Data-Driven Grammar

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

Master of Science
in
Computer Science and Engineering *by Research*

by

ANIL KRISHNA ERAGANI
200702013
anil.eragani@research.iiit.ac.in

International Institute of Information Technology
(Deemed to be University)
Hyderabad - 500 032, INDIA
April 2024

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled " Data-Driven Grammar " by ANIL KRISHNA ERAGANI, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Adviser: Prof. Dipti Mishra Sharma

To My Family

Acknowledgments

I would like to express my deepest gratitude to Prof. Dipti Misra Sharma and Siva Reddy for their invaluable guidance and insightful counsel throughout the development of this thesis. Their expertise and support have been fundamental to my research. Additionally, I am deeply thankful to my family members for their unwavering encouragement and support. They have continually inspired me to persevere and have been instrumental in bringing this thesis to fruition.

Abstract

In the literature several parsing systems for natural language processing exist which use different methods to parse sentences. Most of the previous work done in this domain was either done using hand written grammar or by using data-driven methods. Both these methods have pros as well as cons associated with them. As data-driven parsers rely on annotated treebanks they perform at high accuracies when they encounter familiar data that they have been trained on. The only downside to this method is that when they encounter unknown data in a sentence their performance drops drastically. On the other hand, when we come to parsers which use hand written grammars, they have a broad coverage on the language hence we could say that there is no such a thing as seen or unseen data. The only problem with this method is that it is very hard to write a grammar that has broad coverage in a language.

In this thesis we develop a method where the con of one method is overcome by the pro of the other. In short we could say that we are building a parser which uses grammar rules extracted from a treebank (which would cover the aspect of broad coverage) and also has the data-driven approach included in it.

With this thesis we try to achieve two goals:

- Developing a novel approach to parsing text.
- Being able to use the resulting parser using very minimal resources.

Contents

List of Figures

List of Tables

Chapter *1*

Introduction

Parsing is a process which involves giving structural descriptions to sentences which is one of the many important processes in NLP. The applications of parsing include everything from a very simple task such as phrase finding, e.g. proper noun recognition, to a bit complex systems like Machine Translation, Dialogue Managers, etc. The success of the above mentioned systems is greatly dependent on how best the parser used in them performs, which makes it a core component in all of the above mentioned systems, which in turn suggests the high significance given to parsing.

We usually construct the parse trees using one of the following relations:

- Constituency relation used in Phrase Structure Grammar
- Dependency relation used in Dependency Grammar

A constituency parse tree splits the input sentence into sub-phrases. There is a clear distinction between the "terminal nodes" and "non-terminal nodes" in a constituency parse tree. Here is an example which shows the phrase structure parse of a simple sentence "John sees Bill":

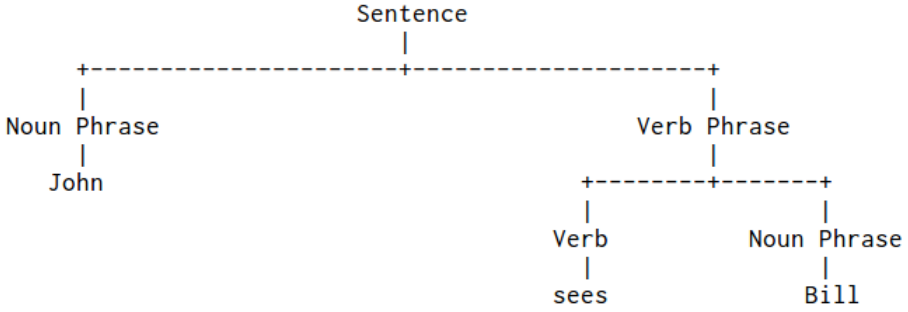


Figure 1.1

The parse tree in the above case is the entire structure, starting from the node "Sentence" and ending in each of the leaf nodes (John, sees, Bill). All the nodes in a given tree can be categorized into one of these three:

- **Root node** - "Sentence" (Non-terminal node)
- **Branch nodes** - "Noun Phrase", "Verb", "Verb Phrase" (Non-terminal nodes)
- **Leaf nodes** - "John", "sees", "Bill" (Terminal nodes)

Phrase structure parse is useful only when the languages we are dealing with have a very strict word order. Since our goal is to build a parser that is useful to all the languages including the ones which don't have a strict word order, e.g. Hindi, we would be looking into dependency parsing.

1.1 Dependendency Parsing

Dependency parsing involves using dependency relation which is used to define the relation between two words (if they are related) in a sentence. This dependency relation views the finite verb as the structural center of all the dependency structures. A "dependency parse" connects words based on the relationship with other words in the sentence. Each node in the tree will constitute a word, child nodes are those words that are "dependent" on the parent nodes, and edges (which are directed) are labelled by the "dependency relation", which defines a certain grammatical relation between the parent node and the child node. When we consider parsing the sentence "John sees Bill" using dependency grammar, we see a major difference between the structures of phrase structure parsing and dependency parsing.

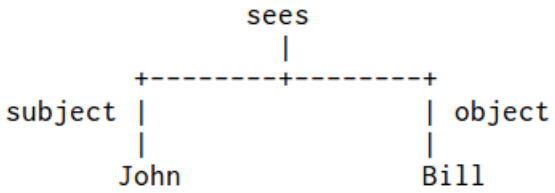


Figure 1.2

In the above structure we can see that both the terminal and non-terminal nodes are words. So there won't be any distinction between the two types of nodes as we saw in the phrase structure parse of the above example. It also has relation names written on the edges. This is different from what we saw in the structure we got using phrase structure parsing. All the non-terminal nodes in the previous structure have been removed because of which we can see a huge decrease in the size of the structure which in turn helps with storage purposes.

From the above example we can say that dependency parsing has the following advantages over phrase structure parsing:

- Dependency relations are very close to the semantic relationships. It would be really difficult to identify "head-modifier" or "head-complement" relations from a tree that are not straightforward.
- The dependency tree has as many nodes in it as there are words in the sentence. And since the parser is tasked to connect the existing nodes and not generate new ones, this process is more straightforward.
- Dependency parsing allows the use of one-word-at-a-time operation, which we saw in phrase structure parsing.

We can say that both the dependency parse and the phrase structure parse have the following properties:

1 **Single Headedness**: Each child node in both the structures has only one other node that it depends on i.e. their parent node.

2 **Connectivity**: All the nodes in the structure are connected. We always have a path from one node to any other node in the structure.

3 **Acyclic**: No matter which path we choose we will never find vertices repeating in that path. In short, there won't be any cycles present in these structures.

In syntactic parsing, a parser could function as a tool searching through all the possibilities to find an accurate parse tree for a given input sentence. The search space of the possible parse trees is defined by the grammar or a set of constraints we write for the parser. In this thesis we show how the grammar we extract and the set of constraints we define (some of which happen to be the properties explained above) are going to restrict the search space of our parser (which is explained in the later chapters).

We now discuss two approaches used for parsing, grammar-driven parsing approach and the data-driven parsing approach, in the following sections.

1.2 Grammar-Driven Parsing

In the grammar-driven approach, parsing is modeled by generating a grammar G which is used to generate the language L(G). A grammar parsing algorithm is then used to compute the analyses of a given input string. The grammar may be hand-crafted or it may be completely or partially generated from an annotated corpus. There are two types of grammar-driven parsing:

- Context free grammar parsing (Unlexicalized grammar driven parsing)
- Constraints based grammar parsing (Lexicalized grammar driven parsing)

1.2.1 Context Free Grammar Parsing

A context free grammar is a formal grammar in which every rule follows the structure "V → w" where "V" is a single non-terminal symbol and "w" is a string of terminals and/or non-terminals. Formally a context free grammar G can be represented by the 4-tuple:

"G = (T, N, S, R)" where,

- **T** - { **T**erminal symbols }
- **N** - { **N**on-terminal symbols }
- **S** - **S**tart symbol (**S** ∈ **N**, a non-terminal symbol)
- **R** - { **R**ules of the form "V → w" }

In general we assume that the language L(G) generated by this grammar G is a very close approximation of the natural language L that we would like to process. In practice, we know that most of the formal grammars that have been developed for natural languages till now do not satisfy this assumption, and many of the research directions in natural language parsing during the past few decades can be seen as motivated by the desire to overcome these problems.

A sample context free grammar for the language English is shown in the below figure:

```
G = (T, N, S, R)
T = {that, this, a, the, man, book, flight, meal, include, read, does}
N = {S, NP, NOM, VP, Det, Noun, Verb, Aux}
S = S
R = {
        S → NP VP                       Det → that | this | a | the
        S → Aux NP VP                   Noun → book | flight | meal | man
        S → VP                          Verb → book | include | read
        NP → Det NOM                    Aux → does
        NOM → Noun
        NOM → Noun NOM
        VP → Verb
        VP → Verb NP
    }
```

Figure 1.3

Since the lexicon used in this formalism is merely a list of entries, we can say that parsing using CFG is an Unlexicalized Grammar Driven Parsing. A sample sentence "The man read this book" parsed using the above grammar is shown in the figure.

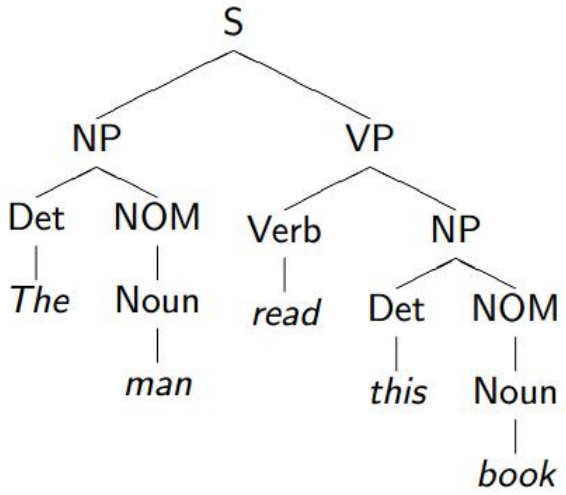


Figure 1.4

One may often find that more than one production rule can be applied to a sequence of words, which will lead to a conflict, an ambiguous parse (as shown in the above example). This grammar is unambiguous i.e. no string has two distinct parse trees. This would also mean that the language "L(G)" generated by the above grammar "G" is far from being an approximation of English (language "L"), as all the natural languages are ambiguous.

The above grammar does display an interesting and troublesome phenomenon called local ambiguity. Consider the sentence "The cops knew Mark shot John". This has a unique parse tree in this grammar. But now look at the first part of it: "The cops knew Mark". This is also considered as a sentence according to this grammar, but not when considered as a part of "The cops knew Mark shot John", which could be problematic for a parser. Locally we have something that looks like a sentence and a parser may prove that this part really is a sentence according to the grammar.

One way to deal with such ambiguous parses is to add more rules, or prioritize the existing rules so that one rule precedes the others. However, this has a huge drawback of multiplying the rules numerously, to the point where they become difficult to manage. One other difficulty we face is over generation, where unlicensed structures, structures that are meaningless, are also generated. Probabilistic grammars overcome these problems by using the frequency of various productions to order them, resulting in a "most likely" interpretation.

A probabilistic context-free grammar (PCFG, also known as stochastic CFG, SCFG) is a context-free grammar, where a certain probability is assigned to each rule. Thus, some

derivations become "more likely" than other. These probabilities are typically computed using machine learning techniques that train on very large corpora. As such, a probabilistic grammar's validity is constrained by the context of the text used to train it because of which the definition of "most likely" would change with the context of the discourse. A very popular example of a probabilistic CFG parser that has been trained using the Penn Treebank is the Stanford Statistical Parser of Klein and Manning.

Different techniques are used to parse a sentence using PCFG, one of which is a variant of CYK algorithm which finds a Viterbi parse, which is the "most likely" parse, of a sequence for a given PCFG.

The unlexicalized grammars have certain advantages that are very straightforward, namely

- Easy to estimate
- Easy to parse with
- Time and space efficient

However, the poor performance of these basic unannotated and unlexicalized grammars has made these advantages irrelevant.

1.2.2 Constraint Based Grammar Parsing

The second approach in grammar-driven parsing is based on the notion of eliminative parsing, where the sentences are analyzed by successively eliminating representations that don't satisfy the constraints until only valid representations remain. In the eliminative approach, parsing is viewed as a constraint satisfaction problem, where any analysis that satisfies all the constraints of the grammar is a valid analysis. This grammar can handle non-projective labeled parse structures along with the projective ones.

Verb valency can be defined as the number of arguments controlled by a verbal predicate. In this formalism (Constraint Based Grammar) all the verbs are classified into different groups based on how similar their "valences" are. Since the words used here are not just simple entries in the lexicon but have a proper structure associated with them, we can say that this formalism is a Lexicalized Grammar Driven Parsing.

So far, we have discussed two main trends in grammar-driven parsing. The first is the Context Free Grammar (CFG) and its variant Probabilistic Context Free Grammar (PCFG) both of which are usually restricted to projective phrase structure parses, where standard techniques from context-free parsing are used to obtain good efficiency in the presence of massive ambiguity. The second is based on a formalization of dependency grammar in terms of constraints, not necessarily limited to projective structures, where parsing is naturally viewed as a constraint satisfaction problem which can be addressed using eliminative parsing methods, although the exact parsing problem is often hard to deal with.

1.3 Data-Driven Parsing

When we come to data-driven parsing, a formal grammar is not relevant for parsing anymore. In this framework we use an annotated treebank to train the language model. Here, the mapping from the sentences to their respective parse structures is learnt by the models, which is used to parse the input sentences.

The basic data-driven parsing framework is as given below:

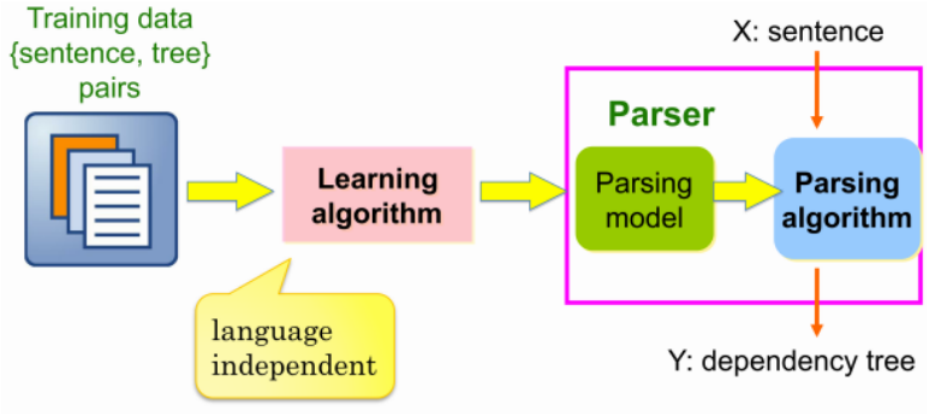


Figure 1.5

One of the major advantages of data-driven parsing approach is that the time taken to develop is shortened by a lot when compared to the systems that rely on hand-crafted resources in the form of lexicons or grammars. However, this can be achieved only if the data resources required to train or tune the data-driven system are readily available [3].

There are two approaches for modeling the data-driven parsers:

- Transition based models
- Graph based models

Malt Parser, a commonly used parser, uses transition based model. The basic idea behind a transition-based model is to define a transition system (state machine) for mapping a sentence to its graph and induce a model for predicting the next state transition, given the transition history. Given this induced model a parser then constructs an optimal transition sequence for all the input sentences, thereby giving us a parse structure as output.

We will discuss about graph based models in the next chapter where we'll analyze in depth about the related literature.

1.4 Problem Statement and Motivation

At the moment, many languages do not have annotated treebanks which can be used by data-driven parsers like Malt Parser, which leads to higher accuracies only when they encounter familiar data that they have been trained on. When we consider parsers which use hand written grammars, they have a broad coverage on the language. We could say that there is no such a thing as seen or unseen data. The only problem with this method is that it is very hard to write a grammar that has broad coverage in a language.

Our goal here is to come up with a hybrid approach which employs the pros of both the approaches mentioned above. We would be building a parser which uses grammar rules extracted from an annotated treebank. Although the state-of-the-art parsers use neural networks, linguistically informed approaches can still work.

1.5 Contribution of the thesis

Below is a list of contributions made by this thesis.

- Building a dependency grammar from an existing treebank
- A new constraint based graphical model for dependency parsing, **Data-driven Grammar** based dependency parsing model
- Improving existing parsing Techniques (Maltparser)
- By products such as word sketches, thesaurus, etc.

Chapter *2*

Review of related Literature and Research

As discussed in the previous chapter, in literature we have several statistical and graph based parsing techniques. One such method developed by Hirakawa (2001) is a graph based parsing technique that reduces the problem of dependency parsing to spanning tree search. This method uses a branch and bound algorithm that is exponential in the worst case.

The other graph based parsing technique, which we are going to discuss in detail in this chapter, is the edge-based factorization of dependency trees [12]. This is used to simplify the problem of dependency parsing to the problem of identifying maximum spanning trees in directed graphs. This method is very similar to that of our approach discussed in detail in the coming chapters.

2.1 Dependency Trees as Spanning Trees

Mcdonald et al 2005 [12] has proved that the problem of dependency parsing can be simplified to finding maximum spanning trees in directed graphs. This can be achieved by applying the edge-based factorization method to the projective languages with the Eisner parsing algorithm and non-projective languages with the Chu-Liu-Edmonds algorithm.

2.1.1 Edge Based Factorization & MST

As described in Mcdonald et al, 2005, if $\boldsymbol{x} = x_1...x_n$ is representing a generic input sentence and if $\boldsymbol{y}$ is representing a generic dependency tree for the input sentence $\boldsymbol{x}$ then we can write "$(i, j) \in \boldsymbol{y}$", if there is a dependency tree in $\boldsymbol{y}$ from word x_i to word x_j.

The score of a dependency tree is factored as the sum of the scores of all the edges in the tree. The score of an edge is the dot product between a high dimensional feature representation of the edge and a weight vector,

$$s(i, j) = \mathbf{w}.\mathbf{f}(i, j) \tag{2.1}$$

Hence, the score of a dependency tree $\boldsymbol{y}$ for a sentence $\boldsymbol{x}$ is,

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{(i,j)\in \boldsymbol{y}} s(i,j) = \sum_{(i,j)\in \boldsymbol{y}} \mathbf{w}.\mathbf{f}(i,j) \tag{2.2}$$

Assuming an appropriate feature representation **f**(i,j) and a weight vector **w**, dependency parsing is nothing but finding tree $\boldsymbol{y}$ with the highest score for a given input sentence $\boldsymbol{x}$.

In their work, a generic directed graph $G = (V,E)$ is represented by its vertex set $V = v_1, ..., v_n$ and set $E \subseteq [1:n] \times [1:n]$ of pairs (i,j) of directed edges $v_i \rightarrow v_j$, where each such edge has a score *s(i,j)*.

This being said, they use Chu-Liu-Edmonds algorithm shown in fig. 2.1 to find the maximum spanning trees (MSTs) in the graphs defined above.

Chu-Liu-Edmonds(G, s)
Graph $G = (V, E)$
Edge weight function $s : E \rightarrow \mathbb{R}$
1. Let $M = \{(x^*, x) : x \in V, x^* = \arg\max_{x'} s(x', x)\}$
2. Let $G_M = (V, M)$
3. If G_M has no cycles, then it is an MST: return G_M
4. Otherwise, find a cycle C in G_M
5. Let $G_C = \text{contract}(G, C, s)$
6. Let $\boldsymbol{y} = \text{Chu-Liu-Edmonds}(G_C, s)$
7. Find a vertex $x \in C$ s. t. $(x', x) \in \boldsymbol{y}, (x'', x) \in C$
8. return $\boldsymbol{y} \cup C - \{(x'', x)\}$

contract$(G = (V, E), C, s)$
1. Let G_C be the subgraph of G excluding nodes in C
2. Add a node c to G_C representing cycle C
3. For $x \in V - C : \exists_{x' \in C}(x', x) \in E$
   Add edge (c, x) to G_C with
   $s(c, x) = \max_{x' \in C} s(x', x)$
4. For $x \in V - C : \exists_{x' \in C}(x, x') \in E$
   Add edge (x, c) to G_C with
   $s(x, c) = \max_{x' \in C} [s(x, x') - s(a(x'), x') + s(C)]$
   where $a(v)$ is the predecessor of v in C
   and $s(C) = \sum_{v \in C} s(a(v), v)$
5. return G_C

Figure 2.1: Chu-Liu-Edmonds algorithm

As described by Mcdonald et al 2005 [12] -

"*The algorithm selects an incoming edge with highest weight for each vertex in the graph. If we do end up with a tree, it must be the maximum spanning tree. If not, there must be a cycle. The process involves identifying a cycle and contracting it into a single vertex and recalculating the edge weights that go into and come out of the cycle. It is observed that a maximum spanning tree on the contracted graph is equivalent to a maximum spanning tree in the original graph (Leonidas, 2003). Hence the algorithm can recursively call itself on the new graph. When the*

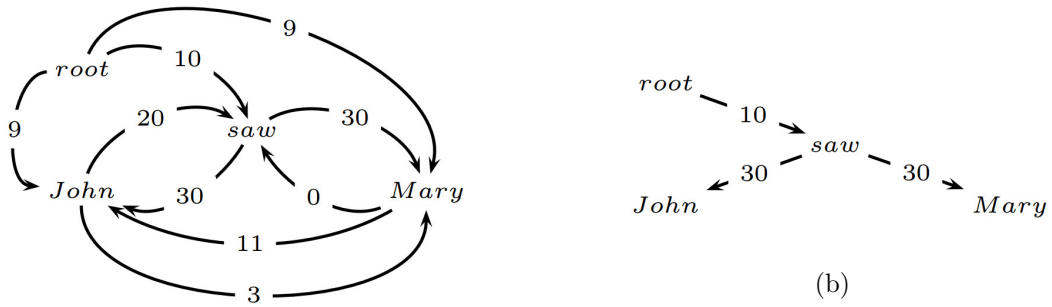


Figure 2.2

algorithm in fig. 2.1 is applied on the directed graph shown in fig. 2.2 (a) we get the resultant tree shown in fig. 2.2 (b)."

The features used in the featuer set f(i,j) are combined with the direction of the attachment and also with the distance between the two words being attached with a relation. As described in Mcdonald et al 2005 [11], these features represent a system of backoff from very specific features over words and part-of-speech tags to less sparse features over just part-of-speech tags.

Our approach is similar to how the dependency parsing problem has been reduced to finding MST. Except, we use Integer Linear Programming to solve the directed graph, that we end up with, by applying the grammatical rules we extracted from the treebank. The coming chapters will explain in detail about how we extract a grammar and apply it on the target sentences. The way we calculate the weight of an edge in the directed graph is where things differ notably when we compare our method with the one explained above.

Chapter *3*

Data-Driven Grammar

Our grammatical framework is very similar to that of the Sketch Grammar [1], which is based on regular expressions over part-of-speech tags. It underlies the word sketches and is written in the Corpus Query Language (CQL). Sketch grammar is designed particularly to identify head-and-dependent pairs of words (e.g., खा [eat] and राम [Ram]) in specified grammatical relations (here, k1 [doer]), in order that the dependent can be entered into the head's word sketch and vice versa. More about the Sketch Grammar would be explained in the coming sections and Sketch Engine would be explained in Chapter 7.

3.1 Grammatical Framework

As explained above, we use Sketch Grammar to identify words in syntactic relations in a given sentence. For example, a grammar rule for the relation

- **k1 (doer)**
  - 2:[tag="NN"] [tag="PSP:ने"] [tag="JJ"]? 1:[tag="VM"]

which specifies that if a noun is followed by a PSP that is followed by an optional adjective which in turn is followed by a verb, then the noun is the kartha/subject of the verb. The head and child are identified by 1: and 2: respectively.

Writing a full-fledged sketch grammar with high coverage is a difficult task even for language experts, as it would involve capturing all the idiosyncrasies of a language. Even though such hand-written rules tend to be more accurate, the recall of the rules is very low. Here, the grammar we use is a collection of POS tag sequences (rules) which are automatically extracted from an annotated Treebank, Hindi Dependency Treebank (HDT-v0.5), which was released for the Coling2012 shared task on dependency parsing (Sharma et al., 2012). This treebank uses IIIT tagset described in (Bharati et al., 2006). This method gives us a lot of rules based on the syntactic ordering of the words.

From the above mentioned treebank (HDT-v0.5) we extract dependency grammar rules (i.e. sketch grammar) automatically for each dependency relation, based on the POS tags appearing in between the dependent words (inclusive).

For example, from the sentence,

राम(Ram) ने(erg.) कमरे(room) मे(inside) आम(mango) खाया(eat)

Ram ate [a] mango in [the] room

we extract rules of the type: (**k1**[doer], **k2**[object], **k7**[location])

- **k1** - 2:[tag="NNP"] [tag="PSP:ने"] [tag="NN"] [tag="PSP:में"] 1:[tag="VM"]
- **k2** - 2:[tag="NN"] 1:[tag="VM"]
- **k7** - 2:[tag="NN"] [tag="PSP:में"] [tag="NN"] 1:[tag="VM"]

In the above example, relation names are in bold with one of the corresponding rules for each of them.

We do include a few lexical features associated with the POS tags PSP (post-position) and CC (conjunction) in order to disambiguate between different dependency relations. For example, in both the relations k1 (doer) and k2 (object) we have the rules (1) and (2) given below respectively:

बच्चे (NN)	ने (PSP)	आम (NN)	खाया (VM)	
(child)	(erg.)	(mango)	(eat.pst)	(the child ate a mango)
2:[tag="NN"]	[tag="PSP\:ने"]	[tag="NN"]	1:[tag="VM"]	------- (1)

कागज़ (NN)	को (PSP)	बाहर (NN)	फेंका (VM)	
(paper)	(acc.)	(outside)	(throw.pst)	([Someone] threw the paper outside)
2:[tag="NN"]	[tag="PSP\:को"]	[tag="NN"]	1:[tag="VM"]	------- (2)

In (1), the ergative marker indicates that the noun (NN) is the doer of the verb (VM). In (2), the accusative marker indicates that the noun (NN) is the object of the verb (VM). Also, (2) is not a complete sentence (in a sense) – the doer has not been mentioned, and only the part of the sentence that the rule is applied to is shown.

If in the rules, the PSP tags didn't contain the lexical features, both the rules would have been the same, and hence both the rules would have been applied on both the sentences, making them ambiguous.

By lexicalizing the PSP POS tag, the rule(s) formed are now less ambiguous, and more accurate.

3.1.1 Weighted Dependency Grammar

After extracting all the dependency rules, we apply each rule on the annotated Treebank (HDT-v0.5), and compute its precision. For example, if the rule

- **k7** (location) - 2:[tag="NN"] [tag="PSP:में"] [tag="NN"] 1:[tag="VM"]

is applied on the HDT-v0.5, we get all the [2:NN, 1:VM] pairs where the rule holds, say N pairs. Out of these N pairs, if M of them are seen correctly with k7 (place) relation in the training data, then the precision of the rule is M/N.

Conditions that need to be satisfied for a rule to be included in the sketch grammar:

- The rule must have a precision of at least 0.05%.
- The frequency of the tag sequence (N) must be greater than 4, to ensure some amount of statistical significance of the rules.
- The context size – the maximum allowed length of the tag sequence (rule), is set to 7 to limit the number of rules generated.

Example rule set (numeral present at the end of the rule tells us the precision of the rule),

- **k1**
  - 2:[tag="NNP"] [tag="PSP:का"] 1:[tag="VM"] **1.0000**
  - 2:[tag="NNP"] [tag="PSP:ने"] 1:[tag="VM"] **0.9624**
  - 2:[tag="NNP"] [tag="PSP:ने"] [tag="NNP"] [tag="PSP:को"] 1:[tag="VM"] **0.8000**
- **k2**
  - 2:[tag="NN"] [tag="PSP:को"] [tag="JJ"] [tag="NEG"] 1:[tag="VM"] **1.0000**
  - 1:[tag="VM"] [tag="VM"] 2:[tag="CC:कि"] **0.9600**
  - 2:[tag="NN"] [tag="PSP:को"] 1:[tag="VM"] **0.7260**
- **k4**
  - 2:[tag="NN"] [tag="PSP:को"] [tag="NN"] 1:[tag="VM"] **0.5230**
  - 2:[tag="NN"] [tag="PSP:को"] [tag="JJ"] [tag="NN"] 1:[tag="VM"] **0.4444**
  - 2:[tag="NN"] [tag="PSP:को"] 1:[tag="VM"] **0.1917**

- **k7p**
  - 2:[tag="NNP"] [tag="PSP:में"] 1:[tag="VM"] **0.6667**
  - 2:[tag="NN"] [tag="PSP:पर"] 1:[tag="VM"] **0.3548**
  - 2:[tag="NN"] [tag="PSP:के"] [tag="NST"] 1:[tag="VM"] **0.1818**

In the examples shown above, the rule 2:[tag="NNP"] [tag="PSP:को"] 1:[tag="VM"] exists in the rule set of both **k2** and **k4**. Each rule may often be matched by more than one relation creating ambiguity. Currently there is no way of disambiguating between such rules in our framework. The rule with the highest precision gets more priority while parsing.

3.2 Comparison

This grammar is similar to Probabilistic Context-Free Grammar (PCFG), discussed in Section 1.2.1, where in a certain probablity is assigned to each rule. We however didn't use any complex machine learning algorithms, but a very simple mechanism mentioned in the above sections. Even in our grammar, the probabilities are constrained by the context of the text (discourse) used to train.

Chapter 4

The Parser

4.1 Integer Linear Programming

Linear programming is a method to achieve the best outcome in a mathematical model whose requirements are represented by linear relationships. Linear programs are problems that can be expressed in canonical format such as:

$$\begin{array}{ll} \text{maximize} & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} & A\mathbf{x} \leq \mathbf{b} \\ \text{and} & \mathbf{x} \geq \mathbf{0} \end{array}$$

where x represents the vector of variables, c and b are vectors of coefficients, A is a matrix of coefficients, and $(\cdot)^{\mathrm{T}}$ is the matrix transpose. The expression to be maximized or minimized is called the objective function (c^Tx in this case). The inequalities $Ax \leq b$ and $x \geq 0$ are the constraints which specify a convex polytope over which the objective function is to be optimized.

If x is restricted only to integers, we call this as an Integer Linear Programming. A variant of this problem is where x is restricted to either 0 or 1, which is known as Zero-One Integer Linear Programming.

We convert our parsing problem into an ILP problem by employing Zero-One Integer Linear Programming to convert the directed graph into a tree. We try to extract the best possible tree from the graph by following a certain set of rules or one might also call them as Constraints in the ILP context. These constraints used in our algorithm are as follows:

- Each node cannot have more than one parent
- Cycles cannot exist in the output parsed tree
- A parent node cannot have more than one child occuring with the following set of dependency relations (deprels) − [k1, k2, etc.] unlike those such as parent-child pairs with relation 'ccof' which can occur multiple times to the same parent node.

- Constraints on conjunctions:
  - The children of conjunctions either being Nouns or Verbs can occur multiple times
  - Only a single child otherwise

4.1.1 Tools and Algorithms

To apply the grammar rules discussed in chapter 3 on an input sentence, we use the Knuth-Morris-Pratt algorithm to find the rule patterns in the given input sentence. By doing so we end up with a graph.

In order to parse this graph (using ILP) we use CVXOPT which is a free software package for convex optimization based on the Python programming language. We also use the NetworkX, another python package, to identify the cycles in the graph that we parse using ILP.

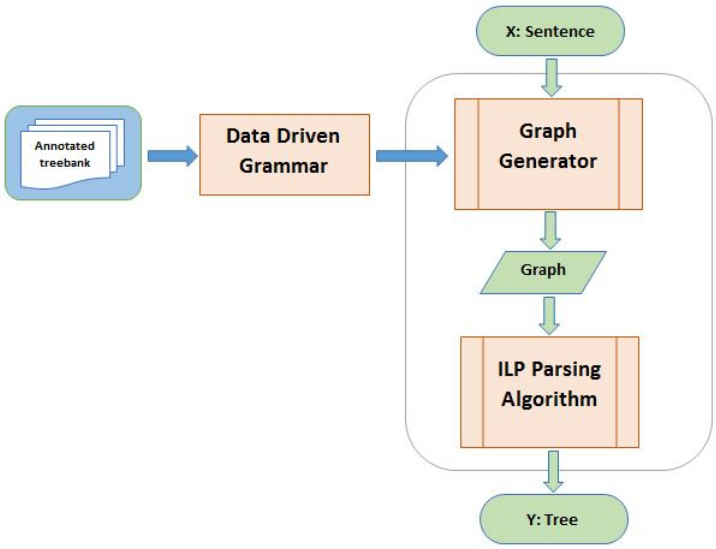


Figure 4.1: Data-Driven Grammar Parser Workflow

4.2 Rule Disambiguity

Lets consider the example sentence (बच्चे ने आम खाया) which is used in the previous sections to apply the grammar and parse the resulting graph.

बच्चे (NN)	ने (PSP)	आम (NN)	खाया (VM)	
(child)	(erg.)	(mango)	(eat.pst)	(the child ate a mango)
2:[tag="NN"]	[tag="PSP\:ने"]	[tag="NN"]	1:[tag="VM"]	

Upon applying the grammar rules discussed in Chapter 3 on the example sentence, we get a graph as shown below.

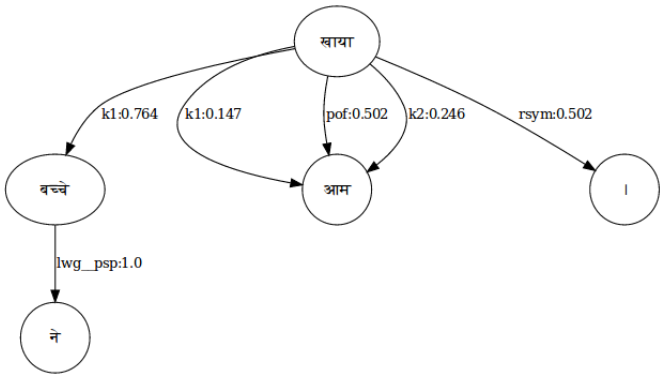


Figure 4.2: Graph to be parsed

Upon using the parser built using the ILP constraints on the graph shown in fig. 4.1, we get the tree as shown below in fig. 4.2.

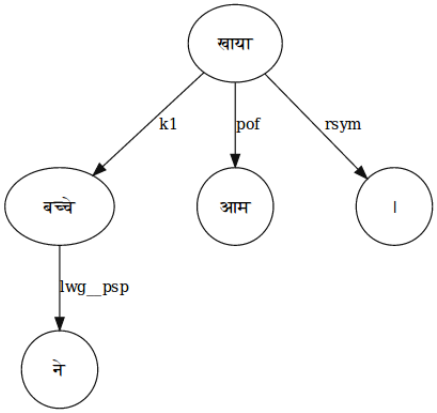


Figure 4.3: Parsed Tree

The parsed tree shown in fig. 4.2 is different from the gold tree (What the parsed tree was supposed to be) shown in fig. 4.3. The dependency relation between खाया (ate) and आम

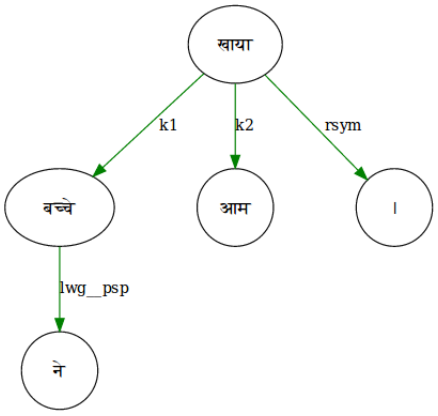


Figure 4.4: Gold Tree

(Mango) is supposed to be **k2** instead of **pof**. This is caused by the ambiguity of the rule shown below which comes up in the grammar rules of both the dependency rules **k2** and **pof**. The parser ends up picking up the edge with dependency relation **pof**.

2:[tag="NN"] 1:[tag="VM"]

If the same grammar is applied upon a sentence (shown below) with little to no ambiguity in the sentence, we can see how well the parser performs (found in fig. 4.4 and fig. 4.5).

- बच्चे (**NN**) ने (**PSP:**ने) आम (**NN**) को (**PSP**:को) कमरे (**NN**) में (**PSP**:में) फेंका (**VM**)

  (The child threw the mango in the room)

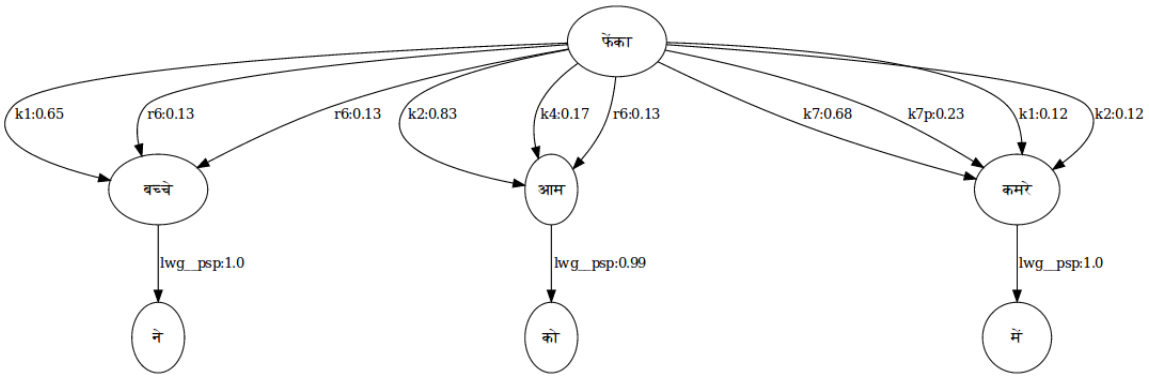


Figure 4.5: Graph to be parsed

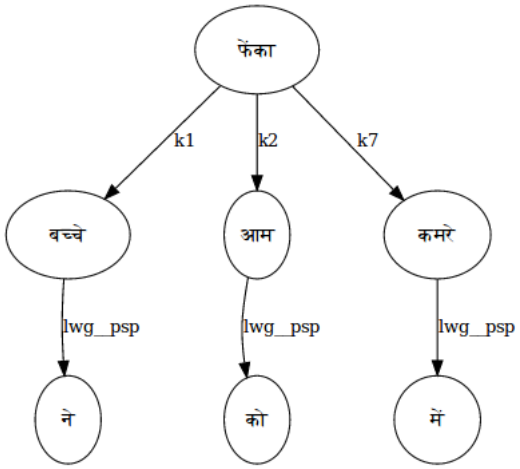


Figure 4.6: Parsed / Gold Tree

4.3 Experiments and Results

4.3.1 Basic Grammar Rules

The initial experiments just had the POS tag patterns directly extracted from the training data of the treebank, **HPST-ver0.51-2012**, as our grammar rules. The resulting graph when parsed performed very poorly, the results of which are shown below.

Labeled attachment score: 11890 / 39775 * 100 = 29.89 %

Unlabeled attachment score: 14315 / 39775 * 100 = 35.99 %

Label accuracy score: 12812 / 39775 * 100 = 32.21 %

4.3.2 Rule Disambiguity In Basic Rules

The low accruracies pointed out that the parser was not able to distinguish between most rules belonging to different dependency relations. This was a result of ambiguity in POS tags lwg___psp and ccof, which led to including the lexical features associated with these tags in the patterns / rules extracted as shown below. This resulted in a considerable improvement in the accruracies of ccof, lwg___cont, lwg___psp, lwg___rp and rsym dependency relations, the total accuracies of which are shown below.

2:[tag="NN"] [tag="PSP:ने"] 1:[tag="VM"]

2:[tag="VM"] 1:[tag="CC:पर"]

Labeled attachment score: 26025 / 39775 * 100 = **65.43 %**

Unlabeled attachment score: 29315 / 39775 * 100 = **73.70 %**

Label accuracy score: 28283 / 39775 * 100 = **71.11 %**

4.3.3 Rule Generalization

In addition to the above grammar rules, for every dependency relation, we have checked the following distance ranges between the parent and child: 1-5, 6-10 and 11-15 to generalize the rules. For example, for the dependency relation **k1**, we obtained the frequencies of the rules:

2:[tag="NN"] [tag="PSP:ने"] [ ]{1,5} 1:[tag="VM"]

2:[tag="NNP"] [ ]{1,5} 1:[tag="VM"]

2:[tag="NNP"] [ ]{6,10} 1:[tag="VM"]

If the obtained frequency of each generalized new rule is considerably high (>= 10), we include the new rule in the grammar. And all the rules that are covered by this generalized new rule are removed from the grammar to remove redundancy. The idea behind doing this was to reduce the rigidity of the rules, thereby dealing with new rule patterns (that didn't appear in the training corpus) which might be similar to the above mentioned new rules. This however has led to a slight reduction in the accuracies.

Labeled attachment score: 24672 / 39775 * 100 = 62.03 %

Unlabeled attachment score: 28747 / 39775 * 100 = 72.27 %

Label accuracy score: 28375 / 39775 * 100 = 71.34 %

4.3.4 Data added from different domains

We have added 300 sentences each from two totally different domains (News Articles and Heritage), from the HDTB pre-release version (**HDTB_pre_release_version-0.05**), to the existing test data of **HPST-ver0.51-2012**. The accuracies, as shown below, has a negligible change. This leads us to believe that the existing rule set performs the same even with new domains.

- **News Articles**

  Labeled attachment score: 29671 / 45706 * 100 = 64.92 %

  Unlabeled attachment score: 33620 / 45706 * 100 = 73.56 %

  Label accuracy score: 32224 / 45706 * 100 = 70.50 %

- **News Articles & Heritage**

  Labeled attachment score: 33276 / 51576 * 100 = 64.52 %

  Unlabeled attachment score: 37846 / 51576 * 100 = 73.38 %

  Label accuracy score: 36103 / 51576 * 100 = 70.00 %

We have also added more data from different domains of **HDTB_pre_release_version-0.05** to the current training set of **HPST-ver0.51-2012** to see if this would result in better accuracies for the existing test data. We observed that the more data was added to the training data from **HDTB_pre_release_version-0.05**, the results were showing reduced accuracies.

- **500 new sentences from HDTB_pre_release_version-0.05**

  Labeled attachment score: 25936 / 39775 * 100 = 65.21 %

  Unlabeled attachment score: 29225 / 39775 * 100 = 73.48 %

  Label accuracy score: 28194 / 39775 * 100 = 70.88 %

- **1000 new sentences from HDTB_pre_release_version-0.05**

  Labeled attachment score: 25764 / 39775 * 100 = 64.77 %

  Unlabeled attachment score: 29005 / 39775 * 100 = 72.92 %

  Label accuracy score: 27995 / 39775 * 100 = 70.38 %

- **1500 new sentences from HDTB_pre_release_version-0.05**

  Labeled attachment score: 25279 / 39775 * 100 = 63.55 %

  Unlabeled attachment score: 29016 / 39775 * 100 = 72.95 %

  Label accuracy score: 27492 / 39775 * 100 = 69.12 %

4.3.5 ISC dependency parser for Hindi

When we try to test the same dataset with the state-of-the-art parser, ISC dependency parser, we can see better accuracies. Although the resulting grammar driven parser is not as good as the ISC dependency parser, the results are decent enough given that the parser depends on grammar rules extracted from the treebank.

- **Test data from HPST-ver0.51-2012**

  Labeled attachment score: 33219 / 39775 * 100 = 83.52 %

  Unlabeled attachment score: 36250 / 39775 * 100 = 91.14 %

  Label accuracy score: 34385 / 39775 * 100 = 86.45 %

- **300 new sentences added to test data from HDTB_pre_release_version-0.05**

Labeled attachment score: 38357 / 45645 * 100 = 84.03 %

Unlabeled attachment score: 41726 / 45645 * 100 = 91.41 %

Label accuracy score: 39665 / 45645 * 100 = 86.90 %

- **600 new sentences added to test data from HDTB_pre_release_version-0.05**

  Labeled attachment score: 43521 / 51576 * 100 = 84.38 %

  Unlabeled attachment score: 47251 / 51576 * 100 = 91.61 %

  Label accuracy score: 44995 / 51576 * 100 = 87.24 %

4.4 Comparison - MST Parser

Our parser is a classic example of the Constraint Based Grammar Parsing explained in the Section 1.2.2, where the parsing is viewed as a constraint statisfaction problem.

In dependencyACL2005 (Mcdonald et al 2005), it is mentioned that using just the features over the parent-child node pairs was not sufficient for high accuracy. In order to solve this problem they added two types of features. The first type of features look at words that occur between a child and its parent. These features take the form of a POS trigram: the POS of the parent, of the child, and of a word in between, for all words linearly between the parent and the child. They explain how this feature was helpful for nouns identifying their parent. The second type of feature provides the local context of the attachment, that is, the words before and after the parent-child pair. This feature took the form of a POS 4-gram: The POS of the parent, child, word before/after parent and word before/after child.

The rules mentioned in the previous chapter can be considered very similar to that of MST's feature set. Although we don't include all of the POS tags of the surrounding words, one could say that the rules are a hybrid of both the features mentioned above. We have all of the POS tags of words between the parent and the child words (inclusive). We could also say that both the parent and child words have half of the local context mentioned above, a word to the left of the parent and a word to the right of the child or vice versa depending on where the parent and child are located in a sentence.

Chapter *5*

Integration with Malt Parser

In this chapter, we present an approach to integrate unlexicalised grammatical features into Malt dependency parser. Malt Parser is a lexicalised parser which has a well-known weakness of data sparseness, similar to every lexicalised parser. Our goal here is to address this issue by providing features from an unlexicalised parser developed as part of this thesis (described in chapters 3, 4).

5.1 Dependendency Parsers

Dependency Parsing has gained popularity due to its ease of representation for any language including free word order languages. With shared tasks like CoNLL [1], [2] and others [3], dependency treebanks for many languages came into existence, and thereby many dependency parsing techniques. Among these, Malt Parser [4] has become one of the popular dependency parsers due to its ease of training, capability of handling various features, faster training and parsing rates. We use Malt Parser in place of other higher-order graph-based parsers as we want to improve greedy dependency parsing, which is still one of the most efficient parsing methods available. Malt is a transition-based parser with greedy local search. Malt uses lexicalised information i.e., the word itself, along with the features specified, to take a best local decision while parsing. While lexicalised information is useful, the algorithm is prone to difficulties when the words are unseen since the algorithm has to base its decision only on the features seen. The features, in general, are unigram, bigram or trigram of various linguistic properties around a fixed window from the current word.

There have been many attempts of improving a parser's performance using features from other parsers. [5] used features from [6] and [7] parsers which improved MSTParser's UAS (Unlabeled Attachment Score) by 1.7%. [8] and [9] used a technique called blending where output of different parsers is used to generate the final parse. [10] used CCG (Combinatory Categorial Grammar) features to improve dependency parser. Output of the CCG supertagger is provided as features to Malt which resulted in significant improvements.

index	word	POS tag	1st best relation	2nd best	3rd best
1	The	DT	4:det:0.89159	3:det:0.870516	2:det:0.846192
2	luxury	NN	3:nn:0.734832	4:nn:0.715789	4:amod:0.1166
3	auto	NN	4:nn:0.734832	6:nn:0.139535	
4	maker	NN	7:nsubj:0.273	6:nn:0.169154	6:amod:0.1293
5	last	JJ	6:amod:0.7330	4:amod:0.2280	
6	year	NN	7:nsubj:0.567	4:tmod:0.1268	
7	sold	VBD	0:ROOT:0.453		
8	1,214	CD	9:num:0.96036	7:dobj:0.185424	
9	cars	NNS	7:dobj:0.6859		
10	in	IN	9:prep:0.6236	7:prep:0.479452	
11	the	DT	12:det:0.2726		
12	U.S.	NNP	10:pobj:0.228		

Figure 5.1: Top 3 parents for each word. The format of each relation above is parent node index : relation name : precision of the rule applied. For example, the 2nd best relation on word index 3 i.e. *auto* is *6:nn:0.139535* indicating that word index 6 is the parent with dependency relation *nn* with confidence 0.139535 (i.e. precision of rule **nn** *2:NN NN JJ 1:NN*. Note: These are not the features themselves. Features are constructed from these relations, e.g. feature **n-Rels** of word index 4 (i.e. maker) are 1-nsubj, 2-nn, 3-amod

5.2 Unlexicalized parser: Intermediate Output

Unlexicalised grammar-driven parsers [11], [12] are well known for their robustness in handling unseen words, and are also competitive in performance with statistical data driven lexicalised parsers. We aim to integrate information from a simple grammar-driven unlexicalised parser into Malt, and thereby increase the robustness of Malt.

We just use a part of the parsing technique described in the previous chapters for integrating with Malt Parser. We define our unlexicalised grammar as a set of rules corresponding to each dependency relation type. Each grammatical rule is a sequence of POS tags between (and including) the head and child words. We extract these rules from the training data. For each sentence in the test data, we apply all the rules extracted in the above steps. If we define a sentence as a set of nodes in a graph, each rule is like an edge connecting two nodes in the graph, with relation name as the edge label. For each child node (i.e. indicated by 2: in an applied rule), we select its n-best parent nodes (i.e. indicated by 1: in an applied rule) by sorting the rules based on their weights (in descending order). In Figure 1, for each word, we show its 3 best parent nodes.

5.3 Integrating Grammatical Features with Malt

As mentioned in the earlier sections, we use the intermediate output of our parser, namely, the n-best relations for each word, to generate features which we integrate with Malt. For each word in the sentence, we specify its possible parents, relations and the scores derived from our

Table 5.1: Impact of new feature set on Malt baseline. McNemar's test, ** = $p < 0.01$, * = $p < 0.05$
† - Unlexicalised

Language	*Experiment/FEATS*	*LAS*	*UAS*
English	Stanford Baseline (BL)	87.71	90.17
	Stanford BL + {5-posTags, 5-ScoreRanges}	**87.90****	**90.34****
	† Stanford Baseline (BL)	79.45	83.39
	† Stanford BL + {3-posTags, 3-ScoreRanges}	**79.90****	**83.85****
	CoNLL Baseline (BL)	88.73	90.00
	CoNLL BL + {2-Rels, 2-posTags}	**88.88***	**90.20****
	† CoNLL Baseline (BL)	83.31	85.68
	† CoNLL BL + {4-Rels, 4-posTags, 4-ScoreRanges}	**84.01****	**86.29****
Hindi	Baseline (BL)	87.97	93.40
	BL + {5-Rels, 5-posTags}	**88.25****	**93.70****
	† Baseline (BL)	83.33	91.79
	† BL + {3-Rels, 3-posTags}	**84.16****	**92.65****

parser. As far as we know, the only way to encode the predicted head of a word in a feature is through InputArc and InputArcDir constructors, but we chose not to use them as they only work on 1-best predictions. Instead, we specify the features of parent word as parent indicating features to the child word. Similarly, it is not possible to specify scores from our parser as numerical features to Malt. Instead we convert score to a range based on the magnitude of the score. Our feature set from the grammar-driven parser is as follows.

1. **n-Rels:** Relations between the current word and the nth-best parents specified by the grammar-driven parser.

2. **n-posTags:** POS tags of the n-best parents.

3. **n-ScoreRanges:** The range in which score of the n-best parents falls in. We define the ranges as follows. 1 for 90-100%, 2 for 80-90% and {3,4,5,6} for 80-0% in splits of 20.

4. **n-ChunkTags:** Chunk tags of the n-best parents. If n is 3, we have 1-ChunkTag, 2-ChunkTag, 3-ChunkTag features.

The generic feature 'rK-V' - where r is the rank, K is one of {rel, H, PS, `H_chunkID`} and V is the value, means there are r features added for the ranks 1 to r.

If n is 3, then we have one feature each for the 1st, 2nd and 3rd best feature value.

Apart from the above described features, we define another feature which is similar to feature 2. In this case we modify the pos tag of each word of both the train and the test data to

match the form `pos_chunkID`, where the chunkType (head/child in the local word group) of the respective word is attached to the postag in the pos column.

key: *rank*+'`H_`'+*chunkID*

value: pos tag of the possible parent node with the chunkID of the parent node

In our experiments, we add the above features from grammar-driven parser to the existing state-of-the-art Malt feature set (refer next section) to form a new feature set, and tune over the complete feature set using development data to get the best settings.

We also experiment with Malt by disabling the lexical features. Since our grammar rules are purely unlexicalised, this would serve as a control experiment to estimate the performance of the unlexicalised Malt Parser.

5.4 Experiments

In this section we describe the datasets we used for English and Hindi and our experimental settings.

5.4.1 Data and Tools

For English, we experimented with two different dependency annotation schemes, namely, CoNLL which has a label set of 11 labels, and Stanford which has a much larger label set of 48 labels, both of which are widely popular. We used Penn Treebank [9] with standard splits - sections 02-21 for training, section 22 for development and section 23 for testing. We extracted Stanford dependencies and CoNLL dependencies from Penn Treebank using Stanford Parser built-in converter with basic projective option and Penn2Malt[1] respectively.

For Hindi, we used the Hindi Dependency Treebank (HDT-v0.5) which was released for the Coling2012 shared task on dependency parsing [3]. This treebank contains 12,041 training, 1,233 development and 1,828 testing sentences with an average of 22 words per sentence and has a lable set of 63 labels.

The dependency labels for English are syntactic; whereas for Hindi, they are syntactico-semantic in nature. We use predicted POS tags for English, and the gold POS tags for Hindi.

5.4.2 Final Feature Set and Malt Settings

We use state-of-the-art English and Hindi feature sets of Malt as our baselines [17], [3]. The state-of-the-art Malt features for English are based on word and POS. For Hindi, apart from word, lemma and POS, the features also include morphological properties such as case markers

[1] http://w3.msi.vxu.se/nivre/research/Penn2Malt.html

Table 5.2: Distance wise accuracies with our improved feature set
1-5,6-10 and >10 represent the parent-child disntance

Language	*Experiment*	*LAS*				*UAS*		
		1-5	*6-10*	*>10*		*1-5*	*6-10*	*>10*
English	Stanford Baseline (BL)	89.76	70.69	72.17		91.89	73.54	74.17
	Stanford BL+BestFeat	89.88	70.46	73.52		91.98	73.22	75.72
	CoNLL Baseline (BL)	90.61	72.64	73.18		91.86	73.34	73.43
	CoNLL BL+BestFeat	90.69	73.00	73.82		91.98	73.70	74.17
Hindi	Baseline (BL)	91.33	73.59	71.42		96.21	82.82	78.14
	BL+BestFeat	91.57	73.86	72.10		96.44	83.24	78.99

for nouns; tense, aspect and modality markers for verbs; and chunk properties such as chunk label and chunk type (indicates if the word is head or non-head of the chunk).

In our models, in addition to the baseline feature set, we also include the features from our grammar-driven parser as described in Section 5.3. These features are provided in the FEATS column of CONLL format[2]. We tried various combinations of features: baseline features + combinations of n-Rels, n-posTags, n-ScoreRanges.

All the experiments are carried out using Malt Parser version 1.7.2[3]. We use *nivrestandard* as the parsing algorithm and *liblinear* as the learner in our experiments. The rest of the settings are set to default.

5.5 Results

Table 5.1 displays the results of our experiments with English and Hindi on test data. We chose the best performing feature set based on the results on development data. As mentioned before, we also ran Malt by disabling lexical features - these results are marked by †.

On lexicalised data for English, we obtained an improvement of 0.19% LAS and 0.17% UAS on Stanford dependencies, and 0.15% LAS and 0.20% UAS on CoNLL scheme. For Hindi, we achieved an improvement of 0.28% LAS and 0.30% UAS. All these improvements are statistically significant ($p < 0.01$ for all except CoNLL-LAS with $p < 0.05$ using McNemar's test).

In Table 5.2, we also present the distance wise improvements of dependency relations. We observed improvements on all distance ranges (except 6-10 on Stanford), which shows that our grammar is effective even for long distance dependencies.

[2]http://nextens.uvt.nl/depparse-wiki/DataFormat
[3]Malt Parser v1.7.2 can be downloaded at http://www.maltparser.org/download.html

We have considered the following distance ranges: '0-5', '6-10' and '>10' (greater than 10). Among these, the *0-5* range is covered by smaller rules and the *>10* range, as explained above, are covered by larger rules. This causes, as shown in Table 5.2, a significant improvement for relations over 10 words apart, and those in the '*0-5*' range.

5.6 Related Work

In our work, we incorporated features into Malt from an unlexicalised grammar-driven parser. There have been many attempts of improving a parser's performance using features from other parsers. [10] used features from [5] and [4] parsers which improved MSTParser's UAS (Unlabeled Attachement Score) by 1.7%. [15] introduced a technique called blending where output of the parsers is used to generate the final parse. In this approach, based on the information from different parsers, a dependency graph is generated. Maximum spanning tree algorithm is used to extract the final dependency tree from this graph. This technique has been used by [7] to produce best performing system in CoNLL 2007 [13]. [14] integrated graph-based model (MST) and transition-based model (Malt) and improved parsing accuracy. Features from one model are used to extend other model during training, which improved accuracies for both models. Using this approach, they obtained significant improvement over the state-of-the-art on CoNLL-X shared task data sets.

The other stream of work is improving parsing performance of a grammatical framework using other grammatical frameworks. [16] improved Head-driven Phrase Structure Grammar (HPSG) parser by using the output of a dependency parser. [6] used higher-order dependency features to improve phrase structure parsing. They incorporated higher-order dependency features into a cube decoding phrase-structure parser and obtained significant gains on dependency recovery for both in-domain and out-of-domain test sets. [8] improved CCG parser using dependency features. They first extracted n-best parsers for a sentence using a CCG parser. Then dependency features from a dependency parser are provided to a re-ranker. [2] used CCG features to improve dependency parser. Output of the CCG supertagger is provided as features to Malt which resulted in significant improvements.

Our approach is much simpler, and widely applicable for many languages without much hassle. Though our method is not as sophisticated as others, we still achieve better results than state-of-the-art Malt settings.

Chapter *6*

Hindi Word Sketches

Word sketches are one-page automatic, corpus-based summaries of a word's grammatical and collocational behavior. These are widely used for studying a language and in lexicography. Sketch Engine is a leading corpus tool which takes a corpus as its input and generates word sketches for the words of that language. It also generates a thesaurus and "sketch differences", which specify similarities and dissimilarities amongst near synonyms. In this paper, we present the functionalities of Sketch Engine for Hindi. We collected HindiWaC, a web crawled corpus for Hindi with 240 million words. We lemmatized, POS tagged the corpus and then loaded it into Sketch Engine.

6.1 Introduction

A language corpus is simply a collection of texts, so-called when it is used for language research. Corpora can be used for all sorts of purposes: from literature to language learning; from discourse analysis to grammar to language change to sociolinguistic or regional variation; from translation to technology.

Corpora are becoming more and more important, because of computers. On a computer, a corpus can be searched and explored in all sorts of ways. Of course that requires the right app. One leading app for corpus querying is the Sketch Engine (Kilgarriff et al., 2004). The Sketch Engine has been in daily use for writing dictionary entries since 2004, first at Oxford University Press, more recently at Cambridge University Press, Collins, Macmillan, and in National Language Institutes for Czech, Dutch, Estonian, Irish, Slovak and Slovene. It is also in use for all the other purposes listed above. On logging in to the Sketch Engine, the user can explore corpora for sixty languages. In many cases the corpora are the largest and best available for the language. For Indian languages, there are the corpora for Bengali, Gujarati, Hindi, Malayalam, Tamil and Telugu. The largest is for Hindi with 240 million words – we would be referring to it as HindiWaC in the rest of the paper.

कर (verb)
HindiWaC Sketches - Grammar_70% freq = 4955783 (18,246.1 per million)

pof	1,345,592	6.6	lwg__cont	1,027,372	5.2	k2	283,996	3.6	k7	161,837	5.7	k1	114,164	2.7
प्राप्त	50,479	10.16	है	629,115	11.08	कि	41,434	9.96	तौर	4,268	9.01	लोग	3,437	7.23
बंद	33,516	9.6	था	151,263	10.6	लोग	4,837	7.43	आधार	3,600	8.93	सरकार	2,592	7.91
तैयार	33,110	9.58	सक	61,025	10.13	कार्य	2,396	7.63	बात	3,250	7.44	पुलिस	1,869	8.03
प्रस्तुत	25,295	9.23	रह	54,961	9.12	बात	2,192	6.61	स्तर	2,517	8.24	सिंह	1,254	7.21
पैदा	24,767	9.18	हो	22,890	6.65	मन	1,581	6.98	विषय	1,804	7.79	कुछ	955	5.94
पूरा	24,715	9.1	गया	22,153	8.66	काम	1,370	6.74	स्थान	1,697	7.53	जी	926	6.75
तय	23,929	9.15	जा	19,941	7.93	जीवन	1,277	6.41	नाम	1,681	7.27	विभाग	672	6.77
स्थापित	23,403	9.12	चाह	17,318	8.6	कमी	1,110	6.87	पद	1,610	7.79	अधिकारी	597	6.47
जारी	23,322	9.09	गई	6,716	7.36	समाज	1,064	6.25	मुद्दा	1,338	7.63	कंपनी	588	6.46
पेश	23,064	9.1	पहले	5,268	7.18	बच्चा	1,050	6.18	समय	1,243	6.37	वह	537	6.44

Figure 6.1: Word sketches for the verb कर (do)

The function that gives the Sketch Engine its name is the 'word sketch', as shown in fig 6.1. Since the images in this paper are screenshots taken from Sketch Engine, translations and gloss have not been provided for the Hindi words in the images. In this paper we first introduce the main functions of the Sketch Engine, with Hindi examples. We then describe how we built and processed HindiWaC, and set it up in the Sketch Engine.

6.2 The Sketch Engine For Hindi

6.2.1 The Simple Concordance Query Function

A Simple concordance query shows the word as it is used in different texts. We have fig. 6.2 shows the query box, while fig. 6.3 shows its output. A simple search query for a word such कर (do) searches for the lemma as well as the words which have कर (do) as the lemma, so कर (do), किया (did), करने (to do), करते ([they] will do), etc. are all retrieved. Figure 3 shows the first 20 results out of the retrieved 5 million results.

Figure 6.2: Simple concordance query

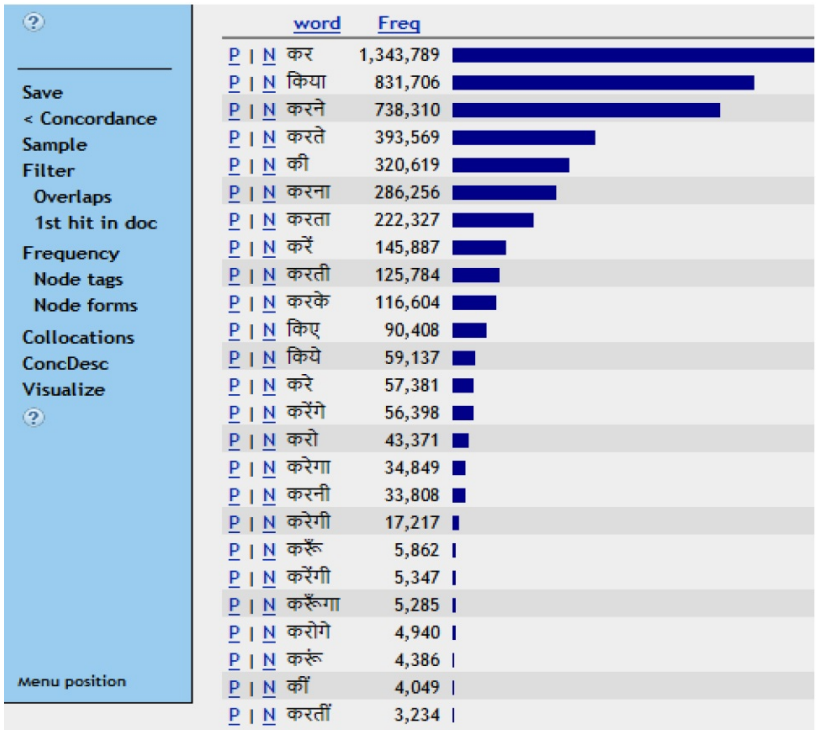
Query कर 4,959,057 (18,258.1 per million)

Page 1 of 247,953 Go Next | Last

#31	आंखों से दिख रहा है, इंद्रियां जिसे अनुभव	कर	रही हैं और दूसरा आध्यात्मिक अर्थात जो हमें
#121	समझ गई कि उसका पति कोई निकृष्ट स्वप्न देख	कर	जगा है। शाम तक उसने उसे कुछ नहीं दिया जब
#200	। ऋषि, महर्षि, आचार्यों ने इस भूमि पर रह	कर	संसार को असत्य अर्थात स्वप्निल कहा है,
#219	स्वप्न के समान। गृहस्थ भी समाज देश में रह	कर	अपना स्थान मात्र स्मारक, फोटो फ्रेम तक
#406	पूर्व का घटित सत्य होता है। आगे इस पर विचार	करेंगे	कि प्रायः दिखने वाले स्वप्नों के क्या कारण
#506	है और कहां चली जाती है। इसे कौन नियंत्रित	करता	है ? वह ÷मन' है। शांत भाव से सोचते हैं
#561	है। जैसे बंद मोबाइल, कंप्यूटर जब भी चालू	करेंगे	, वह समय ठीक ही बताएगा। अर्थात उसमें कुछ
#765	संग्रहालय है। जागृत अवस्था में मन से उसे संचालन	करते	हैं फिर भी बहुत से मस्तिष्क रूपी कंप्यूटर
#777	बहुत से मस्तिष्क रूपी कंप्यूटर खोलने और बंद	करने	में दिन भर की बाधाएं आती रहती हैं। मस्तिष्क
#803	होती तो दिमाग बोझिल होकर निंद्रा में, कार्य	करने	पर स्वप्नों का निर्माण करता है। जब जगते
#808	निंद्रा में, कार्य करने पर स्वप्नों का निर्माण	करता	है। जब जगते हैं तो उनका अपुष्ट स्वरूप ध्यान
#824	उनका अपुष्ट स्वरूप ध्यान आता है, फिर विचार	करते	हैं कि यह शुभ फल देगा या अशुभ। स्वप्न का
#871	वर्तमान चेतना के बोध को भले ही अस्वीकार	करे	, लेकिन सच्चा ज्ञान एवं घटित कार्य कलाप
#929	विचारक मनीषी इसे तालिका के रूप में प्रस्तुत	करते	हैं। बहुत ही प्राचीन काल से यह धारणा चली
#1129	इसके उपचार में यथासाध्य दिमागी रोग का इलाज	करते	हैं। ज्योतिष संभावनाओं और पूर्व सूचनाओं
#1197	बाह्याभ्यन्तरः शुचिः'॥ बुरे सपनों को दूर	करने	हेतु जातक या उसके माता पिता को श्रद्धा
#1212	पिता को श्रद्धा से पूजा-पाठ के समय इसका जप	करना	चाहिए। नियमित रूप से इष्ट/गुरु के सम्मुख
#1238	जा सकता है। कुछ ही दिनों में भय उत्पन्न	करने	वाले एवं निकृष्ट सपनों का निश्चित परिमार्जन
#1268	आकृतियों का आभास होने लगता है। आंखें बंद	कर	पूजा करने वाले भी देवताओं की आकृति का स्मरण
#1270	का आभास होने लगता है। आंखें बंद कर पूजा	करने	वाले भी देवताओं की आकृति का स्मरण करते

Page 1 of 247,953 Go Next | Last

Figure 6.3: The resulting concordance lines



Save
< Concordance
Sample
Filter
Overlaps
1st hit in doc
Frequency
Node tags
Node forms
Collocations
ConcDesc
Visualize
Menu position

	word	Freq
P | N	कर	1,343,789
P | N	किया	831,706
P | N	करने	738,310
P | N	करते	393,569
P | N	की	320,619
P | N	करना	286,256
P | N	करता	222,327
P | N	करें	145,887
P | N	करती	125,784
P | N	करके	116,604
P | N	किए	90,408
P | N	किये	59,137
P | N	करे	57,381
P | N	करेंगे	56,398
P | N	करो	43,371
P | N	करेगा	34,849
P | N	करनी	33,808
P | N	करेगी	17,217
P | N	करूँ	5,862
P | N	करेंगी	5,347
P | N	करूँगा	5,285
P | N	करोगे	4,940
P | N	करूं	4,386
P | N	कीं	4,049
P | N	करतीं	3,234

Figure 6.4: Frequency of word forms of कर (do)

6.2.2 The Frequency Functions

The Sketch Engine interface provides easy access to tools for visualizing different aspects of the word frequency (see fig. 6.4). The Frequency Node forms function on the left hand menu in Figure 4 shows which of the returned forms are most frequent.

Thus we have immediately discovered that the commonest forms of the lemma कर (do) are कर (do), किया (did) and करने (to do).

The p/n links are for positive and negative examples. Clicking on p gives a concordance for the word form, while clicking on n gives the whole concordance except for the word form.

6.2.3 The Word List Function

The Word List function allows the user to make frequency lists of many types (words, lemmas, tags). We have fig. 6.5 which shows the most frequent words in the corpus. In addition to most frequent words, keywords of any target corpus can be extracted. This is done by comparing frequent words from the target corpus with the frequent words from a general purpose corpus. We have fig. 6.6 which displays the keywords of a Hindi Election corpus, where this is the target corpus, and the general purpose corpus is the HindiWaC.

Figure 6.5: Word list function

Almost every keyword closely relates to the trend of news articles in the 2014 Indian Parliament elections. Since the Hindi Election Corpus is of small size, the frequency-per-million column contains projected values. These are significantly higher than the same words in the HindiWaC since the Election Corpus is domain specific.

	Hindi Election Corpus		*HindiWaC*		
word	Freq	Freq/mill	Freq	Freq/mill	Score
केजरीवाल	234	7801.8	2,097	31.9	8.5
पार्टी	209	6968.3	17,336	263.6	6.3
मोदी	161	5367.9	4,884	74.3	5.9
सरकार	222	7401.7	59,774	908.8	4.4
कांग्रेस	130	4334.3	18,838	286.4	4.1
चुनाव	115	3834.2	11,435	173.9	4.1
राहुल	105	3500.8	8,324	126.6	4.0
सम्मेलन	84	2800.7	4,093	62.2	3.6
गुजरात	83	2767.3	5,203	79.1	3.5
कहा	285	9502.2	136,695	2078.3	3.4
भाजपा	82	2734.0	9,501	144.5	3.3
गांधी	88	2934.0	15,291	232.5	3.2
दिल्ली	107	3567.5	30,665	466.2	3.1
प्रधानमंत्री	75	2500.6	12,890	196.0	2.9
मंत्री	74	2467.2	17,495	266.0	2.7
वाराणसी	54	1800.4	1,499	22.8	2.7
उन्होंने	130	4334.3	65,402	994.4	2.7
नरेंद्र	44	1467.0	1,502	22.8	2.4
मुख्यमंत्री	53	1767.1	10,822	164.5	2.4
विधानसभा	44	1467.0	3,764	57.2	2.3

Figure 6.6: Frequency list of the whole corpus for Words and Keywords extracted automatically from Hindi Election Corpus by comparing it with Hindi Web Corpus

6.2.4 The Word Sketch and Collocation Concordance Functions

The Word Sketch function is invaluable for finding collocations. The word sketches of the word लोग (people) for three dependency relations are shown in fig. 6.7.

The dependency relations that we use are based on the Paninian framework (Begum et al., 2008). Three of the most common dependency relations given by this model are as follows:

1. k1: agent and/or doer
2. k2: object and/or theme
3. k3: instrument

लोग (noun)
HindiWaC Sketches - Grammar_70% freq = 568946 (2,094.7 per million)

nmod adj	240,145	4.3	k1 inv	22,290	6.4	nmod	12,337	9.8
कुछ	24,854	10.33	जम	159	7.05	मानसिकता	104	7.01
उन	11,086	10.12	मान	1,117	6.42	ज्यादातर	52	6.04
ऐसे	10,530	9.98	कह	2,606	6.0	मुताबिक	73	6.04
कई	11,367	9.66	मार	265	5.83	अधिकतर	40	5.83
सभी	7,664	9.28	मर	152	5.77	जी	86	5.8
सब	5,960	9.21	जता	76	5.73	रसूख	21	5.73
आम	5,412	9.16	सोच	277	5.72	सोच	65	5.51
जिन	4,453	9.1	मचा	57	5.56	आय	37	5.49
लाख	4,871	9.1	समझा	141	5.53	राशि	86	5.45
इन	7,082	9.02	बता	616	5.53	रख	550	5.36
अन्य	4,844	8.76	सराह	36	5.5	गुजर	57	5.29
स्थानीय	3,444	8.72	पूछ	172	5.48	पहुंचने	33	5.22
यह	11,814	8.67	घेर	48	5.42	बस	30	5.11
कम	3,329	8.46	देख	727	5.14	मर	81	5.03
हजारा	2,322	8.22	थाना	49	5.08	विचारधारा	29	5.03
सारा	2,851	8.2	उठा	166	5.02	विपरीत	24	4.99
ज्यादातर	1,958	7.98	चुन	68	4.99	आमदनी	15	4.86
अधिक	2,581	7.97	बोल	193	4.99	पैसा	49	4.83
अनेक	2,215	7.93	बैठ	142	4.95	छवि	25	4.8
ज्यादा	2,257	7.87	बजा	42	4.95	कमा	31	4.78

Figure 6.7: Word Sketch results for लोग (people)

These relations are syntactico-semantic in nature, and differ slightly from the equivalent thematic roles mentioned above. More about how we get the word sketches shown in fig. 6.7 is explained in Section 6.3.

The three dependency relations shown in fig. 6.7 are:

1. nmod_adj: noun-modifier_adjective

2. k1_inv: doer_inverse

3. nmod: nounmodifier

The word sketch function assigns weights to each of the collocates and also to the dependency relations.

Clicking on the number after the collocate gives a concordance of the combination (fig. 6.8).

Figure 6.8: Concordance lines for लोग (people) in combination with its gramrel "nmod"

6.2.5 The Bilingual Word Sketch Function

A new function has been added recently to the Word Sketch, which is the Bilingual Word Sketch. This allows the user to see word sketches for two words side by side in different languages. Fig. 6.9 shows a comparison between लाल (red) and red. Interestingly, the usage of word red in Hindi and English are very diverse. The only common noun which is modified by red in both languages is गुलाब (rose).

6.2.6 Distributional Thesaurus and Sketch Diff

The Sketch Engine also offers a distributional thesaurus, where, for the input word, the words 'sharing' most collocates are presented. Fig. 6.10 shows the top entries in similarity to कर (do). The top result is हो (be). Clicking on it takes us to a 'sketch diff', a report that shows the similarities and differences between the two words in Figure 11.

The red results occur most frequently with हो (be), the green ones with कर (do). The ones on white occur equally with both.

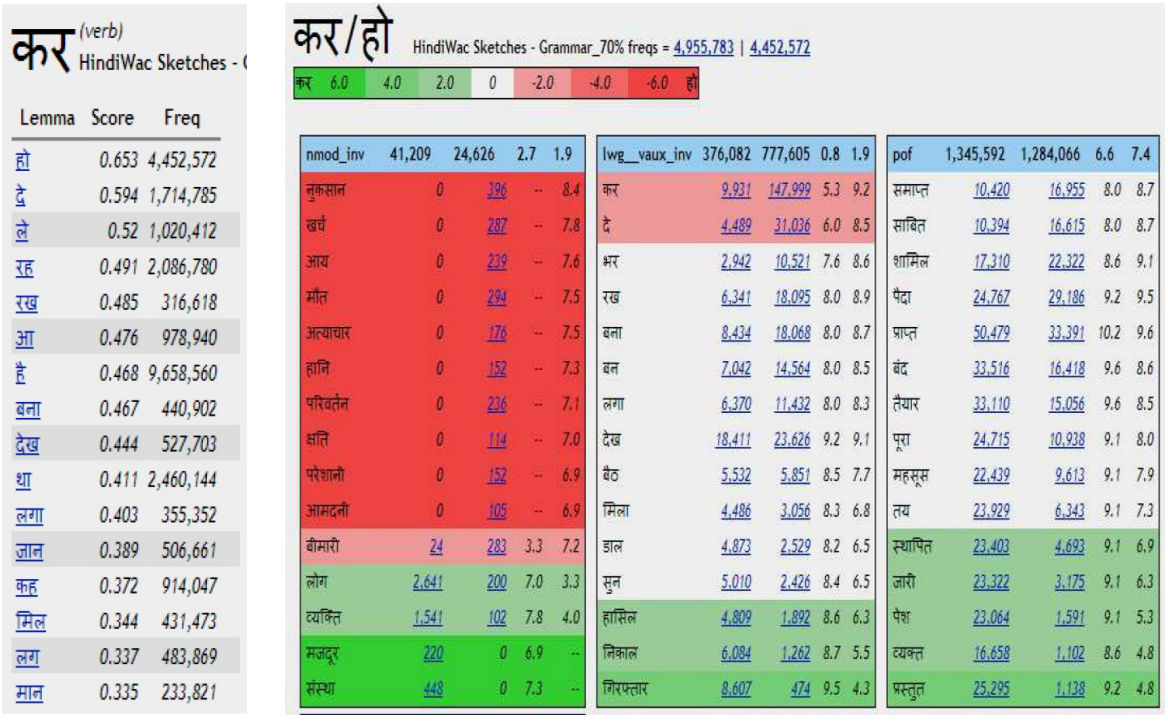
लाल (adjective) Alternativ HindiWac Sketches - G

modifier of	6,082	7.8
मिर्च	579	10.95
रंग	938	10.64
किला	337	10.53
किताब	303	9.83
शास्त्री	168	9.6
चंदन	160	9.51
बत्ती	112	9.12
वस्त्र	148	9.08
फूल	136	8.64
पुष्प	62	8.15
धागा	54	8.01
गुलाब	54	8.01
यादव	64	7.87
डोरा	45	7.86
कपड़ा	76	7.86
बादू	47	7.68

red (noun) ukWaC Freq

modifies	76,712	3.1
squirrel	1,549	9.12
tape	2,781	9.11
wine	2,668	8.63
herring	853	8.4
deer	952	8.26
brick	1,070	8.14
sandstone	753	8.09
pepper	818	8.04
flag	1,087	7.9
cell	2,590	7.81
meat	938	7.62
carpet	639	7.59
kite	434	7.37
onion	528	7.36
light	2,350	7.34
rose	375	7.03

Figure 6.9: Adjective results of a bilingual word sketch for Hindi लाल (red) and English red English translations of some of the Hindi words are: chilli, colour, fort, flower, rose, cloth, Shastri



कर (verb) HindiWac Sketches - (

Lemma	Score	Freq
हो	0.653	4,452,572
दे	0.594	1,714,785
ले	0.52	1,020,412
रह	0.491	2,086,780
रख	0.485	316,618
आ	0.476	978,940
है	0.468	9,658,560
बना	0.467	440,902
देख	0.444	527,703
था	0.411	2,460,144
लगा	0.403	355,352
जान	0.389	506,661
कह	0.372	914,047
मिल	0.344	431,473
लग	0.337	483,869
मान	0.335	233,821

कर/हो HindiWac Sketches - Grammar_70% freqs = 4,955,783 | 4,452,572

कर 6.0 4.0 2.0 0 -2.0 -4.0 -6.0 हो

nmod_inv	41,209	24,626	2.7	1.9
नुकसान	0	396	--	8.4
खर्च	0	287	--	7.8
आय	0	239	--	7.6
मौत	0	294	--	7.5
अत्याचार	0	176	--	7.5
हानि	0	152	--	7.3
परिवर्तन	0	236	--	7.1
क्षति	0	114	--	7.0
परेशानी	0	152	--	6.9
आमदनी	0	105	--	6.9
बीमारी	24	283	3.3	7.2
लोग	2,641	200	7.0	3.3
व्यक्ति	1,541	102	7.8	4.0
मजदूर	220	0	6.9	--
संस्था	448	0	7.3	--

lwg__vaux_inv	376,082	777,605	0.8	1.9
कर	9,931	147,999	5.3	9.2
दे	4,489	31,036	6.0	8.5
भर	2,942	10,521	7.6	8.6
रख	6,341	18,095	8.0	8.9
बना	8,434	18,068	8.0	8.7
बन	7,042	14,564	8.0	8.5
लगा	6,370	11,432	8.0	8.3
देख	18,411	23,626	9.2	9.1
बैठ	5,532	5,851	8.5	7.7
मिला	4,486	3,056	8.3	6.8
डाल	4,873	2,529	8.2	6.5
सुन	5,010	2,426	8.4	6.5
हासिल	4,809	1,892	8.6	6.3
निकाल	6,084	1,262	8.7	5.5
गिरफ्तार	8,607	474	9.5	4.3

pof	1,345,592	1,284,066	6.6	7.4
समाप्त	10,420	16,955	8.0	8.7
साबित	10,394	16,615	8.0	8.7
शामिल	17,310	22,322	8.6	9.1
पैदा	24,767	29,186	9.2	9.5
प्राप्त	50,479	33,391	10.2	9.6
बंद	33,516	16,418	9.6	8.6
तैयार	33,110	15,056	9.6	8.5
पूरा	24,715	10,938	9.1	8.0
महसूस	22,439	9,613	9.1	7.9
तय	23,929	6,343	9.1	7.3
स्थापित	23,403	4,693	9.1	6.9
जारी	23,322	3,175	9.1	6.3
पेश	23,064	1,591	9.1	5.3
व्यक्त	16,658	1,102	8.6	4.8
प्रस्तुत	25,295	1,138	9.2	4.8

Figure 6.10: Thesaurus search showing entries similar to कर (do) (left) and Sketch Diff comparing collocates of कर (do) and हो (be) (right)

6.3 Building and processing HindiWaC and loading it into the Sketch Engine

HindiWaC was built using the Corpus Factory procedure (Kilgarriff et al., 2010). A first tranche was built in 2009, with the crawling process repeated and more data added in 2011, and again in 2014. Corpus Factory method can be briefly described as follows: several thousands of target language search queries are generated from Wikipedia, and are submitted to Microsoft Bing search engine. The corresponding hit pages for each query are downloaded. The pages are filtered using a language model. Boilerplate text is removed using body text extraction, deduplication to create clean corpus. We use jusText and Onion tools (Pomikálek, 2011) for body-text extraction and deduplication tools respectively.

The text is then tokenized, lemmatized and POS-tagged using the tools downloaded from http://sivareddy.in/downloads (Reddy and Sharoff, 2011). The tokenizer found here is installed in the Sketch Engine.

For the grammar to be applied on the HindiWaC, we use the Sketch Grammar extracted from the HDT (mentioned in Chapter 3). Each rule in this grammar must have a precision of at least 70%. A higher cut-off gives us rules with better precision, but less recall, and it is the reverse for lower cut-off limits. Though these rules do not have all the lexical cues of a language, the hope is that, when applied on a large-scale web corpus, the correct matches (sketches) of the rules automatically become statistically more frequent, and hence more significant.

6.4 Error Analysis

The word sketches may not always be accurate due to the ambiguous nature of rules, and pos tagging errors. For example, when a rule such as

1:[tag="NN"] [tag="PSP: ने"] 2:[tag="VM"]

is applied on sentences which have nouns ending with the honorific जी in the data, जी is likely to be a collocate of the words. This is an error due to POS tagger that we use to tag the HindiWaC. The tagger tags the honorific जी as **NN** and not **RP**. These word sketches can be improved further by improving the POS-tagger, or the grammar, for example by involving local word group information. The other related errors are due to **UNK** pos tag which is generally assigned to words that are unknown to the tagger. As the length of a rule increases so does its sparsity, i.e. the number of sentences on which the rule can be applied since all the POS tags in the rule have to occur in that particular order. Hindi being a free word order language the possibility of all the POS tags occurring in that order is even less.

As the length of a rule increases so does its sparsity, i.e. the number of sentences on which the rule can be applied since all the POS tags in the rule have to occur in that particular order.

Hindi being a free word order language the possibility of all the POS tags occurring in that order is even less.

Chapter 7

Conclusions

7.1 Data-Driven Grammar Parser

We presented a simple unlexicalised grammar driven parser using POS tag sequences. Although the accuracies of this parser is far from the accuracies of the state of the art parser, those are still decent accuracies gained by just considering the POS tag sequences.

7.2 Malt Parser Extension

We integrated features from the grammatical parser described above into Malt Parser. Our features are found to outperform the state-of-the-art Malt Parser feature set for both English and Hindi. We obtained a significant improvement over Malt Parser's unlexicalised baseline using our syntactic rules, and since Malt is a lexicalised parser, this indicates an improvement in the robustness of Malt Parser.

This work can be extended to other languages to see if these observations hold on them as well. We could also focus on building compact grammars using regular expressions since our current grammar is prone to data sparseness as the length of rule increases. Yet, we were able to improve long distance dependencies.

7.3 Hindi Word Sketches

Corpora are playing an increasing role in all kinds of language research as well as in language learning, lexicography, translation, literary studies and discourse analysis. The requirements are, firstly, a suitable corpus, and secondly, a corpus query tool. We have presented HindiWaC, a large corpus of Hindi, which has been prepared for use in the Sketch Engine. We have described how we used Hindi Dependency Treebank to develop the grammar underlying the word sketches. And we have shown the core features of the Sketch Engine, as applied to Hindi.

Our current grammar is prone to data sparseness as the length of rule increases. A future direction of this work could be on building compact grammars using regular expressions. Additionally, one could also explore the usefulness of morphological features in grammar rules to make them semantically accurate.

Related Publications

1. Anil Krishna Eragani, Varun Kuchibhotla: **Improving malt dependency parser using a simple grammar-driven unlexicalised dependency parser.** IALP 2014: 211-214

2. Anil Krishna Eragani, Varun Kuchibhotla, Dipti Misra Sharma, Siva Reddy, Adam Kilgarriff: **Hindi Word Sketches.** ICON 2014: 328-335

Bibliography

[1] Word sketch grammar. https://www.sketchengine.eu/documentation/writing-sketch-grammar. Accessed: 2022-10-08.

[2] B. R. Ambati, T. Deoskar, and M. Steedman. Using ccg categories to improve hindi dependency parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 604–609, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

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

[4] E. Charniak. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 132–139. Association for Computational Linguistics, 2000.

[5] M. Collins. Head-driven statistical models for natural language parsing. *Computational linguistics*, 29(4):589–637, 2003.

[6] G. Coppola and M. Steedman. The effect of higher-order dependency features in discriminative phrase-structure parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 610–616, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

[7] J. Hall, J. Nilsson, J. Nivre, G. Eryigit, B. Megyesi, M. Nilsson, and M. Saers. Single malt or blended? a study in multilingual parser optimization. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 933–939, Prague, Czech Republic, June 2007.

[8] S. M. Kim, D. Ng, M. Johnson, and J. Curran. Improving combinatory categorial grammar parse reranking with dependency grammar features. In *Proceedings of COLING 2012*, pages 1441–1458, Mumbai, India, December 2012. The COLING 2012 Organizing Committee.

[9] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[10] R. McDonald. *Discriminative learning and spanning tree algorithms for dependency parsing*. PhD thesis, Philadelphia, PA, USA, 2006.

[11] R. McDonald, K. Crammer, and F. Pereira. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 91–98, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

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

[13] J. Nivre, J. Hall, S. Kübler, R. McDonald, J. Nilsson, S. Riedel, and D. Yuret. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932, 2007.

[14] J. Nivre and R. McDonald. Integrating graph-based and transition-based dependency parsers. In *Proceedings of ACL-08: HLT*, pages 950–958, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[15] K. Sagae and A. Lavie. Parser combination by reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 129–132, New York City, USA, June 2006. Association for Computational Linguistics.

[16] K. Sagae, Y. Miyao, and J. Tsujii. Hpsg parsing with shallow dependency constraints. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 624–631, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[17] Y. Zhang and J. Nivre. Analyzing the effect of global learning and beam-search on transition-based dependency parsing. In *Proceedings of COLING 2012: Posters*, pages 1391–1400, Mumbai, India, December 2012. The COLING 2012 Organizing Committee.