer of the model as discussed above.

5.6.2 Part of Speech Tagging and Chunking Model

We trained POS taggers and Chunkers for all the five languages using a similar neural network architecture like parsing, discussed above. Second order structural features in the form of lexical and non-lexical units were used. The input layer consisted of the current word, words in the context size of 2 surrounding the current word and the last four characters of all these words. Intra-word information is extremely useful when dealing with morphologically rich languages as word internal features contribute more context than word external features while predicting POS and chunk tags.

Using POS tags as feature has obvious benefits for chunking. At least chunk tags can be deterministically predicted if the POS tags are known. But a chunking model using auto POS tags gives less accuracy than a sans POS model. For example in Kannada, using gold POS tags in chunker gave an accuracy of 99.46%, sans POS model gave 95.25% but model having auto POS tags reduced it to 95%. So we stuck to using only lexical and suffix features while chunking.

5.6.3 Representation of Lexical Units

In our non-linear parsing model, we use distributed representation of lexical features. Using distributed representation, units of words are projected to a low dimensional continuous vector space. Unlike sparse representation in linear models, these word embeddings allow words that are closer in the embedding space to share the model parameters, thus providing an efficient solution to the problem of data sparsity. Moreover since word embeddings are assumed to capture semantic and syntactic aspects of a word, they can also improve the correlation between words and dependency labels. The same representations are also used in the POS tagger.

The monolingual corpora of all the languages are used to learn their respective word embeddings. The data is collected from various sources such as Wikipedia dump[10], ILCI - health, tourism agriculture and entertainment data [51], raw corpus from EMILLE / CIIL [90], LCC [44], part of Opensubtitles corpus [81], to train rich domain independent word-embeddings so that our parsing model is not biased. We use the Skip-gram model with negative sampling implemented in the open-source `word2vec` toolkit [66] to learn word representations. The context window size was kept to 1, as shorter context captures more syntactic relatedness compared to longer contexts that capture semantic and topical similarity. The word embedding size was experimented with and embeddings of dimension 64 gave the best results.

[10] `https://dumps.wikimedia.org`

5.6.4 Representation of POS, Chunk and GNP Tags

POS tags are small in number, but show semantic similarity like words. We use distributed representations for POS tags also by projecting them to a continuous low dimensional vector space. Similar settings as the above word embedding mode were used, while keeping the embeddings' dimension to be 20. The model for each language was trained on ILCI POS tagged data and treebank data that we were already using. The words were replaced by their corresponding tags to form a sequence. To represent chunk tags and GNP information, we use randomly initialized embeddings in the range of -0.25 to +0.25. The dimension of input vectors are taken to be 5.

In a real time setting, GNP information cannot be learnt from unlabeled monolingual data but require the presence of a morphological analyzer. It is an expensive tool to build. Due to the unavailability of a decently accurate tool for these resource poor languages, we have used gold tags in all our experiments just to observe their influence on parsing.

5.6.5 Representation of Vibhakti (Suffix and Postpositions)

Morphologically rich languages like Dravidian Languages, are highly agglutinative. The same root words inflect to have many word forms with different suffixes and prefixes. These morphemes denote the grammatical relation between a word and its arguments and may also represent TAM. This poses a problem to efficiently learn word embeddings for them. Most word embedding models consider word as a basic independent entity without considering its internal structure and shape. No explicit relationship among morphologically related words are captured too. While some work has been done to learn character based embeddings using deep neural networks for specific tasks like POS tagging, learning language models, learning word similarity etc, they are a different end to end architecture in themselves and cannot be used in integration with our parsing model. Therefore we thought it might be a good idea to treat suffixes - the last 4 characters of a word as separate units and learn embedding for them using `word2vec` to capture the linguistic regularity. This provides a potential solution for estimating rare and complex words rather than representing them in a crude way using only one or a few vectors. Instead of using the last few characters we could have used the case and TAM information present in the treebank in the form of linguistic morphemes for each word, but due to the absence of a decent or no morphological analyzer for these languages, these features would not have been available for real time parsing of development and test set. Moreover since there are more than one morpheme in a word, methods to jointly learn word and character embeddings and composing them to yield a single representation [18], need to be explored for these languages.

For Indo-Aryan langauges the degree of case and TAM marking being a part of word morphology varies according to the morphological richness of the language. This information can

also be expressed lexically as postpositions or as auxiliaries in contrast to the Dravidian languages. Since we experiment on inter-chunk parsing and establish relations between heads of chunks, this information is lost. So we compose a vector by averaging the representations (that are looked up from the `word2vec` embedding model described above) of these postpositions and auxiliaries present in a chunk, and use it as a feature.

5.7 Results

The results of experimenting with the features described in Section 4 for all the 5 languages are presented in Table 5.3 and Table 5.4. The metrics used for evaluation are Unlabeled and labeled attachment score (UAS and LAS) and label accuracy (LA). The performance corresponding to the highest performing feature set has been highlighted.

The tags in our treebanks are syntactico-semantic and it has been observed with other treebanks that learning such tags is difficult (Nivre et al., 2007a). Despite that we achieve decent LAS for all 5 languages. We also experiment with a coarser scheme of POS tags for Telugu to see the effect of the granularity of POS tag on dependency parsing. Since some 149 sentences are lost in automatic conversion from coarser to finer treebank representation for Telugu, we cannot directly compare their parsing performance but can still get an idea that the coarser scheme is better in predicting LAS. This was not so intuitive as the richer information encoded in finer POS tagset should have helped the parser disambiguate dependency relations. Label wise dependency relation analysis, taking into account the granularity of the POS tags can be explored as future work. Our delexicalised parser using only POS tags (f1) achieves good results for unlabeled parsing for all languages and serves as a good baseline. However it gives poor results for LA and in turn for LAS as was expected, lowest LAS and UAS being for Marathi. On addition of suffix features to POS tags (f2) LAS shows a substantial increase for all languages, for an example +21.37% for Kannada gold test set. Though the highest increase is for Marathi as its baseline is very poor and even the partial lexical information gives the parser a major boost. The lowest increase of +9.1% is in Telugu gold test set. Different Dravidian languages show different levels of sophistication in case marking encoded in their suffixes. While in Kannada a lot of information for case marking is encoded in suffix so adding full lexical information (f3) to the baseline delexicalised parser does not increase accuracy a lot in comparison to f2. In Malayalam f2 that is partial lexical information (suffix and POS tags) perform better than POS and full lexical information (words).

We see that addition of suffix embeddings to word and POS tags (f4), acts as a complementary feature and shows substantial increase for Kannada and Malayalam, whereas quite less for Telugu. Bengali parser however does not show much increase as it is an Indo-Aryan language. Marathi shows a considerable increase despite being an Indo-Aryan language as it is morphologically richer and behaves likes pseudo Dravidian. Geographically it is also the southernmost

Indo-Aryan language and shows syntactic convergence with the neighboring Dravidian language family. Similarly adding postposition information (f5) benefits Bengali parser considerably as compared to other Dravidian languages and Marathi.

For all our experiments we perform inter chunk parsing. The inter chunk part of the data contains only dependency relations between chunk heads of the sentences. The information of relations between the tokens of a chunk and how they interact with chunk head is lost. It is noticed that adding chunk tag information (f6) helps across all languages, specially in LA as was conjectured. However the increase is slightly more for Indo-Aryan compared to Dravidian as in the latter the the average number of words in a chunk is less owing to the agglutinative nature of the languages. The head word and its morphemes encode most of the information for finite or non finiteness of verbs and case markers and is available to us in inter-chunk parsing. While in Bengali and Marathi the information marked by verb auxiliaries and postpositions supporting the head word in a chunk are lost, so the additional chunk information helps to disambiguate between the root and non root verb in complex constructions.

Next we see the effect of GNP information (f7) on parsing accuracies. There is an increase in all languages except Malayalam. It is reasonable as there is no agreement between Malayalam verbs and their arguments. However it increases for Malayalam in the auto development and auto test set. It could be due to inconsistencies within the data. GNP marking is also very noisy for Marathi data, may be it could be looked into for validation. We could not report results for Bengali for this feature as the data is not marked for morphological information. Thus we see this feature does not add a lot of value to the increase in accuracy for these languages. Computation of GNP feature is also quite expensive and must be extracted by building a morphological analyser for each language. As a part of future work, ways may be explored on efficient representation of these features in the non linear framework as they carry important syntactic cues. This would give a better understanding of the importance of these features for the languages in concern.

We have also reported the performance of our POS tagger and chunker for all 5 languages in Table 5.2. With very simple features it gives better or comparable results for all languages compared to Bengali [41, 1], Malayalam [85]. Our results on Telugu and Bengali parsing or POS tagging cannot be compared directly to the previous works as we used a different dataset with a finer POS tagging scheme. Numerically it is still better than their results, it could be owed to the increase in size of the dataset, the architecture of our neural network models and dense representation of features.

Thus we show empirically that the presented feature set is useful for a range of morphologically rich languages across different language families, however some features are more important to certain languages than others. We have shown the essential features for each language by performing various experiments. Thus according to the need and availability of resources some features which do not boost parsing accuracy a lot for a specific language, can be pruned.

5.8 Summary

We have presented our work to adapt an existing neural network parser to suit the particularities for 5 Indian languages Kannada, Malayalam, Telugu, Bengali and Marathi belonging to two major language families Dravidian and Indo-Aryan. We proposed a unified strategy for all languages for the inclusion of rich-morphosyntactic cues in the existing parsing framework. The cost effective representation of the linguistically motivated features such as suffix, postposition, chunk and GNP aim to capture the linguistic intricacies of all languages. A detailed discussion of the rationale behind each feature and their effect on parsing accuracy was presented. Our results provided the comparison that suffix information is more useful for parsing Dravidian languages while postposition is for Indo-Aryan languages, with the exception of Marathi. We showed the performance of our parser in real time settings by using auto POS and chunk tag. In turn we also built POS taggers and chunkers for these resource poor languages. Through our work we aimed to open avenues for further research in dependency parsing for these underrepresented languages. As a future work we propose to build cross-lingual parsers for these languages by exploiting the topological and genetic similarities among them. Since Indian languages are morphologically very rich, ways of learning character-aware POS tagging and dependency parsing models could also be explored.

		Kannada	Malayalam	Telugu Bis	Telugu Ann.	Bengali	Marathi
chunk	Dev	95.23	96.59	93.73	91.89	94.26	94.42
	Test	95.25	96.74	91.28	93.17	94.25	94.93
pos	Dev	92.85	93.06	83.76	90.29	89.74	91.49
	Test	92.31	92.78	83.31	88.81	89.34	91.83

Table 5.2: Accuracy of Chunker and POS Models for Kannada, Malayalam, Bengali, Marathi, Telugu Bis and Anncorra (Ann.) Development (Dev) Set and Test Set.

Feat.	Gold						Auto					
	Development			Test			Development			Test		
	LAS	**UAS**	**LA**	**LAS**	**UAS**	**LA**	**LAS**	**UAS**	**LA**	**LAS**	**UAS**	**LA**
Kannada												
f1	54.95	79.04	56.92	55.62	80.61	57.28	52.94	77.4	55.31	53.6	78.9	55.67
f2	75.82	90.8	78.58	76.99	92.29	79.18	73.15	88.71	76.76	73.89	89.88	76.99
f3	76.01	90.06	78.63	77.16	91.32	79.41	73.36	88.02	76.87	74.27	89.2	77.26
f4	79.46	91.54	82.37	80.74	92.94	83.03	76.63	89.36	80.42	77.53	90.76	80.71
f5	79.5	91.76	82.46	80.89	92.99	83.39	76.88	89.68	80.73	77.67	90.74	81.21
f6	**79.61**	**91.48**	**82.64**	80.68	93.07	83.27	76.92	89.95	80.79	77.59	90.98	81.02
f7	79.52	91.62	82.5	**80.99**	**93.26**	**83.47**	**77.01**	**89.83**	**80.82**	**78.07**	**91.03**	**81.45**
Malayalam												
f1	50.36	77.35	54.66	49.08	76.17	53.79	48.43	75.63	53.12	47.29	74.42	52.11
f2	65.15	84.29	70.57	66.69	85.94	71.34	62.06	82.29	68.21	64.18	83.8	69.66
f3	61.95	82.93	66.43	61.52	82.92	66.08	59.46	81.19	64.24	59.67	81.58	64.44
f4	67.88	85.41	72.88	68.5	85.99	73.47	64.66	83.35	70.5	65.91	84.37	71.12
f5	68.17	84.58	73.39	**70.76**	**86.29**	**75.64**	65.21	82.68	71.2	**68.12**	**84.58**	**73.36**
f6	**68.94**	**85.31**	**73.83**	70.02	86.5	74.79	65.28	83.03	70.88	67.44	84.65	72.81
f7	68.38	84.76	73.59	69.89	86.09	74.96	**65.78**	**83.32**	**71.42**	67.57	84.94	73.3
Telugu (Anncorra)												
f1	57.37	87.84	58.85	54.53	85.16	56.54	55.28	86.24	57.13	52.89	83.51	55.24
f2	69.04	92.63	70.02	66.67	93.17	67.49	68.3	91.65	69.16	65.72	92.46	67.02
f3	74.2	94.1	75.43	70.67	92.93	71.85	72.73	93.37	74.32	69.85	92.11	71.26
f4	74.69	93.98	75.92	**73.14**	**94.11**	**74.32**	74.03	93.73	75.68	**71.61**	**93.29**	**73.14**
f5	74.82	93.61	76.29	72.44	94.11	73.5	74.2	93.73	75.92	70.44	92.58	72.08
f6	**75.43**	**94.84**	**76.78**	71.73	93.05	72.91	**74.45**	**93.37**	**76.17**	70.55	93.05	71.97
f7	75.31	94.59	76.29	72.79	93.76	73.97	72.97	92.75	74.57	70.91	92.82	72.2

Table 5.3: Parsing accuracies of our neural network based parser. Auto development and test set contain predicted POS and chunk tags. Gloss of the features are f1 = POS only, f2 = f1+ suffix, f3 = POS + word, f4 = f3 + suffix , f5 = f4+ PSP, f6 = f5 + chunk, f7 = f6 + GNP

Feat.	Gold						Auto					
	Development			Test			Development			Test		
	LAS	**UAS**	**LA**	**LAS**	**UAS**	**LA**	**LAS**	**UAS**	**LA**	**LAS**	**UAS**	**LA**
Telugu (BIS)												
f1	54.73	90.03	55.63	56.26	87.61	57.65	53.45	89.0	55.12	55.37	87.23	57.14
f2	66.11	93.09	67.65	65.36	92.04	66.88	63.55	91.82	65.86	65.23	91.66	66.75
f3	69.95	93.48	71.61	69.15	91.91	70.54	69.31	93.09	70.97	67.64	91.28	69.28
f4	70.72	93.09	72.76	69.28	91.4	71.3	70.72	92.97	72.89	68.72	91.15	70.54
f5	72.25	93.99	73.66	69.28	91.66	71.3	71.1	93.73	72.63	69.15	91.66	71.18
f6	**73.53**	**94.63**	**74.81**	71.55	93.17	72.95	**72.38**	**93.99**	**73.91**	**69.91**	**92.16**	**71.93**
f7	72.89	94.88	74.17	**71.93**	**92.92**	**73.58**	71.61	93.86	73.53	68.35	90.39	70.67
Marathi												
f1	34.81	59.92	39.29	34.06	59.11	38.5	34.83	60.24	39.1	33.45	58.63	38.24
f2	64.25	83.52	68.79	62.57	81.15	67.38	63.96	83.38	68.44	61.98	80.74	66.94
f3	66.27	84.6	69.67	65.22	83.66	69.2	66.08	84.58	69.45	65.11	83.41	69.14
f4	70.33	86.99	74.1	68.12	84.33	72.69	70.39	87.04	74.04	68.07	84.48	72.61
f5	70.47	87.32	74.26	68.42	85.18	72.44	70.25	87.19	74.15	68.0	84.92	72.07
f6	71.01	87.72	74.75	69.56	86.28	73.42	70.46	87.5	74.18	68.45	86.06	72.3
f7	**71.56**	**88.05**	**74.95**	**69.75**	**86.41**	**73.52**	**70.97**	**87.69**	**74.47**	**69.01**	**85.98**	**72.82**
Bengali												
f1	52.71	78.08	55.22	52.52	78.7	54.93	49.33	74.65	52.82	48.34	74.89	51.63
f2	68.19	85.37	70.82	67.6	84.86	70.68	64.42	82.1	68.26	63.61	81.75	67.38
f3	71.54	85.43	74.51	70.45	85.23	73.29	68.76	83.09	72.48	66.96	82.22	70.8
f4	72.86	85.81	76.07	71.66	86.26	74.55	69.55	83.22	73.55	68.5	83.62	72.69
f5	75.82	87.6	79.05	74.66	87.27	78.22	73.26	85.72	77.21	**72.65**	**85.86**	**76.55**
f6	**76.43**	**88.41**	**79.67**	**75.64**	**88.41**	**78.63**	**73.28**	**86.08**	**77.29**	72.24	85.99	75.92
f7	-	-	-	-	-	-	-	-	-	-	-	-

Table 5.4: Parsing accuracies of our neural network based parser. Auto development and test set contain predicted POS and chunk tags. Gloss of the features are f1 = POS only, f2 = f1+ suffix, f3 = POS + word, f4 = f3 + suffix , f5 = f4+ PSP, f6 = f5 + chunk, f7 = f6 + GNP

Chapter *6*

Conclusions

Indian languages are morphologically rich and free word order. Dependency parsing is challenging for such languages. Majority of the Indian languages are underrepresented. They lack annotated resources for developing NLP tools for basic processing of text like morph analyser, POS tagger and syntactic parsers. However substantial work has been done to develop state of the art tools for Hindi over the last decade. This was facilitated by the development of rich annotated resources - majorly the Hindi Dependency Treebank. However the Computational Paninian formalism followed to represent dependency relations in this resource is very fine grained. A majority of downstream applications like Machine Translation, Question Answering may not need such high levels of differentiation in the predicate-argument relations. Through our work we explore the effect of reducing granularity of the hierarchical structure of relations to see how it impacts parsing accuracy and whether underspcifying them leads to crucial loss of information. We further convert this important resource to an internationally accept scheme of annotation - Universal Dependencies, making it useful as a resource for multilingual language technology. We discuss the differences in annotation in both the schemes, present parsing experiments for both the formalisms and empirically evaluate their weaknesses and strengths for Hindi. The conversion was a challenging task and there are constructions which are yet to be addressed to be fully compliant with the UD scheme like that of ellipsis etc. As a part of future work, we propose to come up with better techniques to resolve empty nodes in the absence of predicational or verbal heads. Also a few attachment schemes must be reanalyzed and revised to handle long distance dependencies efficiently.

We continue our efforts to extend the benefits of NLP advancements to other resource poor languages belonging to major language families - Indo Aryan and Dravidian. We have adapted an existing neural network parser to suit the particularities for 5 Indian languages Kannada, Malayalam, Telugu, Bengali and Marathi. We discuss how the traditional linear parsing scheme using sparse representations posed problems. We show how the shift to dense distributed representations used in a non linear architecture solves the problem of sparsity, incompleteness and expensive feature extraction. We proposed a unified strategy for all languages for the

inclusion of rich-morphosyntactic cues in the neural network framework. The cost effective representation of the linguistically motivated features such as suffix, postposition, chunk and GNP aim to capture the linguistic intricacies of all languages from the two families. A rich set of syntactic features were already being used earlier but they relied on manual annotation and expensive automatic tools. In our scheme of things we propose to extract them in a cost effective manner using vector space modelling. Using distributed representations also help in making the parser robust and domain unbound. This is a first attempt where rich linguistic features are being used in a neural network framework. We report our parsing accuracies for auto POS, chunk too. As an auxiliary contribution neuual network based POS taggers and chunkers was also delivered.

We hope our work provides direction for developing of more sophisticated tools for these resource poor languages. Character aware parsing models can be builts using techniques like Birdirectional LSTM to capture the morpholofical features of these languages more elegantly and to handle the problem of lexical sparsity. As a part of future work cross lingual parsers can be built exploiting the typological simialrities among various Indian languages.

Related Publications

- Conversion from Paninian Karakas to Universal Dependencies for Hindi Dependency Treebank, **Juhi Tandon**, Himani Chaudhry, Riyaz Ahmad Bhat and Dipti Misra Sharma, *In Proceedings of LAW X – The 10th Linguistic Annotation Workshop, pages 141–150, Berlin, Germany, August 11, 2016.*
- Unity in Diversity: A unified parsing strategy for major Indian languages, **Juhi Tandon** and Dipti Misra Sharma, *In Proceedings of the Fourth International Conference on Dependency Linguistics, pages 255-265,Pisa, Italy, September 18-20 2017.*

Bibliography

[1] F. Alam, S. A. Chowdhury, and S. R. H. Noori. Bidirectional lstms—crfs networks for bangla pos tagging. In *Computer and Information Technology (ICCIT), 2016 19th International Conference on*, pages 377–382. IEEE, 2016.

[2] 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.

[3] B. R. Ambati, S. Husain, J. Nivre, and R. Sangal. On the role of morphosyntactic features in hindi dependency parsing. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 94–102. Association for Computational Linguistics, 2010.

[4] 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.

[5] A. Bharati, V. Chaitanya, R. Sangal, and K. Ramakrishnamacharyulu. *Natural Language Processing: A Paninian Perspective*. Prentice-Hall of India, 1995.

[6] A. Bharati, D. S. S. Husain, L. Bai, R. Begam, and R. Sangal. Anncorra: Treebanks for indian languages, guidelines for annotating hindi treebank (version–2.0), 2009.

[7] 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.

[8] A. Bharati, S. Husain, D. Misra, and R. Sangal. Two stage constraint based hybrid approach to free word order language dependency parsing. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 77–80. Association for Computational Linguistics, 2009.

[9] A. Bharati, S. Husain, D. M. Sharma, and R. Sangal. A two-stage constraint based dependency parser for free word order languages. In *Proceedings of the COLIPS International Conference on Asian Language Processing 2008 (IALP)*, 2008.

[10] A. Bharati, R. Sangal, and D. M. Sharma. Ssf: Shakti standard format guide. 2007.

[11] A. Bharati, R. Sangal, D. M. Sharma, and L. Bai. Anncorra: Annotating corpora guidelines for pos and chunk annotation for indian languages. *LTRC-TR31*, 2006.

[12] R. A. Bhat. Exploiting linguistic knowledge to address representation and sparsity issues in dependency parsing of indian languages. 2017.

[13] R. A. Bhat, I. A. Bhat, N. Jain, and D. M. Sharma. A house united: Bridging the script and lexical barrier between hindi and urdu. In *International Conference on Computational Linguistics (COLING 2016)*, 2016.

[14] R. A. Bhat, I. A. Bhat, and D. M. Sharma. Improving transition-based dependency parsing of hindi and urdu by modeling syntactically relevant phenomena. *ACM Transactions on Asian and Low-Resource Language Information Processing (TALIP)*, 2016.

[15] R. A. Bhat, R. Bhatt, A. Farudi, P. Klassen, B. Narasimhan, M. Palmer, O. Rambow, D. M. Sharma, A. Vaidya, S. R. Vishnu, et al. The hindi/urdu treebank project. In *Handbook of Linguistic Annotation*. Springer Press, 2014.

[16] 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.

[17] B. Bohnet. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd international conference on computational linguistics*, pages 89–97. Association for Computational Linguistics, 2010.

[18] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[19] C. Bosco, S. Montemagni, and M. Simi. Converting italian treebanks: Towards an italian stanford dependency treebank. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 61–69. Citeseer, 2013.

[20] J. Bresnan, A. Asudeh, I. Toivonen, and S. Wechsler. *Lexical-functional syntax*, volume 16. John Wiley & Sons, 2015.

[21] S. Buchholz and E. Marsi. Conll-x shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 149–164. Association for Computational Linguistics, 2006.

[22] J. Carroll, T. Briscoe, and A. Sanfilippo. Parser evaluation: a survey and a new proposal. In *Proceedings of the 1st International Conference on Language Resources and Evaluation*, pages 447–454, 1998.

[23] 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.

[24] P.-C. Chang, H. Tseng, D. Jurafsky, and C. D. Manning. Discriminative reordering with chinese grammatical relations features. In *Proceedings of the Third Workshop on Syntax and Structure in Statistical Translation*, pages 51–59. Association for Computational Linguistics, 2009.

[25] H. Chaudhry and D. M. Sharma. Annotation and issues in building an english dependency treebank. 2011.

[26] D. Chen and C. D. Manning. A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750, 2014.

[27] N. Choudhary and G. N. Jha. Creating multilingual parallel corpora in indian languages. In *Language and Technology Conference*, pages 527–537. Springer, 2011.

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

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

[30] M. de Lhoneux, S. Stymne, and J. Nivre. Old school vs. new school: Comparing transition-based parsers with and without neural network enhancement. In *TLT*, pages 99–110, 2017.

[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 *LREC*, volume 14, 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. Genoa Italy, 2006.

[33] M.-C. De Marneffe and C. D. Manning. Stanford typed dependencies manual. Technical report, Technical report, Stanford University, 2008.

[34] J. Devlin, R. Zbib, Z. Huang, T. Lamar, R. M. Schwartz, and J. Makhoul. Fast and robust neural network joint models for statistical machine translation. In *ACL (1)*, pages 1370–1380. Citeseer, 2014.

[35] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[36] C. Dyer, M. Ballesteros, W. Ling, A. Matthews, and N. A. Smith. Transition-based dependency parsing with stack long short-term memory. *arXiv preprint arXiv:1505.08075*, 2015.

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

[38] 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.

[39] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.

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

[41] A. R. Ghosh. *Memory Based Learner for Bengali POS Tagging.* PhD thesis, JADAVPUR UNIVERSITY, 2013.

[42] Y. Goldberg. A primer on neural network models for natural language processing. 2016.

[43] Y. Goldberg and M. Elhadad. Hebrew dependency parsing: Initial results. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 129–133. Association for Computational Linguistics, 2009.

[44] D. Goldhahn, T. Eckart, and U. Quasthoff. Building large monolingual dictionaries at the leipzig corpora collection: From 100 to 200 languages. In *LREC*, pages 759–765, 2012.

[45] J. Gorla, A. K. Singh, R. Sangal, K. Gali, S. Husain, and S. Venkatapathy. A graph based method for building multilingual weakly supervised dependency parsers. In *Advances in Natural Language Processing*, pages 148–159. Springer, 2008.

[46] J. Hall. Maltparser: An architecture for labeled inductive dependency parsing. *Licentiate thesis, Växjö University, Växjö, Sweden*, 2006.

[47] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[48] M. Hohensee. *It's only morpho-logical: Modeling agreement in cross-linguistic dependency parsing.* PhD thesis, 2012.

[49] M. Hohensee and E. M. Bender. Getting more from morphology in multilingual dependency parsing. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 315–326. Association for Computational Linguistics, 2012.

[50] S. Husain, P. Mannem, B. Ambati, and P. Gadde. The icon-2010 tools contest on indian language dependency parsing. *Proceedings of ICON-2010 Tools Contest on Indian Language Dependency Parsing, ICON*, 10:1–8, 2010.

[51] G. N. Jha. The tdil program and the indian langauge corpora intitiative (ilci). In *LREC*, 2010.

[52] A. Johannsen, H. M. Alonso, and B. Plank. Universal dependencies for danish. In *International Workshop on Treebanks and Linguistic Theories (TLT14)*, page 157, 2015.

[53] S. R. Kesidi. *CONSTRAINT-BASED HYBRID DEPENDENCY PARSER FOR TELUGU.* PhD thesis, International Institute of Information Technology Hyderabad, India, 2013.

[54] T. H. King, R. Crouch, S. Riezler, M. Dalrymple, and R. M. Kaplan. The parc 700 dependency bank. In *Proceedings of the EACL03: 4th international workshop on linguistically interpreted corpora (LINC-03)*, pages 1–8, 2003.

[55] E. Kiperwasser and Y. Goldberg. Simple and accurate dependency parsing using bidirectional lstm feature representations. *arXiv preprint arXiv:1603.04351*, 2016.

[56] 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.

[57] 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.

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

[59] 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.

[60] J. Lipenkova and M. Soucek. Converting russian dependency treebank to stanford typed dependencies representation. In *EACL*, pages 143–147, 2014.

[61] P. Mannem, H. Chaudhry, and A. Bharati. Insights into non-projectivity in hindi. In *Proceedings of the ACL-IJCNLP 2009 Student Research Workshop*, pages 10–17. Association for Computational Linguistics, 2009.

[62] R. McDonald, K. Crammer, and F. Pereira. Online large-margin training of dependency parsers. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 91–98. Association for Computational Linguistics, 2005.

[63] R. McDonald and J. Nivre. Analyzing and integrating dependency parsers. *Computational Linguistics*, 37(1):197–230, 2011.

[64] R. McDonald, S. Petrov, and K. Hall. Multi-source transfer of delexicalized dependency parsers. In *Proceedings of the conference on empirical methods in natural language processing*, pages 62–72. Association for Computational Linguistics, 2011.

[65] R. T. McDonald, J. Nivre, Y. Quirmbach-Brundage, Y. Goldberg, D. Das, K. Ganchev, K. B. Hall, S. Petrov, H. Zhang, O. Täckström, et al. Universal dependency annotation for multilingual parsing. In *ACL (2)*, pages 92–97. Citeseer, 2013.

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

[68] J. Nivre. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57. Association for Computational Linguistics, 2004.

[69] J. Nivre. *Inductive dependency parsing*, volume 34. Springer, 2006.

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

[71] J. Nivre, M.-C. de Marneffe, F. Ginter, Y. Goldberg, J. Hajic, C. D. Manning, R. McDonald, S. Petrov, S. Pyysalo, N. Silveira, et al. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*, 2016.

[72] 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(2):95–135, 2007.

[73] 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.

[74] 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.

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

[76] S. Pyysalo, J. Kanerva, A. Missilä, V. Laippala, and F. Ginter. Universal dependencies for finnish. In *Proceedings of NoDaLiDa*, pages 163–172, 2015.

[77] K. Singla, A. Tammewar, N. Jain, and S. Jain. Two-stage approach for hindi dependency parsing using maltparser. *Training*, 12041(268,093):22–27, 2012.

[78] R. Socher, J. Bauer, C. D. Manning, and A. Y. Ng. Parsing with compositional vector grammars. In *ACL (1)*, pages 455–465, 2013.

[79] J. Tandon, H. Chaudhary, R. A. Bhat, and D. M. Sharma. Conversion from pāṇinian kārakas to universal dependencies for hindi dependency treebank. *LAW X*, page 141, 2016.

[80] L. Tesnière. *Eléments de syntaxe structurale*. Librairie C. Klincksieck, 1959.

[81] J. Tiedemann. News from OPUS - A collection of multilingual parallel corpora with tools and interfaces. In N. Nicolov, K. Bontcheva, G. Angelova, and R. Mitkov, editors, *Recent Advances in Natural Language Processing*, volume V, pages 237–248. John Benjamins, Amsterdam/Philadelphia, Borovets, Bulgaria, 2009.

[82] R. Tsarfaty, D. Seddah, Y. Goldberg, S. Kübler, M. Candito, J. Foster, Y. Versley, I. Rehbein, and L. Tounsi. Statistical parsing of morphologically rich languages (spmrl): what, how and whither. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 1–12. Association for Computational Linguistics, 2010.

[83] 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.

[84] P. D. Turney and P. Pantel. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37:141–188, 2010.

[85] D. V V and D. M. Sharma. Significance of an accurate sandhi-splitter in shallow parsing of dravidian languages. In *Proceedings of the ACL 2016 Student Research Workshop*, pages 37–42, Berlin, Germany, August 2016. Association for Computational Linguistics.

[86] A. Vaidya, J. D. Choi, M. Palmer, and B. Narasimhan. Analysis of the hindi proposition bank using dependency structure. In *Proceedings of the 5th Linguistic Annotation Workshop*, pages 21–29. Association for Computational Linguistics, 2011.

[87] M. Wang and C. D. Manning. Effect of non-linear deep architecture in sequence labeling. In *IJCNLP*, pages 1285–1291, 2013.

[88] D. Weiss, C. Alberti, M. Collins, and S. Petrov. Structured training for neural network transition-based parsing. *arXiv preprint arXiv:1506.06158*, 2015.

[89] F. Xia, O. Rambow, R. Bhatt, M. Palmer, and D. M. Sharma. Towards a multi-representational treebank. In *The 7th International Workshop on Treebanks and Linguistic Theories. Groningen, Netherlands*, pages 159–170, 2009.

[90] R. Xiao, A. McEnery, J. Baker, and A. Hardie. Developing asian language corpora: standards and practice. In *The 4th Workshop on Asian Language Resources*, 2004.

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

[92] D. Zeman, D. Marecek, M. Popel, L. Ramasamy, J. Stepánek, Z. Zabokrtskỳ, and J. Hajic. Hamledt: To parse or not to parse? In *LREC*, pages 2735–2741, 2012.

[93] C. Zhu, X. Qiu, X. Chen, and X. Huang. A re-ranking model for dependency parser with recursive convolutional neural network. *arXiv preprint arXiv:1505.05667*, 2015.