Application of Scattered Context Grammar for Resolving Dependency in Hindi Language

Thesis submitted in partial fulfilment of the requirements for the award of degree of

Master of Engineering

In

Computer Science & Engineering

Submitted By

Manisha Singla

(801532029)

Under the supervision of:

Dr. Ajay Kumar

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA – 147004

July 2017

I hereby certify that the matter which is being presented in the seminar report titled, "***Application of Scattered context grammar for resolving dependency in Hindi Language***", in partial fulfilment of the requirements for the award of degree of Master of Engineering in *Computer Science and Engineering* submitted in ***Computer Science and Engineering*** Department of Thapar University, Patiala, is a survey carried out by me, under the supervision of ***Dr. Ajay Kumar***. The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

Manisha

Manisha Singla
801532029
ME (CSE)

This is to certify that the above statement made by the candidate is correct and true to my knowledge.

Kumar

Dr. Ajay Kumar
Assistant Professor
Department of CSE
Thapar University
Patiala

First of all, I would like to thank the Almighty, who has always guided me to work on the right path of life. This work would not have been possible without the encouragement and able guidance of my supervisor **Dr. Ajay Kumar.** I thank my supervisor for his time, patience, discussions and valuable comments. His enthusiasm and optimism made this experience both rewarding and enjoyable.

I am equally grateful to **Dr. Maninder Singh**, Head of Computer Science and Engineering Department, a nice person, an excellent teacher and a well-credited researcher, who always encouraged me to keep working well and always advised me with his invaluable suggestions. I will be failing in my duty if I do not express my gratitude to **Dr. S.S. Bhatia**, Dean of Academic Affairs, for making provisions of infrastructure such as library, computer labs equipped with Internet facilities, immensely useful for the learners to equip themselves with the latest in the field. I am also thankful to the entire faculty and staff members of Computer Science and Engineering Department for their direct-indirect help, cooperation, love, and affection, which made my stay at Thapar University memorable.

Last but not least, I would like to thank my family whom I dearly love and without whose blessings none of this would have been possible. To my parents, I own thanks for their care and encouragement. I would also like to thank my brother since he insisted that I should do so. I would also like to thank my close friends for their constant support.

Manisha

(Manisha Singla)

Abstract

Parsing Indian languages has always been a challenging task. In recent years there have been various approaches explored for improving parsing accuracy for Hindi and other Indian languages. In this work, we present our experiments to improve dependency parsing for Hindi language. The Hindi language is considered amongst the richest morphological language because of the various forms generated from a single word form. In this paper, dependencies among various words in Hindi sentences are entertained. Networkx is used over here to generate the dependencies between the words. The dependency graph generated from the networkx clearly defines the dependencies between the words. These dependencies are converted to their plural forms using the rules to convert various singular word forms to their plurals. So this work also includes the rules which are to be considered while converting nouns, adjectives and verbs from their singular form to plural form using Transducers.

The second part of the thesis includes the use of scattered context grammar which is used in representing the dependencies among the words. These dependencies are generated when nouns, verbs and adjectives are converted from singular to their plural forms. Scattered context grammar is used to represent these dependencies using its generalised production rules. This study also presents the production rules in which sentences are transformed on the basis of neither-nor clauses, Interrogative clauses, existential clauses, sentences having question tag and generation of grammatical sentences. So this paper also covers a step higher of scattered context grammar which is transformational scattered context grammar in this work.

Keywords: Serial and cross-serial Dependency, Morphology, Transducers, Scattered context grammar, Transformational scattered grammar.

Table of Contents

List of Figures

List of Tables

List of abbreviations

SCG	Scattered context grammar
MST	Minimum Spanning Tree
LIBSVM	A Library for Support Vector Machines
LIBLINEAR	A Library for Large Linear Classification
POS	Part of speech
NP	Noun Phrase
VP	Verb Phrase
MIRA	Margin Infused Relaxed Algorithm
ICON	International conference on Natural Language Processing
LAS	Labelled Attachment Score
UAS	Unlabelled Attachment Score
LA	Label Accuracy
CCG	Combinatory categorical grammar
S-Constraint	Soft Constraints
SOV	Subject-Object-Verb
SMT	Statistical Machine Translation
BLEU	Bilingual Evaluation Understudy
MACA	Morphological Analysis Converter and Aggregator
BSD	Berkeley Software Distribution
XML	Extended Markup Language
IL	Indian Languages
DL	Dependency Labels
DNA	Deoxyribonucleic acid
RNA	Ribonucleic acid

Chapter 1
Introduction

1.1 Scattered Context Grammar (SCG)

Context free grammar and context sensitive grammar are used for representing dependencies. Context free grammar has a limitation of not being able of defining languages such as $\{ ww \mid w \in \{0,1\}^* \}$. Context free grammar are also having the limitation that these grammar cannot transmit the information to those words which are far apart from each other (i.e. non- adjacent words) in the sentence. Context sensitive grammar are too powerful for this work but these are difficult to apply. In generating a sentence from a language, a context-sensitive grammar may send a nonterminal symbol back and forth through the sentence to transmit information [1]. So the above scenario gives birth to the necessity of rewriting the grammar in such a sense that resending of non-terminal back and forth through the sentence to transmit information is avoided. S. Greibach and J. Hopcroft were the first who introduced scattered context grammar in 1969 which is a generalization of context free grammars. Scattered context grammar is the grammar which specifies both the natural language as well as the programming languages. Each and every production of this grammar is having n-context free productions which are applied in parallel to the current sentential form. There are several advantages of using scattered context grammar as these grammars can even describe some of the context sensitive languages. This makes it easy to avoid the use of context sensitive grammar from certain places to reduce the complexity. A scattered context grammar is a quadruple $G = (V, T, P, S)$ where

- V is a total alphabet;
- T an alphabet of terminals;
- P is a finite set of rules of the form

$$(A_1, \ldots A_n) \rightarrow (x_1, \ldots x_n)$$

Where $n > 1$, $A_n \in V - T$, and $x_i \in V^*$ for all i, $1 \leq i \leq n$ (each rule may have different n);

• S ∈ V -T is the start symbol.

If $u=u_1A_1 \ldots u_nA_nu_{n+1}$

$v=u_1x_1 \ldots u_nx_nu_{n+1}$

And p= $(A_1 \ldots \ldots A_n)$, Where $u_i \in V^*$ for all i, $1 \leq i \leq n+1$, then G makes a derivation step from u to v according to p, symbolically written as

$$u \Rightarrow_G v[p]$$

Or, simply, $u \Rightarrow_G v$. Set

$$lhs\ (p) = A_1 \ldots A_n$$

$$rhs\ (p) = x_1 \ldots x_n$$

And $$len\ (p) = n$$

If $len(p) \geq 2$, p is said to be a context-sensitive rule while for $len(p) = 1$, p is said to be context-free. The language of G is denoted by L (G) and defined as

$$L\ (G) = \{w \in T^* | S \Rightarrow^*_G w\}$$

L is a scattered context language if there exists a scattered context grammar G such that $L = L(G)$. As scattered context productions are having n- context free rules which are to be applied in parallel to the current sentential form [1], so this grammar can also devise some of the languages of context sensitive grammar. This point of scattered context grammar can be better understood by an example given below:

Example 1.1[2]: A scattered context production

$$(A, B, C) \rightarrow (aA, bB, cC)$$

Is having three context free productions,

$$A \rightarrow aA, B \rightarrow bB, C \rightarrow cC$$

which are to be accessed parallel in order to make use of scattered context grammar. This is applicable to the current sentential form $aAbBcC$ so that all the context free components can run in parallel and the non-terminals on the left-hand side are in the same order as they appear in the current sentential form. As a result,

$$aAbBcC \Rightarrow aaAbbBccC$$

So, it can be easily deduced that by adding production $(S \rightarrow ABC)$ and $(A, B, C) \rightarrow (a, b, c)$, where S is the start symbol of the grammar, non-context free grammar $(a^n b^n c^n : n > 1)$ can be easily derived.

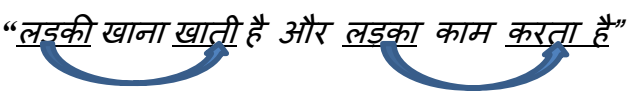
1.2 Dependency Graphs

Dependency graphs are used to represent words in a sentence and their relationship to syntactic modifiers with the help of directed edges. There are two classes of dependency graphs

- Projective (Non- crossing edges)
- Non- Projective (Crossing edges)

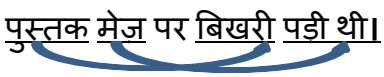
Projective dependency is the dependency among the words of a sentence which is shown using edges and those edges must not cross each other. Assuming a unique root as the left most word in the sentence, a projective graph is one that can be written with all words in a predefined manner. Example 1.2.1 belongs to the special class of dependency graphs that contains projective (also known as nested or non-crossing) edge in a linear order. In projective graphs, edges are drawn on the plane above the sentence, with no edge crossing each another [3]. It can be said that a dependency graph is projective if and only if an edge from word x to word y implies that there exists a directed path in the graph from x to every word between x and y in the sentence.

Example 1.2.1:



Non- Projective dependency is the dependency among the words of a sentence which is shown using crossed edges i.e. two or more than two edges intersect with each other. Example 1.2.2 belongs to the non- projective dependency parsing of the Hindi sentence since in this example edges are crossing each other.

Example 1.2.2:



This example represents the non-projective parsing since dependent edges are crossing each other.

1.3 Hindi Dependency parsing

The Hindi language is considered amongst the richest morphological language because of the various forms generated from a single word form. Hindi dependency parsing is originated because of dependencies in Hindi words in the sentences. These dependencies need to be analysed in order to generate rules corresponding to them. Parsing of these dependencies is very much necessary so that the generalized rules corresponding to these dependencies can be formalized. In this work, dependencies among various words in Hindi sentences are entertained. These dependencies can be viewed while converting singular sentence to its plural form. These dependencies can also be observed while converting the tense of the sentence. Another way by which dependency can be observed is by changing the noun i.e. masculine noun to feminine noun. This work includes the rules where dependencies are to be considered while converting nouns, adjectives and verbs from their singular form to plural form using Transducers. Hence the application of scattered context grammar is explored in using nouns, adjectives and verbs in the sentences. This work also covers a step higher of scattered context grammar which is transformational scattered context grammar in the transformation of Hindi sentence.

Hindi language currently used by millions of people but people are not able to make correct Hindi sentences and its morphology. We describe and discuss various constructions and transformation of grammatical Hindi sentences which can be made correctly using linguistic study concerning Hindi language. Here we have represented serial and cross serial dependencies in Hindi language in the form of nouns, verbs and adjectives in Hindi. It leads to correct transformation when the sentence is converted from singular to plural. Usually cross serial dependencies are handled by context sensitive grammar.

There are various Hindi dependency parsers which can be used for determining the dependencies among the words in Hindi sentences. We have used Hindi Dependency Parser for representing the dependencies among those words. We can also make use of other parsers for the dependency generation in Hindi words in the sentences and these parsers are defined below.

1.3.1 Various types of Hindi dependency parser

Following are some of the parsers which can be used for Hindi dependency parsing:

1. Malt Parser

2. Shallow Parser

3. MST Parser

4. Hindi dependency parser

5. Turbo Parser

Explanation:

- **Malt Parser**

Malt Parser is a system designed for data-driven dependency parsing. It can be used to generate a parsing model from treebank data and to parse new data (in case of testing) using the induced model. Malt Parser is developed by Johan Hall, Jens Nilsson and Joakim Nivre at Växjö University and Uppsala University, Sweden [4]. Unlike traditional parser-generator constructs a parser on the basis of given grammar, this data driven dependency parser is used to build a parser on the basis of treebank. This parser is an implementation of induced machine learning, where the syntactic analysis of the sentences is used for the parsing of unseen data. It is used to guide the parser at some non- deterministic points. The methodology used for parsing by malt parser is based upon following three important components:

1) It is used for building labeled dependency graphs.
2) It is used for predicting next parsing action at non-deterministic points.
3) It is used for discriminative learning to map History to parsing actions [4].
4) Malt parser is built under Java platform and can also be used for the parsing of various other languages like English, Chinese, Tamil and Bangla etc.

Malt parser can be used to implement following eight deterministic parsing algorithms namely:

1) Nivre arc-standard
2) Nivre arc-eager
3) Stack swap-eager
4) Covington non-projective
5) Stack swap-lazy
6) Covington projective

7) Planar

8) 2-Planar

Following two Machine Learning packages are included in Malt Parser [4]:

- LIBSVM - A Library for Support Vector Machines
- LIBLINEAR - A Library for Large Linear Classification

There is one limitation of MaltParser i.e. it is deterministic by nature. It can be considered as an advantage somewhere but mostly deterministic nature leads to error propagation. As MaltParser is deterministic, it cannot go back if at some point it makes any wrong decision.

- **Shallow Parser**

Shallow Parser analyzes the sentence to identify constituents in it. These constituents include noun groups, verb groups, verbs etc. but cannot be used for specifying internal structure. These are also not used for the identification of any role in the main sentence. It is similar to the lexical analysis for computer languages. The shallow parser is used to convert sentences into grouping of words called phrases, and their relations.

Following are the tasks performed by shallow parser:

1. Part of speech (POS) tagging
2. Chunking
3. Relation finding

Part-Of-Speech tagging involves morpho-syntactic class of the words (nouns, verbs etc.). Chunking involves those words which can be grouped like NP, VP etc. and relation finding describes relations with the main verb like Subject, Object etc. Shallow parsers are automatically built with the help of machine learning techniques.

- **MST Parser**

MST parser is used for non-projectivity which searches for maximum spanning tree over some directed graphs although it also supports projective parsing. It consists of a graph-based algorithm which finds the parse tree which is having maximum score from all possible outputs. All the possible complete trees are explored to get the maximum score. The Models of dependency structure in MST parser are based upon large-margin discriminative training methods. This parser is also implemented in Java.

The build system of MST Parser is based upon Apache Ant. Ant is little but handy tool that uses a build file written in XML (build.xml) as building instructions [5].

Following are the features of MST Parser:

1. It estimates edge-wise confidence.
2. It uses perceptron and k- best MIRA training.
3. It supports first and second order projective and non-projective parsing.

The Arc-factored model of MST parser can be illustrated from [6].

Following are the disadvantages of MST Parser:

a) It takes more time in learning.

b) It takes more time in parsing the given sentence.

c) It is more constrained in terms of choice of features.

- **Hindi Dependency Parser**

Hindi dependency parser uses MALT parser for its implementation. Malt Dependency Parser is trained and tested on Hindi ICON 2010 shared task data in Hindi dependency parser. The features word, lemma and postag id are used in this parser to train parser. This parser can be build better parser using other features like morphological information etc. Accuracy is described below in Figure 1.1.

```
Parser Accuracy: (Parser trained on features word, lemma and postag)

  Labelled   attachment score: 4948 / 6588 * 100 = 75.11 %
  Unlabelled attachment score: 5555 / 6588 * 100 = 84.32 %
  Label accuracy score:       5199 / 6588 * 100 = 78.92 %
```

Fig 1.1: Accuracy estimation in Hindi dependency parser [7]

- **Turbo Parser**

Turbo parser is an example of a constraint based parser. It constructs the problem of non-projective dependency parsing as a polynomial-sized integer linear program. It can also encode prior knowledge as hard constraints. It can also learn soft constraints from data. In particular, its model is able to learn correlations among neighboring arcs, word valency, and tendencies toward projective parses. The model parameters are learned in a max-margin framework by using a linear programming relaxation. Turbo parser will train a second-order non-projective parser with features for arcs, consecutive siblings and grandparents. The default training algorithm is cost-augmented MIRA (Margin-infused relaxed algorithm), but it also provides other options. The decoder uses AD3 algorithm. It will also train a probabilistic model for

unlabeled arc-factored pruning, which speeds up parsing by reducing the number of possible arcs. In situations where learning speed is essential than accuracy, a simple arc-factored model can be preferred. In this mode, it speeds up the process by not applying first order pruner. Turbo parser provides following three settings for training: basic, standard and full.

• Basic: It enables arc-factored parts.

• Standard: It enables arc-factored parts, consecutive sibling parts and grandparent parts.

• Full: It enables arc-factored parts, consecutive sibling parts, grandparent parts, arbitrary sibling parts and enables head bi-gram parts [6].

Chapter 2
Literature Survey

2.1 Related Work

In this chapter, we will discuss work done by researchers on dependency parsing, regulated grammar and scattered context grammar. There is a significant amount of work done in the area of dependency parsing on various languages like Dutch, Hindi, Telgu and Bangla etc. In this chapter, firstly we will discuss the work on dependency parsing and later on we will discuss the work on scattered context grammar. To illustrate the dependencies among the words in the sentences and to describe its parsing, various dependency parsers were used by the researchers.

Singla et al. [8] presented in their paper an approach towards Hindi dependency parsing as a part of Hindi Shared Task on Parsing, COLING 2012. Their approach included the effect of using different settings which is available in Malt Parser. It is further followed by the two-step parsing strategy which splits the data into interChunks and intraChunks to obtain the best accuracy. Their system achieved best LAS of 90.99% for Gold Standard track and second best LAS of 83.91% for automated data. The dataset used for the experiment consists of 1204 sentences, 26,809 tokens and 22.27 average sentence lengths in the training data. While for Development dataset, number of sentences are 1233 and tokens generated are 26,416 and average sentence length for this is 21.42. For testing data, number of sentences are 1828 and tokens for the same are 39,975 with average sentence length is 21.87.

Kosaraju et al. [9] presented in their experiments the dependency parsing on three languages including Hindi, Bangla and Tamil and these languages are rich in morphology and they also explained that how these languages differ in their morphological richness. They explored the Malt parser over a large feature pool with different strategies of parsing. They showed the utilization of shallow parser for Hindi language. They also attempted to add semantics in parsing. They reported their results on the test data provided at ICON tools contest, 2010.

Ambati et al. [10] showed that dependency parsing of Hindi can be improved from a strongly lexicalized formalism like Combinatory Categorical Grammar (CCG). They first described an easy way to get a CCG lexicon and treebank from an existing dependency Treebank with the help of a CCG parser. They used the output of a

supertagger which was trained on the CCG bank as a feature for Hindi dependency parser (Malt). Their results showed that with the help of CCG categories, accuracy of Malt on long distance dependencies can be improved.

Bresnan and Kaplan [11] studied the cross serial dependencies in Dutch language and found that these cross dependencies cannot be context free.

Fernau and Meduna[12] proved that a scattered context grammar can generate every recursively enumerable language with reduced number of context sensing productions and also the reduced number of non- terminals. They improved their work by proving that scattered context grammar can generate every recursively enumerable languages with not more than two context sensitive productions [13].

Husain et al. [14] investigated challenges in IL parsing by making annotated data available to the larger community. The ICON10 tools [11-12] contest was devoted to the task of dependency parsing for Indian languages (IL). Following statistic was used for the training data:

- 1400 sentences, 7602 word counts and 5.43 average sentences for Telgu language
- 980 sentences, 10305 word counts and 10.52 average sentences for Bangla language
- 1500 sentences, 28522 word counts and 19.01 average sentences for Hindi language

The testing set had 150 sentences for all the three languages. The performance of the system was supposed to be measured in terms of various standard measures like Unlabelled attachment score (UAS) and Labelled attachment score (LAS). Following average scores over the 2 rounds for different languages are mentioned below:

- UAS is 84.7, LAS is 59.7 and LS is 61.41 for Telgu language
- UAS is 81.9, LAS is 66.9 and LS is 71.1 for Bangla language
- UAS is 88.7, LAS is 72.8 and LS is 75.59 for Hindi language

Following are the result of the best performing system from the above experiment reported by them is as shown below:

- UAS is 85.7, LAS is 65.5 and LS is 66.21 for Telgu language
- UAS is 90.3, LAS is 84.2 and LS is 85.85 for Bangla language
- UAS is 90.2, LAS is 79.3 and LS is 81.66 for Hindi language

Meduna [15] connects the theoretically oriented discussion of scattered context grammar with the pragmatically oriented discussion of scattered context grammar in English linguistics. Scattered context grammar is also modified to its transformational version so that it can be easily applied to the syntax related modifications of the sentences [15].

For example,

He works well in computer science.

If in the above sentence, *"He",* changes to *"I",* the sentence would be illegal to write if it is written as

I works well in computer science.

Because the above sentence is illegal from grammatical point of view, so changing "he" to "I" would lead to change from "works" to "work" and the sentence would be written as:

I work well in computer science.

So there is some dependency in these words of the sentence. This can be handled by context free grammar if these dependent words can be adjacent but if these words are not adjacent to each other, then context sensitive grammar is required. This grammar requires many rules so Meduna proposed the use of scattered context grammar that comes between context free grammar and context sensitive grammar in terms of generative power. For example,

He usually works well in computer science.

This sentence can be changed into

I usually work well in computer science.

And the scattered context production for this change can be

$$(He, works) \rightarrow (I, work)$$

Meduna also proposed the use of transformational context grammar in English linguistics and made an assumption regarding the set of all English words that this set is fixed and finite.

2.1.1 Use of transformational scattered grammar

Example of various sentences in which transformational scattered context grammar is proposed are:

- Neither and nor clauses

  Example 2.1[15]: Application of transformational Context grammar used in *neither and nor* clauses is as shown below:

  Neither Ram nor his son went to the party

  Transformational scattered grammar helped in the negation of the above sentence in the following manner and Meduna [15] described the corresponding production rule in this way:

  Both Ram and his son went to the party.

  And the corresponding rule is:

  (<Neither>, <Nor>)→(<Both>, <And>)

- Interrogative clauses

  Example 2.2 [15]: In interrogative clauses, transformational scattered context grammar is used in two different ways depending upon the predictor in the sentence. If the predictor is an auxiliary verb, then the interrogative sentence can be generated by just swapping the predictor and the subject in the sentence as shown through an example below.

  He is eating the meal

  Here the predictor is auxiliary verb so its interrogative clause would be:

  Is he eating the meal?

  But Meduna [15] noticed that if the predictor is not an auxiliary verb rather it is a lexical verb, then the interrogative clause is generated by inserting dummy *do* at the beginning of the sentence.

  Example 2.3[15]:

  She eats up early in the morning

  And the interrogative clause for this would be

  Does she eat up early in the morning?

And the transformational scattered rule corresponding to the above sentence is:

$$\{(< p >, < \pi_{3rd}(v) >) \rightarrow (does\ \ p, vX) |\ v\ \in T_{Vpl} - T_{VA}, p \in T_{PPn}\}$$

- Generation of grammatical sentences

Meduna stated that a scattered context grammar can generate an infinite number of context free grammar with a subset of English language.

Example 2.4[15]:
My grandparents are all my grandfathers and grandmothers.

My great-grandparents are all your great grandfathers and great grandmothers. Or

Your great -great- grandparents are all your great -great -grandfathers and great -great -grandmothers. And so on.

So the transformational scattered context rule for this is:

$L = \{ your\{great\}^i grandparents\ are\ all\ your\ \{great-\}^i$
$grandfathers\ and\ all\ your\ \{great-\}^i\ grandmothers\ | i \geq 0 \}$

- Sentences having Question tags

Meduna also gave the use of transformational scattered context grammar in these sentences with question tags.
Example 2.5[15]:

I am always ready

The same sentence with question tags can be written as

I am always ready, aren't I?

And the corresponding rule is

$$(< I >, < am >, < ready >) \rightarrow (I, am\ X, Y\ ready\ aren't\ I)$$

- Existential clauses

Example 2.6 [15] : Meduna phrased that while using existential clause in the sentence, there used to be a dummy subject as shown in the example.

There was a doctor present

Here “there” is a dummy subject and the whole sentence can be rephrased as:

A doctor was present

So the corresponding transformational rule given is:

$$(<a>, <was>) \rightarrow (\text{There was a } X, \in)$$

Yeleti and Deepak [16] described dependency parsing using the overall design of a new two stage constraint based hybrid approach. They defined the two stages and showed how at appropriate stages, different grammatical construct are parsed. This division made by them leads to a selective identification and resolution of specific dependency relations at both the stages. Also they showed how to use the hard constraints and soft constraints which can help them in building an efficient and robust hybrid parser. Lastly, they evaluated the implemented parser on ICON tools contest using Hindi data. They observed that the best Labeled and unlabeled attachment accuracies for Hindi data are 62.20% and 85.55% respectively. The parameters those were finalized for the experiment are as described below.

- Value of k is equal to 2
- Best S-constraint(Method 1) is {C-POS, P-POS}
- Value of N is 2000
- Margin initialization is {C-POS, P-POS}

They have tried all the three methods on the development data and method 2 [16] gave them the best results on the development data. So they gave the output of method 2 on the testing data for the evaluation for the tools contest. The second method is used starting with the best S-constraint {C-POS, P-POS}. Their results on the Hindi test data comprises of UA value 85.55, LA value 62.20 and L value 65.88.

Bhat et al. [17] in their article enhanced the transition-based parsing of Hindi and Urdu by modifying and restating the features and feature extraction procedures that were earlier proposed in the parsing literature of Indian languages. They proposed and effectively showed that properly including syntactically relevant information like complex predication, case marking, and grammatical agreement in an arc-eager parsing model can effectively improve parsing accuracy. Their experiments exactly showed an absolute improvement of ∼2% LAS for parsing of both Hindi and Urdu over a competitive algorithm which used rich features like part-of-speech (POS) tags,

cluster ids, chunk tags and lemmas. They also proposed some methodology to identify ezafe constructions in Urdu texts which also showed promising results in parsing of these constructions. Following are the statistics used in training, testing and development in Hindi language in their experiment:

- Count of tokens for training is 3, 47,744, for testing it is 43,556 and for development it is 43,556.
- Count of chunks for training is 1, 87, 029, for testing it is 23,418and for development it is 23, 417.
- Count of sentences for training is 16,629, for testing it is 2,077 and for development it is 2,077.

Following are the statistics used in training, testing and development in Urdu language:

- Count of tokens for training is 1, 53,317 for testing it is 19,065 and for development it is 19,065.
- Count of chunks for training is 72,319, for testing it is 9,010 and for development it is 9,010.
- Count of sentences for training is 5,432, for testing it is 677 and for development it is 677.

The performance Bhat et al. [17] reported about POS Taggers, Morphological Analyzers and Chunkers on Hindi Test Set is as described below:

- Lemma % comes out to be 90.65 with GN% is 93.46 in case of morphological analysis and it was 96.02% for POS tagging. It had accuracy of 98.87% for Gold and 97.50% for predicted accuracy in chunking.

The performance Bhat et al. [17] reported about POS Taggers, Morphological Analyzers and Chunkers on Urdu Test Set is as described below:

- Lemma % comes out to be 88.12 with GN% is 89.87 in case of morphological analysis and it was 92.92% for POS tagging. It had accuracy of 97.70% for Gold and 95.250% for predicted accuracy in chunking.

Following are the accuracies of baseline parsing for Hindi using ZN feature template [17]:

- In case of POS tagging, UAS, LS and LAS for development is 92.82%, 89.50% and 86.89% respectively and UAS, LS, LAS for development is 92.77%, 89.51% and 86.79% respectively.
- In case of Chunking, UAS, LS and LAS for development is $93.38^{0.56}$, $89.69^{0.19}$and $87.45^{0.56}$respectively and UAS, LS, LAS for development is $93.31^{0.54}$, $89.56^{0.05}$and $87.42^{0.63}$respectively.
- In case of Lemma, UAS, LS and LAS for development is $93.39^{0.01}$, $89.95^{0.26}$and $87.61^{0.16}$respectively and UAS, LS, LAS for development is $93.24^{-0.07}$, $89.73^{0.17}$and $87.52^{0.10}$respectively.
- In case of Clusters, UAS, LS and LAS for development is $93.51^{0.12}$, $90.21^{0.26}$and $87.82^{0.21}$respectively and UAS, LS, LAS for development is $93.52^{0.28}$, $90.01^{0.28}$and $87.77^{0.25}$ respectively.

Following is the Baseline parsing accuracy of Urdu language using ZN feature template [17]:

- In case of POS tagging, UAS, LS and LAS for test is 88.19%, 84.44% and 80.16% respectively and UAS, LS, LAS for development is 88.14%, 84.41% and 80.57% respectively.
- In case of chunking, UAS, LS and LAS for test is 88.55^{0}, $84.70^{0.26}$ and $81.01^{0.41}$ respectively and UAS, LS, LAS for development is $88.46^{0.32}$, $84.59^{0.18}$ and $81.00^{0.43}$respectively.
- In case of Lemma, UAS, LS and LAS for test is $88.70^{0.15}$, $84.80^{0.10}$ and $81.13^{0.12}$ respectively and UAS, LS, LAS for development is $88.69^{0.23}$, $84.77^{0.18}$ and $81.11^{0.11}$ respectively.
- In case of Clusters, UAS, LS and LAS for test is $88.81^{0.11}$, $84.88^{0.08}$and $81.34^{0.21}$ respectively and UAS, LS, LAS for development is $88.77^{0.08}$, $84.84^{0.07}$ and $81.29^{0.18}$ respectively.

They showed the improved parsing of Hindi and Urdu language with the help of improved accuracy and the evaluation techniques they used are:

- Leave-one out evaluation
- Only-one evaluation

Accuracies for Hindi language on both the evaluation technique is as described below [17]:

- For POS it is 86.43 and 86.89 for leave-one out and only one evaluation technique respectively.
- For Chunk it is 86.98 and 86.06 for leave-one out and only one evaluation technique respectively.
- For Lemma it is 87.14 and 83.69 for leave-one out and only one evaluation technique respectively.
- For clusters it is 87.61and 85.09 for leave-one out and only one evaluation technique respectively.

Accuracies for Urdu language on both the evaluation technique is as described below [17]:

- For POS it is 77.24 and 80.60 for leave-one out and only one evaluation technique respectively.
- For Chunk it is 77.96 and 80.10 for leave-one out and only one evaluation technique respectively.
- For Lemma it is 80.22and 77.85for leave-one out and only one evaluation technique respectively.
- For clusters it is 81.13 and 77.06 for leave-one out and only one evaluation technique respectively

Xu et al. [18] introduced the precedence reordering approach which was based on a dependency parser to statistical machine translation systems. Their method can efficiently add linguistic knowledge into SMT systems without increasing the complexity of decoding which is similar to other preprocessing reordering approaches. But they showed significant improvements in BLEU scores for a set of five subject-object-verb (SOV) order languages, when translating from English, compared to other reordering approaches, in state-of-the-art phrase-based SMT systems [18]. Following training corpus statistic of their experiment for 5 SOV languages is as described below [18]:

- They used 303M for source and 267M for target for English to Korean system.
- They used 316M for source and 350M for target for English to Japanese system.
- They used 16M for source and 17M for target for English to Hindi system.
- They used 17M for source and 19M for target for English to Urdu system.

- They used 83M for source and 76M for target for English to Turkish system.

For all 5 languages, they achieved statistically significant improvements in BLEU scores over a state-of-the-art phrase based baseline system [18].

McDonald et al. [19] made it possible to search over all the projective trees in $O(n^3)$ through their representation and their representation made it possible to extend the parsing to non- projective contexts also. They evaluated their method on Prague dependency Treebank and finally showed that the accuracy and efficiency is increasing using MST Parser for non- projective dependency parsing. Zeman and Resnik [20] described an approach of adapting a parser on a new language. This new language is not so good in terms of linguistic resources although the source language is. This approach is tested over two European languages because of availability of test set. This approach can be easily applicable to other pairs of related languages also.

Ambati et al. [21] showed the effects of minimal semantics on parsing [21]. They used Malt Parser and MST Parsers as the data driven parsers for their experiment. The work was performed over Hyderabad Dependency Treebank. Because of minimal semantics, they achieved an increase of 2.01% and 1.65% in labeled accuracy and labeled attachment score respectively over state-of-the-art data driven dependency parser.

Tsarfati et al. [22] replicated the use of English parsing to some other languages also. Those parsing languages which are having complex word structure but flexible word order had been shown to capture non-trivial adaptation.

Husain et al. [23] proposed a modular cascaded approach for data driven parsing model. They introduced an artificial root node in the dependency structure of a sentence. They did this by catering to distinct dependency label sets. These label sets reflected the function of the set internal labels through a distinct and identifiable linguistic unit, at different layers.

Kolachina et al. [24] applied MaltParser over Hindi languages. They motivated the use of arc-directionality features which is based on empirical evidence. These evidences were in the form of statistics which were collected over the treebanks. In this paper, the outputs obtained for all the three languages like Hindi, Bangla and Telgu are blended together.

Kosaraju et al. [25] presented two approaches i.e. statistical and rule based for annotating intra-chunk dependencies in Hindi. The annotator of intra-chunk dependency annotator provided a fully parsed dependency tree for a Hindi sentence.

Hockenmaier et al. [26] attempted to translate the Penn Treebank into CCG (Combinatory categorical grammar). They founded that the resulting corpus of CCG bank included 99.4% of the sentences in the Penn Treebank.

Parsing of context sensitive grammar although is difficult but attempted by Rychnovsky [27]. In this paper also, scattered context grammar was used to parse some of the context sensitive grammar.

The Literature summary is as shown in table 2.1.

Table 2.1 Literature Summary

Name of the Author	Main Features and Key Findings
Gadde et al. [28]	i. To improve the parser performance, data driven dependency parsing approach is used which uses clausal information of a sentence. ii. They used modified version of MaltParser for all the experiments. iii. They achieved an overall improvement of 0.77% and 0.87% in labeled attachment and unlabeled attachment accuracies respectively over the baseline parsing accuracies.
Bharati et al. [29]	i. They proposed a two stage constraint based dependency parser for especially for free word order languages broadly. ii. They obtained that their final results are good with a labelled attachment and maximum attachment accuracy of 75% and 90.1% respectively.
Bhatt et al. [30]	i. The creation of a Hindi/Urdu multi-representational and multilayered Treebank is described in this paper. ii. The simultaneous development of dependency structure and phrase structure treebanks for Hindi and Urdu, as well as a Prop Bank is described in this paper..

Tsarfaty et al. [31]	i. They synthesized the contributions of various researchers worked on parsing languages like Arabic, Basque, French, German, Hebrew, Hindi and Korean to point out some shared solutions across languages.
Singla et al. [32]	i. Their approach included the effect of using different settings available in Malt Parser which followed the two-step parsing strategy and these are splitting the data into interChunks and intraChunks to obtain the best possible LAS, UAS and LA accuracy. ii. Their system achieved best LAS which is of 90.99% for Gold Standard track and second best LAS of 83.91% which is for automated data.
Nivre. [33]	i. Nivre described about the use of MaltParser for three languages Hindi, Telgu and Bangla. ii. They achieved an unlabeled attachment score which was nearly 90% for Hindi and Bangla, and this score was over 85% for Telugu, while the labeled attachment score was 15–25 percentage points lower for all.
Bharati et al. [34]	i. They defined the two stages of parsing and showed that how different grammatical construct are parsed at these two appropriate stages. ii. They constructed robust hybrid parser of MACA by showing the proper use of hard and soft constraints so that parser can be efficient and robust.
Kalra et al. [35]	i. They represented DNA and RNA biological sequences with the help of deep pushdown automata and state grammar. They parsed both the sequences in $O(n)$ time which is the major improvement of existing work.
Kalra et al. [36]	i. They investigated and proved that fuzzy deep pushdown automata can be constructed from fuzzy state grammars and vice versa.

Chapter 3
Problem Statement

This chapter contains gap analysis, problem statement and the methodology which is to be followed to accomplish the task.

3.1 Gap Analysis

There are various dependency parsers exists for different natural languages like English, Bangla, Telgu, Dutch and Spanish etc. Work is being carried on among the dependencies between words in these languages. Singla et al. [32] included the effects of using different settings available in Malt Parser which followed the two-step parsing strategy. These are splitting the data into interChunks and intraChunks to obtain the best possible LAS, UAS and LA accuracy. This work can also be extended for both interchunks and intrachunk dependencies by using MST Parser. Meduna [15] applied the concept of scattered context grammar on English linguistic. Therefore, their work can also be extended for Hindi language. Dependencies amongst the words play a significant role in Hindi language. Dependencies amongst the words refer to the change in one word causes the change in another word.

There are many Hindi dependency parsers which are designed to work on these dependencies of words in the sentence. Context free grammar can be used to handle dependencies between adjacent words of the sentence but it cannot handle those dependencies where infinite number of words can come between the dependent words. These dependencies are represented with context-sensitive grammar. We have reviewed serial dependency for converting singular Hindi sentences to their plural forms. Context sensitive grammar can be used for handling cross serial dependencies. These rules are required to send the information concerning the form of a word to another word, which may occur even at the opposite end of the sentences [15]. This becomes problematic to use context sensitive grammar in such a situation. So the work of Meduna [15] can be extended to facilitate Hindi linguistics as well.

3.2 Objectives

Context free and context sensitive productions are used to handle the dependencies between words in a sentence. Context free grammar is not able to represent the dependencies amongst the non-adjacent words of the sentence. These dependencies are represented using context sensitive grammar.

Based on literature survey, following objectives are outlined:

i. To analyze varieties of tools and techniques available to handle dependency.

ii. To design transducers for converting singular to plurals for nouns, verbs and adjectives when they are converted to their plural forms.

iii. To generate dependency tree or graph for representing inter chunk as well as intra chunk dependencies in Hindi language.

3.3 Methodology

Methodology to analyze the Hindi dependency using scattered context grammar is shown in figure 3.1. The figure 3.1 shows the steps which are to be performed in order to achieve the objectives.

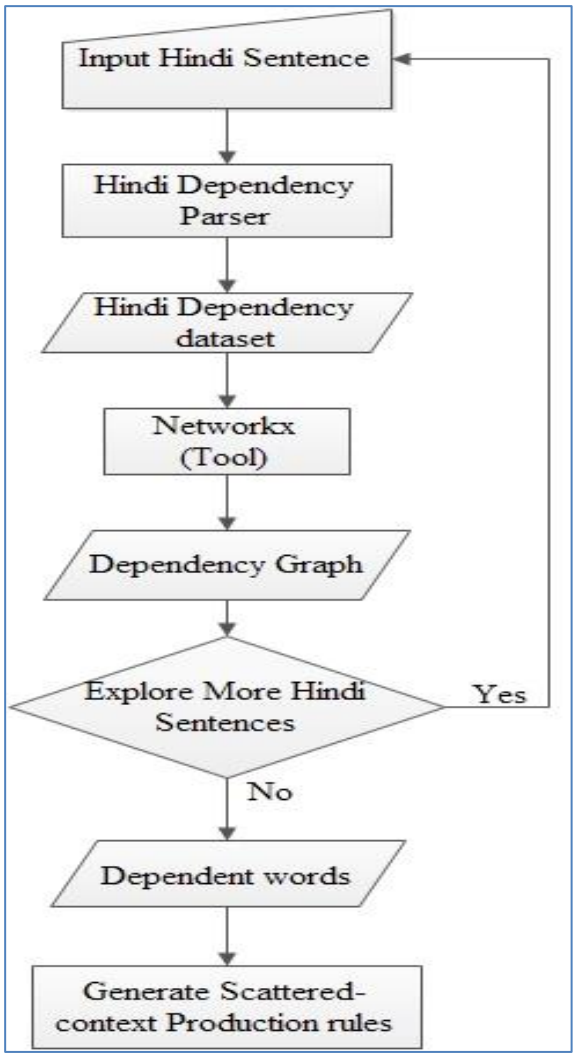


Fig 3.1: Workflow diagram

Steps to be performed:

- Hindi sentences are manually input by the user.

- These Hindi sentences are fed to the Hindi dependency Parser which is helpful in generating the Hindi dependency dataset.
- For understanding the dependencies between the Hindi words, Networkx tool is used to generate the dependency graph.
- These dependent words are converted to their plural forms and scattered context grammar is applied over these words to generate rules over these dependencies.

Chapter 4
Proposed solution

We have proposed the use of scattered context grammar for handling serial and cross-serial dependencies between words. Scattered context grammar has the generative power between context free grammar and context sensitive grammar.

Dependent sets of words needs to be generated for applying scattered context grammar. These dependent sets are analyzed using dependency graph drawn with the help of Networkx. The algorithm for designing dependency graph is as shown below in figure 4.1. First dependent sets for the Hindi sentence are revealed and then these sets are transferred to their plural forms depending upon nouns, verbs and adjectives used in the sentence. Rules for generating plurals from singular sentences are explained in next subsection.

4.1 Generation of plurals from singular sentence in Hindi Language

Rules are generated and described using transducers for nouns, verbs and adjectives in Hindi language. Singular sentences are converted to their plurals on the basis of these rules.

4.1.1 Transducer for Nouns

Type of noun i.e. feminine or masculine plays a significant role in the transformation of nouns. These rules are described by transducers in Figure 4.2. Dependency in Hindi sentences exists between the subject and the verb used in objects. Subject is generally a noun and object is a kind of action which is expressed using verbs. These dependencies are discussed in chapter 3.

4.1.2 Transducer for Verbs

Whenever a sentence is translated from singular to plural, the corresponding verb gets translated from singular verb to plural verb. In Hindi, verbs are manipulated according to gender, person, number, tenses and mood described in the Hindi sentence [38]. Tenses can be classified into present, past and future. Example of these verbs for present tense [39] is shown in Table 4.1 and the rules are similar for past and future as shown in Table 4.2 and Table 4.3 respectively. Scattered context grammar notifies the dependency in all modes and acts accordingly.

DependencyGraph (G, traverseArray, arrWords, arrLines, filedependencies (index file), f, dictword, dictLine, parentid, index, rootword and word)

1. *Start*
2. *Create two arrays parent[] and arrLines[]*
3. *Create a dictionary dictLine {}.*
4. *from manisha.output as f :(generated by Hindi dependency Parser)*
   4.1 for line in f:
   4.2 print line and
   4.3 while (line)
   4.4 Add columns in dictLine according to index, word, rootword, type, parentid.
   4.5 Append dictLine to arrLines
   4.6 filedependencies.write (dictLine ['index'] + dictLine ['word']) ***)*** *and repeat step 4.1.*
5. *Create a function traverseArray(arrWords, parentid, G)*
6. *Call traverseArray(arrLines,0,G)*
7. *traverseArray(arrWords, parentId, G):*
   7.1 for dictword in arrWords:
   7.2 if dictword [parentId] equals parentId:
   7.3 print its parent, index, rootword and word
      7.3.1 if(dictword['parentId']) not equals to zero:
      7.3.2 filedependencies.write ([parentId] →['index']
   7.4 Graph G.add_edge (dictword [parentId], dictword ['index']) and repeat step 7 call traverseArray (arrWords, dictword ['index'], G)
8. *Return Graph G.*
9. *End*

Figure 4.1: Algorithm to design dependency graph

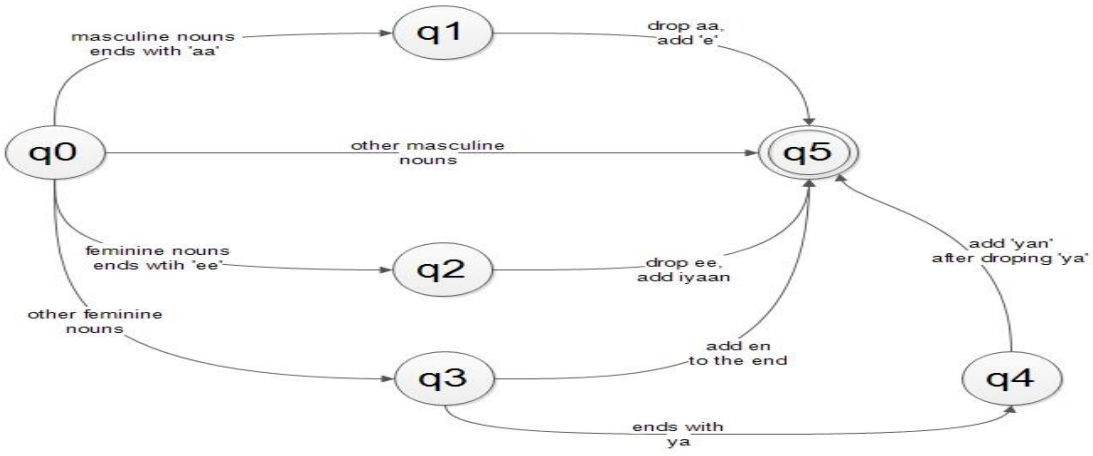


Fig 4.2: Transducer showing singular nouns to plural nouns

Table 4.1: Change of verbs in present mode

Person	Singular	Plural
1st	मैं अमीर हूँ।	हम अमीर हैं।
2nd person(casual)	तू अभी छोटा है।	तुम अभी छोटे हो।
2nd person (honors)	आप ईमानदार हैं।	आप सब ईमानदार हैं।
3rd person	वह स्कूल जाता है।	वे स्कूल जातें हैं।

Table 4.2: Change of verbs in past mode

Person	Singular	Plural
1st	मैं अमीर था।	हम अमीर थे।
2nd person(casual)	तू छोटा था।	तुम छोटे थे।
2nd person (honors)	आप ईमानदार थे।	आप सब ईमानदार थे।
3rd person	वह स्कूल जाता था।	वे स्कूल जातें थे।

Table 4.3: Change of verbs in future mode

Person	Singular	Plural
1st	मैं अमीर होऊँगा।	हम अमीर होएंगे।
2nd person(casual)	तू छोटा होगा।	तुम छोटे होगे ।
2nd person (honors)	आप ईमानदार होगे।	आप सब ईमानदार होंगे ।
3rd person	वह स्कूल जाता होगा ।	वे स्कूल जातें होंगे ।

The rules for the change in verbs from singular to plural verbs [40] can be shown with the help of transducer as shown in figure 4.3.

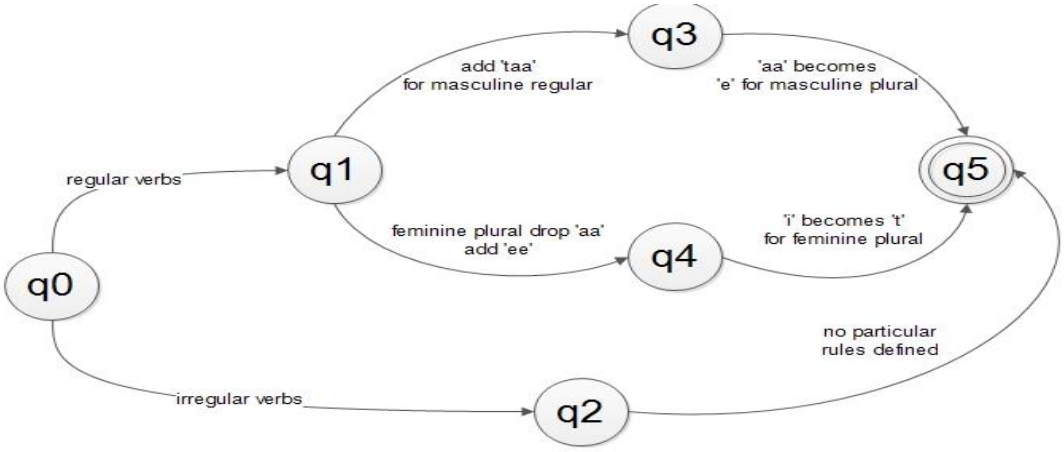


Fig 4.3: Transducer showing singular to plural verbs

4.1.3 Transducer for Adjectives

Change in adjectives also depends upon the masculine and feminine nouns used with the adjectives [41]. Adjectives changes with nouns in Hindi in contrast to English sentences. Dependency amongst the adjectives and nouns in Hindi can be analyzed using scattered context grammar and the rules are shown in figure 4.4 with the help of transducer.

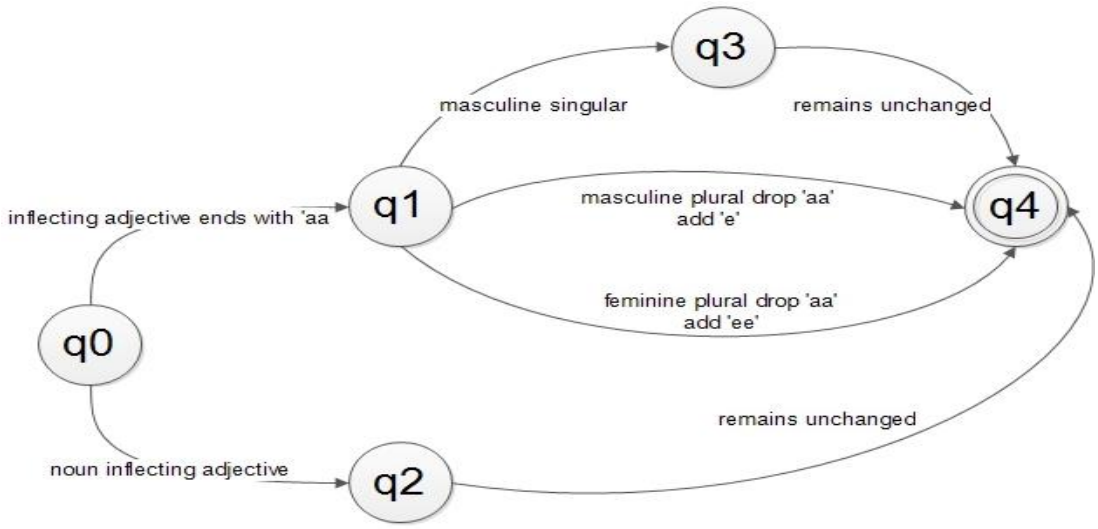


Fig 4.4: Transducer showing singular adjectives to plural adjectives

4.2 Proposed Approach:

Application of Scattered context grammar in Hindi language: As previously describes, scattered context grammar was previously applied to the English linguistics [15]. In this section, we will explore how scattered context grammar can be applied in Hindi language. For example,

"लड़का खाना खाता है"

However if this sentence is modified according to feminine noun, then the sentence can be written as:

"*लड़की खाना खाता है*"

It is grammatically incorrect because the predictor ("khaata") needs to be changed according to the doer ("ladkaa"). Therefore, in Hindi language if *"लड़का"* is changed to feminine noun *"लड़की"*, the sentence changes in this manner:

"लड़की खाना खाती है"

We can see that the verb also changes according to the noun used. This indicates that there is some dependency in the words of the sentence although the words are not adjacent to each other. Here "***लड़का***" and ***"खाता"*** although are not adjacent are changed into "***लड़की***" and "***खाती***" in **Hindi language but not in English language.** But if the same sentence is analysed in English grammar then the sentence would be:

"Boy eats the food"

If "Boy" is replaced by "girl", the rest words of the sentence remain same like:

"***Girl eats the food".***

Here "**eats the food"** remains unchanged, while if we translate the same sentence in Hindi language, it changes. These changes are reflected in Hindi language and can be readily shown using scattered context free language, because dependency in individual words of the sentence can be effectively reflected by this grammar. ***Although it is fairly easy to use context sensitive grammar where a single word comes in between the related words but the case in which more than one word comes in between the related words, many auxiliary rules needs to be designed to handle such dependencies hence scattered context grammar is used here*** **[15].**

Example is shown below.

लड़का कभी-कभी देर से खाना खाता है**।**

Or लड़का समय से, लेकिन कभी-कभी देर से खाना खाता है**।**

Production of scattered context grammar can be shown as:

$$(\textit{लड़का, खाता}) \longrightarrow (\textit{लड़की, खाती})$$

This rule would check whether there is any "ladkaa" in the sentence as the doer and "khaata" as the predicator. If any of the Hindi sentences satisfies the above property, then that sentence changes *in parallel* according to the right hand side of the production as shown in scattered context grammar production. Words in the sentences also changes when it is converted from singular to plural. These changes from singular to plural leads to the change of certain words in the sentence while leaving other words of the sentence unchanged. These can be handled by **scattered context grammar**.

i. Application of Scattered context grammar for Nouns in Hindi sentences:

In case of change of sentence from singular to plural, the corresponding noun in the sentence needs to be changed which leads to the change in the object of the sentence in order to make the sentence grammatical. And this rule for changing depends on the case whether we are using masculine noun or feminine noun as shown above by using transducer in section 4.1. Masculine nouns are those which are related to men, boys and male animals while feminine nouns are those which are related to females, girls and female animals. Masculine nouns which ends with "*aa"* [42] (for example ladkaa i.e. लड़का) are changed by dropping "aa" from the end and adding "e" instead [43].

लड़का हर रोज़ अपने दोस्तों के साथ फुटबॉल खेलता है।

For example "ladkaa" in Hindi is changed to "ladke" in the following example.3

लड़के हर रोज़ अपने दोस्तो के साथ फुटबॉल खेलतें हैं।

Here "***लड़का"*** is changed to "***लड़के***" while converting the sentence from singular to plural and verb changes from "***खेलता है"*** to "***खेलतें हैं"***. There are some irregular nouns which do not change during the conversion of a sentence from singular to plural. Similarly feminine nouns in Hindi which ends with "i" (for example *ladki* i.e. लड़की")

drops "i" from the end and adds "iyaan" to change from singular to plural. For example,

लड़की कम उम्र मे भी बहुत सुंदर नृत्य करती है।

In this sentence "***लड़की***" is changed to "***लड़कियाँ***" in the following manner with the change of "***करती है***" to "***करतीं हैं***".

लड़कियाँ कम उम्र मे भी बहुत सुंदर नृत्य करतीं हैं।

Similarly, there are some feminine nouns which ends with "अ" and when these are converted into plural ," ए" is added after dropping "अ" from the end. For example,

"पुस्तक मेज़ पर रख दी"।

When this sentence is translated to its plural, it becomes

"पुस्तकें मेज़ पर रख दीं"।

There is one more case where the feminine noun ends with "**आ**" and its plural drops nothing but adds "**एँ**" at the end. For example, "**लता**" is changed to **"लताएँ"** in the following sentence

पेड़ की लता हरी-भरी है।

Is changed to

पेड़ की लताएँ हरी-भरी हैं।

These rules can be represented by:

(*Singular noun as subject, singular verb of predicates)* ⟶ *(Plural nouns as subject, plural verb of predicates)*

ii. Application of Scattered context grammar for handling Adjectives in Hindi sentences:

Adjectives are the words which describe nouns [44]. The adjectives used in English do not change themselves according to nouns but the Hindi adjective changes themselves according to nouns. For example, "tall boy", if transformed to plural, then it would be written as "tall boys". Here "tall" remains the same even if the above is transformed to feminine noun from masculine noun, example "tall girl". In Hindi

adjectives changes themselves according to the nouns used. Adjectives can be classified into inflecting adjectives and non-inflecting adjectives. Inflecting adjectives are those adjectives which inflect. These adjectives usually end with "**आ**" or "**aa**". For example,

When masculine noun is changed to plural, adjectives change according to nouns. Adjective drops "aa" or "**आ**" from the end and add "e" or "ए" at the end. The above adjectives are converted in this manner:

As adjectives change according to the change in the noun from "***लड़का***" to "***लड़के***" and changes the verb accordingly (from "***जानता है***" to "***जानते हैं***" in the above example). These dependencies can be better illustrated using **scattered context grammar** and the rules for this can be generated in the same manner as devised above for nouns. If the same adjective is converted into feminine noun, then it would be written as:

Now, if the same is converted into plural, then it would be written as:

As dependency is shown by the arrow in the above sentences, these dependencies can be handled by the scattered context grammar. In case of non-inflecting adjectives, adjective remains unchanged in both the cases of feminine and masculine singular or plural. For example, "*गरीब लड़का*", when this is converted to its plural form, it is written as "*गरीब लड़के*". In case of feminine noun also, adjectives are written as "*गरीब लड़की*". Now this feminine noun is converted as "*गरीब लड़कियाँ*". So "*गरीब*" remains the same in both the cases.

Following are the rules generated for inflecting and non-inflecting adjectives:

a) For Inflecting adjectives

(Singular adjectives, singular noun, singular verb) ⟶ *(Plural adjective, plural noun, plural verb)*

b) For non-inflecting adjectives

(Adjectives, singular noun, singular verb) ⟶ *(Same adjective, plural noun, plural verb)*

iii. Application of Scattered context grammar for Verbs in Hindi sentences:

In Hindi, verb must change itself according to the gender, number, person, tenses and mood in the subject. There are two forms of verbs described in the sentence namely *Infinitive verbs and Participles*. Infinitive verbs are those verbs which end with "na" (ना) [45]. For example, *खाना, पीना, जाना, आना, पढ़ना, करना, सीखना, देना, लाना, लिखना.* Plural of masculine "aa" of "naa" becomes "-e". For example, "लेना" ("lenaa") becomes "लेने" ("lene"). Plural of feminine –i ending either remains –i or becomes –i~ (nasalized). For example, "देखी" ("dekhi") becomes "देखीं" ("dekhi~). Verbs in Hindi sentences changes according to the doer noun which leads to the serial dependency as previously shown in Nouns section changes according to gender, number, tenses, person and mood as shown below.

राम को अभी गुलाब्ज़ामुन खाना है।

Here we can see the dependency between "गुलाब्ज़ामुन" (Masculine noun) and "खाना" in such a way that if such a masculine noun is replaced by feminine noun, the verb would change accordingly.

For example,

राम को अभी कुलफी खानी है।

In above example, the dependent words are adjacent and can be represented using context free grammar.

राम को गुलाब्ज़ामुन खाना खाने के बाद खाना है।

is converted into

राम को कुलफी खाना खाने के बाद खानी है।

Context free grammar cannot handle the case where the dependent words are not adjacent as shown in above example. It can be handled by context sensitive complex set of rules or by using scattered context grammar. In above example dependency is because of gender of the noun ("गुलाब्ज़ामुन" as masculine and "कुलफी" as feminine) which modifies the verb ('खाना'), there can be dependency because of person, tenses, mood etc. So the rule of scattered context grammar in case of dependency due to gender is as shown below:

(Masculine noun in subject (singular), verb (ending with "aa") in object)→ (Feminine noun in subject (singular), verb (ending with "i") in object)

Similarly the rest of the rules for the verbs can be generated where the verb is dependent on number, person, tenses or mood of the subject.

Participles are of further two types:

a. Present (or imperfective participle)

   It adds "ta" at the end of the verb. For example, *बोल+ता=बोलता.* These participles are used in the formation of present and past indicatives.

b. Past (or perfective participle)

   It adds "aa" at the end of the verb. For example, *बोल+आ=बोला.* These are used in the formation of perfective tenses [46].

Dependency in case of imperfective participle

राम वन-वन भटक कर लकड़ियाँ काटता है।

In this sentence, if masculine noun (ram) is changed to feminine noun (sita), then the verb will change accordingly in the described manner:

सीता वन-वन भटक कर लकड़ियाँ काटती है।

Scattered grammar production for imperfective participle is as shown below:

(Masculine noun in subject, verb ending with "taa") →(Feminine noun in subject, verb ending with "i")

Dependency in case of perfective participle

राम ने कल एक कार्यक्रम देखा।

In the above sentence, if "एक" is changed to "सभी", i.e. there is dependency because of number, and then the verb also changes from "देखा" to "देखे" as shown below:

राम ने कल सभी कार्यक्रम देखे।

Therefore the scattered grammar production for such types of perfective participle is as shown below:

(Singular noun (in case of number, verb ending with "aa")) →(Plural noun, verb ending with "ee").

There are around 6 irregular verbs as described in the Hindi grammar rules. When a verb stem ends with "a" vowel, the masculine singular form requires that a "y" should be inserted before the suffix "a" and optionally in the masculine plural and feminine nouns [42]. Scattered context grammar can handle such dependencies while with context sensitive grammar it appears difficult to do so.

Cross dependencies in Hindi sentences

Cross dependency can also exist in Hindi language as shown in example below.

There is another way of dependency also which leads to cross serial dependency in the sentences and this is shown below. For example,

पुस्तक मेज़ पर बिखरी पड़ी थी।

The above sentence seems like the adjoined sentence from these two sentences "पुस्तक बिखरी थी" and "मेज़ पर पड़ी थी". All which is needed to do is to convert these sentences in their plural form "पुस्तकें बिखरीं थीं"and "मेज़ पर पड़ीं थीं" then again adjoin these sentences to get the result [47]. This cross dependency can easily be handled by scattered context grammar while this may lead to generation of many rules when context sensitive grammar is used. The plural form of the sentence is as shown below:

"पुस्तकें मेज़ पर बिखरीं पड़ीं थीं"।

Another example illustrating cross-serial dependency in Hindi language s as shown below:

राष्ट्रपति ने कारगिल लड़ाई के शहीद को पुष्प चढा कर श्रधांजलि दी।

Again this cross-serial dependency can be handled by dividing the above sentence into two sentences, "राष्ट्रपति ने पुष्प चढ़ाए" and "शहीद को श्रधांजलि दी". These sentences in their plural forms are "राष्ट्रपति ने पुष्प चढ़ायें" and "शहीदों को श्रधांजलि दीं". Individually both the sentence can be handled by scattered context grammar as described above and then again these sentences are adjoined to give this sentence.

राष्ट्रपति ने कारगिल लड़ाई के शहीदों को पुष्प चढ़ा कर श्रधांजलि दीं

Clauses with "neither-nor" or "न-न"

Sentences in English having **neither-nor** [48] can be complemented by using **both-or** in place of neither-nor and this negation can make use of transformational context grammar as explained in [49].

"Neither ram nor sita went to college this Friday".

The negation of this sentence converts the *neither-nor* to *both-and* in this way.

"Both ram and sita went to college this Friday".

So this production in English can be written as

$$(Neither, nor) \longrightarrow (Both, or)$$

Similarly in Hindi language,

"मैं न हिन्दी न इंग्लीश सीखूँगी"

Can be converted (negation of above sentence) as

"मैं या हिन्दी या इंग्लीश सीखूँगी"।

So here scattered context production for such sentences is:

$$(न, न) \longrightarrow (या, या)$$

This is how the scattered context grammar can be used in handling such dependencies between words in Hindi language. But how we can analyze the dependency and the dependent words is described above in methodology section. It can be done using Hindi Dependency Parser and the results of using Hindi dependency parser are shown in the next section.

Chapter 5
Implementation and Simulation Results

In this chapter implementation of the proposed solution is described. Hindi dependency Parser is used to show the dependency between the words of the sentence. Firstly, the user can enter any Hindi sentence to generate the dependency tree of that Hindi sentence. The Hindi dependency parser generates the Hindi dependency dataset corresponding to the sentence input by the user. This Hindi dependency dataset is used in the designing of dependency tree with the help of NetworkX. Therefore in the final output, we are presented with dependencies between the words of the sentences and dependent words are recognized easily from the graph. Now from those dependent sets, we can generate their plural forms and apply scattered context rules at the end over those sets.

Anaconda is used for implementing python and networkx over it.

5.1 Overview of Python

Python is an object oriented programming language. It is also used as a glue or scripting language which is used to connect existing components to one another.

Following are the features of Python language [50]:

- It is simple and easy to learn.
- It is an interpreted and high-level language with dynamic semantics.
- It is mainly used in the area where rapid application development is required.
- Its syntax emphasizes readability.
- It also supports packages and modules which emphasize modularity of the program and can also help in code reuse.
- Its standard libraries and interpreter are easily available in binary form and it can be freely distributed.

5.2 Overview of NetworkX

NetworkX is software which is built under Python language [52]. It is mainly used for manipulation, creation and study of the dynamics, structure and various functions of complex networks.

Following are the features of NetworkX [51]:

a) It consists of network analysis measures and structure.
b) It is a generator for random graphs, classic graphs and synthetic networks.
c) It is an open source BSD license.
d) Its nodes can be text, images or XML records.
e) It uses python language data structures for digraphs, graphs, and multi-graphs.
f) It consists of many standard graph algorithms.

5.3 Proposed system

Following are the steps used to define dependencies between words:

a) Rules are defined for nouns, adjectives and verbs and works for conversion of singular to plural form.
b) Hindi dependency parser describes the dependencies among the words of the sentence. This dependency parser is accessed using Python language.
c) NetworkX is used to generate the dependency tree whose working is also defined in the Python language.
d) Dependent sets of the words are generated from dependency graph and these are converted from singular to their plural form. After this scattered context grammar is applied over these dependent sets.

5.3.1 Application of Hindi Dependency parser

Hindi dependency parser takes Hindi sentence as input and generates corresponding output file comprises of following columns.

a) Word id: Hindi dependency parser allocates an id to each word of the sentence in order to differentiate it from other existing words.
b) Word: These are the word forms or punctuation symbol.
c) Lemma: Lemma is the root word of the previous column "word".
d) POS Tag: POS tagging means assigning parts of speech to each and every word of the sentence like Nouns, Adjectives, verbs etc.
e) Parent id: This is considered as the Head of the current token, which is either a value of word ID or zero ('0'). Depending on the original treebank annotation, there may be multiple tokens with an ID of zero.
f) Dependency Labels (DL): Dependency Label is the dependency relation to the HEAD. The set of dependency relations depends on the particular language.

Depending on the original treebank annotation, it should be noted that the dependency relation can be meaningful or simply 'ROOT'.

The command used to generate Hindi dependency dataset is:

make hindi.output

This command will generate an output file named "hindi" having above set of columns.

5.3.2 Application of NetworkX

NetworkX is used here to generate the dependent graph. For using NetworkX (for graphical purpose over here), we have to first import networkx by using the following command:

import networkx as nx

Anaconda is used for python platform and use of networkx is done under python (Appendix A). Networkx is unable to label its nodes in Hindi language so indexing is used and the index file is shown separately. The result shown here gives the dependent sets of words in the Hindi sentence both graphically and theoretically. Anaconda is having an advantage that using anaconda avoids installing python and matplotlib explicitly. The overall working is coded in such a way that the user can enter any Hindi sentence by a single command of running the python file in the Terminal. Here the name of the file is Graph.py and the command which is to be used for running the above work is:

python Graph.py

Running this command would ask user to enter his text and the user is required to enter the text. After entering the text, the task is all done from the user side, now the code will generate output corresponding to the input performed by the user. Result comprises of an output file (which comprises of the dependency dataset corresponding to input file), indexing file (named as dependencies.input.txt over here) and the dependency graph generated using networkx.

The example which is used here is:

राम ने श्याम के साथ सीता को भी हमेशा साथ रहने की सलाह दी

This is the sentence which is used in all the examples for explaining the work. Hindi dependency set, indexing and the dependency graph is all shown using the above

Hindi sentence. After installing the Anaconda on Ubuntu platform, the steps performed are written with their results below:

1. Running python file (Graph.py): The command which is to be run over the terminal is as shown by figure 5.1 and figure 5.2.

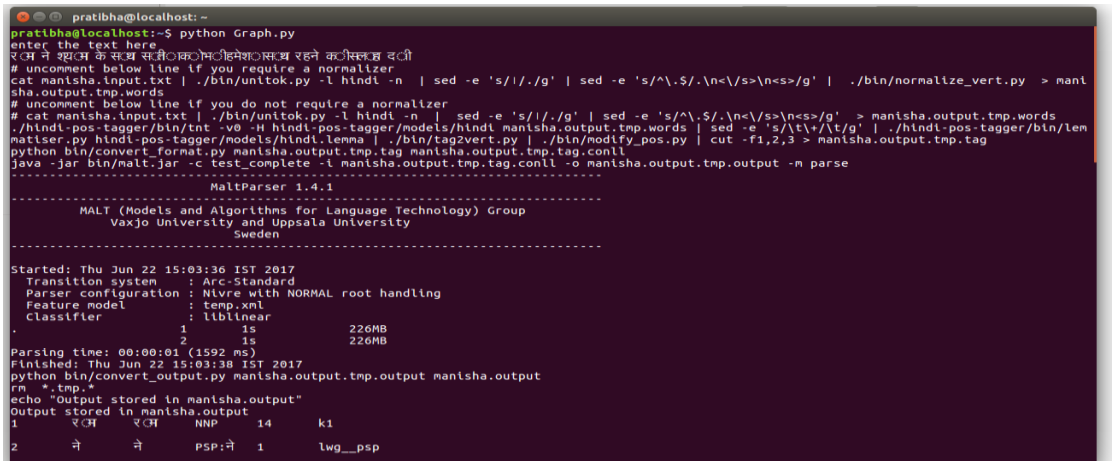


Fig 5.1: Graph.py file

As it can be seen in Figure 5.3, the user is presented with the location of both the indexed file and also the dependency graph with their names. This all is possible because of Hindi dependency parser which generates an output file corresponding to each text, the user enters. When the user enters any text, it stores that text in its input file as shown in figure 5.4. Here the name of the input file given is *manisha.input.txt* which is responsible for generating the output having name *manisha.output.txt* which comprises of Hindi dependency dataset corresponding to that Hindi input provided by the user. This output file is having all those columns of Hindi dependency datasets like parent id, word, word id, dependency labels, Lemma etc. And the output file corresponding to the given input is as shown in figure 5.5.

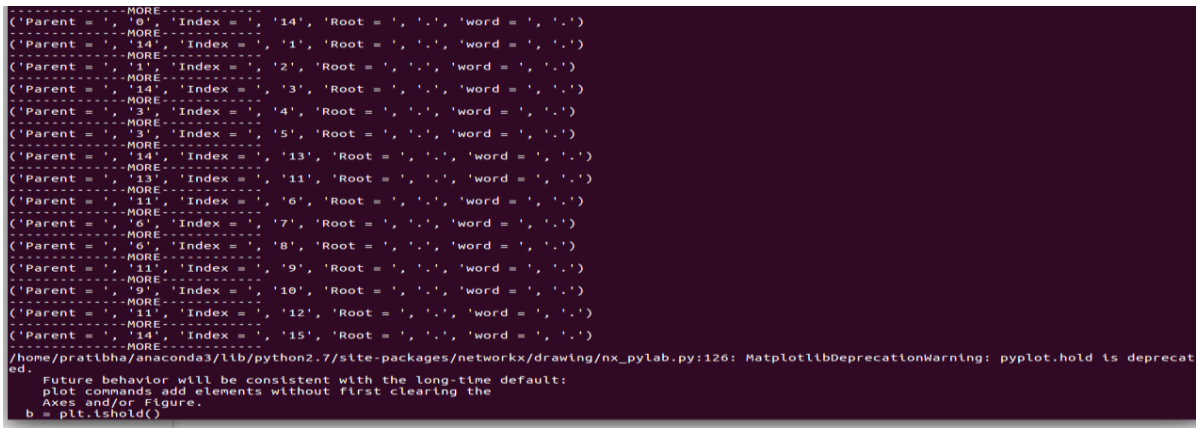


Fig 5.2: python file (Continued)

From the working itself, user would be aware of index file and the dependency graph with the names of their files as shown in figure 5.3.

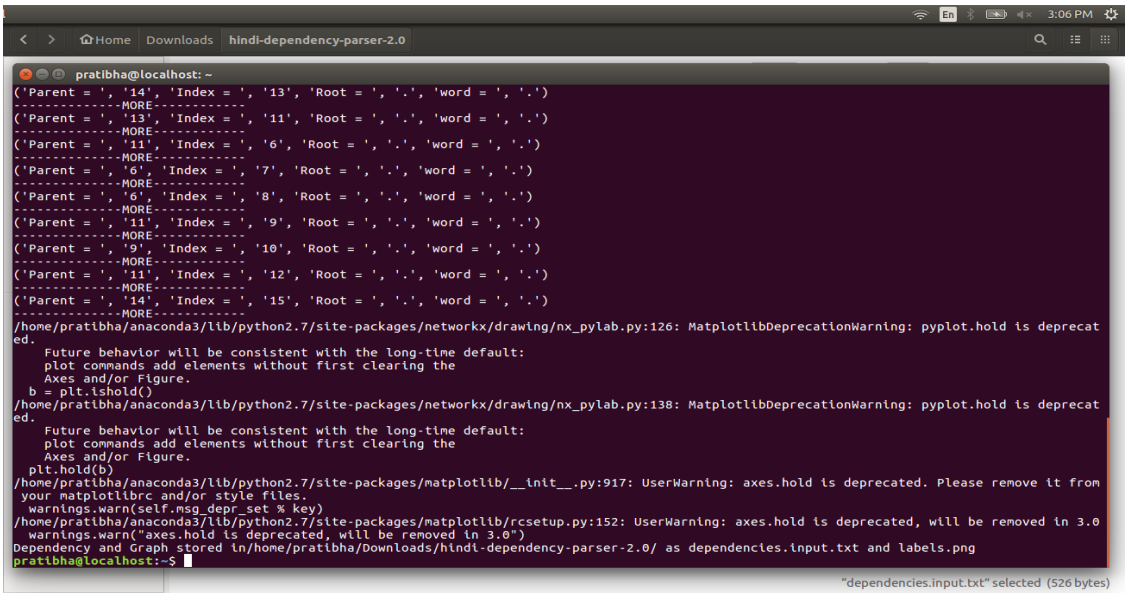


Fig 5.3: Location of index file and graph

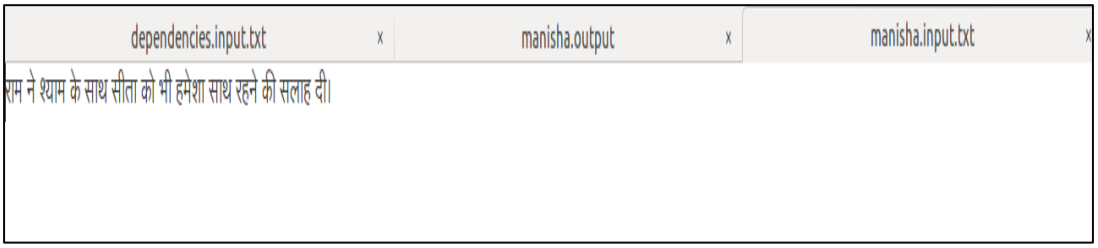


Fig 5.4: Input file

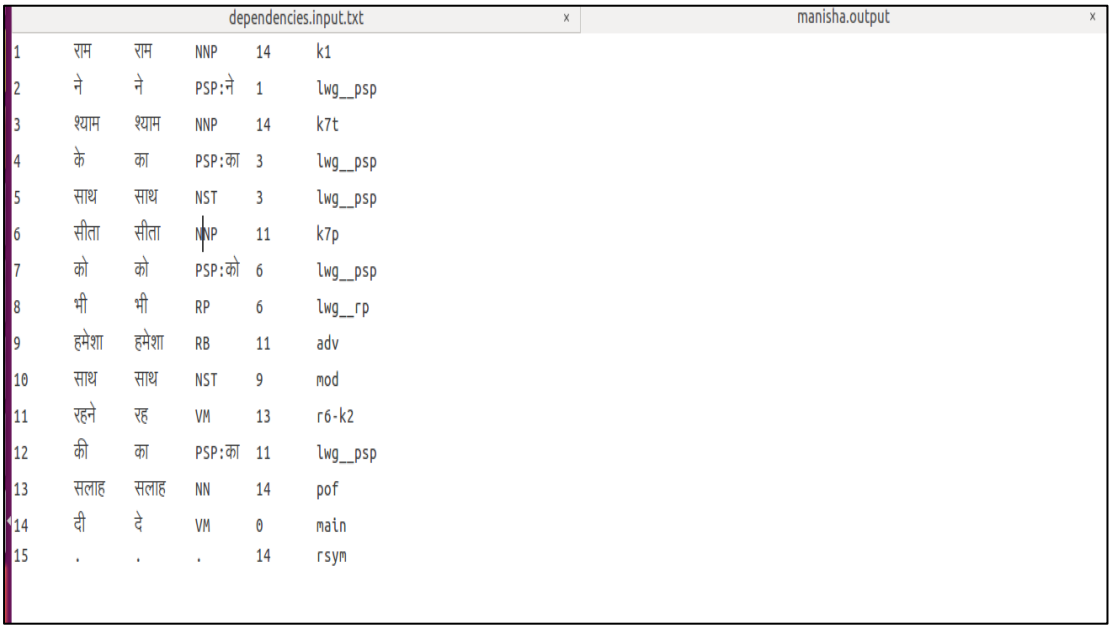


Fig 5.5: Output file generated

Now after this, the code will generate the index file which gives id to each word of the sentence. In this file not only the index of the words is mentioned but also the dependencies among the words are defined which leads to the generation of dependency tree. This index file is shown in figure 5.6 and it is named as *dependencies.input.txt* in our work. After this networkx comes into functioning which generates the dependency tree for the input given by the user. This is as shown in figure 5.7. But figure 5.7 is unable to explain the dependency graph without index file so dependency graph with index file is also shown below in figure 5.8. Here user can get the idea of each of the node of dependency graph.

Fig 5.6: Index file

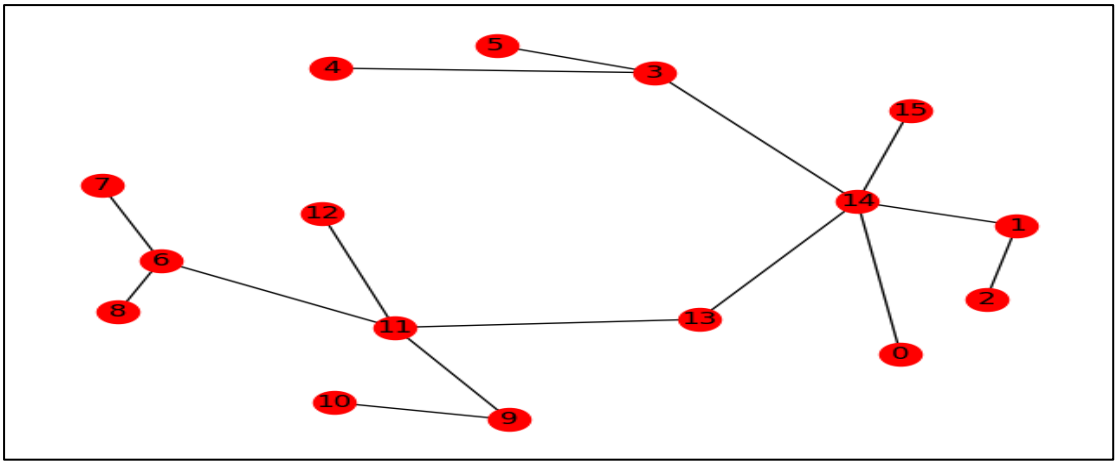


Fig 5.7: Dependency graph based on index file

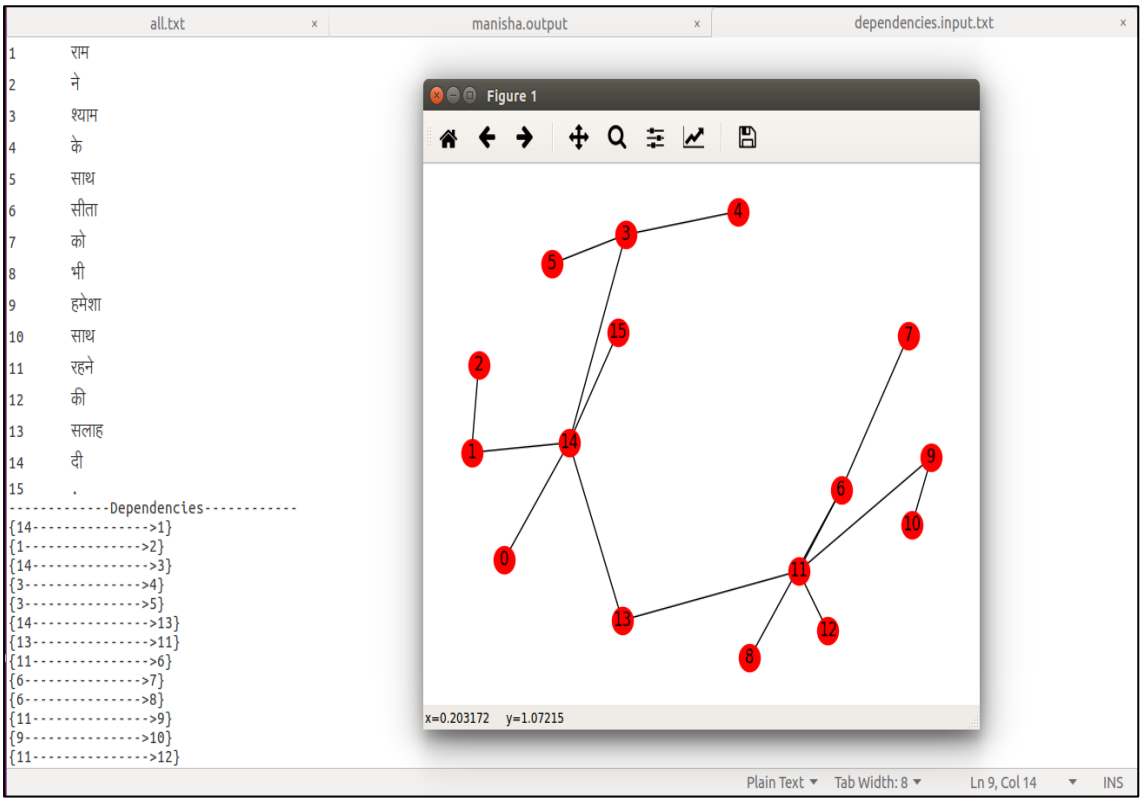


Fig 5.8: Index file explaining Dependency graph

Another example illustrating the generation of dependencies is shown with the help of another Hindi example.

वह बहुत पुराना महल बहुत विशाल एवं बहुत डरावना है

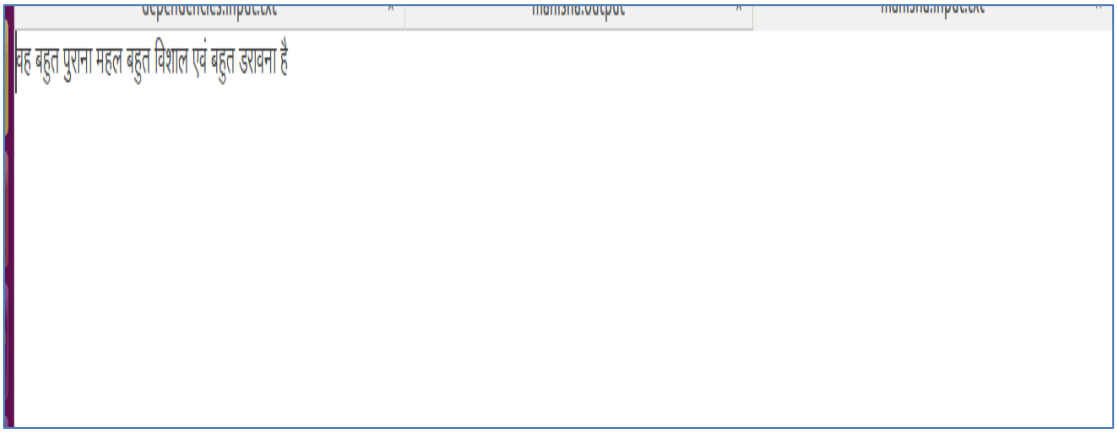


Fig 5.9: Input file stored by Hindi dependency parser

When this sentence is entered by the user, this sentence gets entered in the input file of Hindi dependency parser as shown in figure 5.9. The task of Hindi dependency parser is to generate the Hindi dependency tagset which is as shown in figure 5.10.

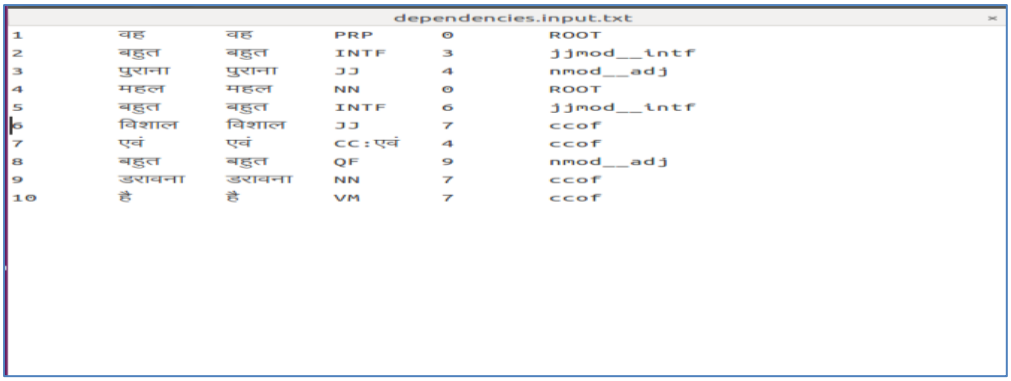


Fig 5.10: Output file generated by Hindi Dependency Parser

On the basis of above file generated, dependency graph needs to be generated which applies scattered context grammar over the dependent rules and the dependency graph is shown in figure 5.11.

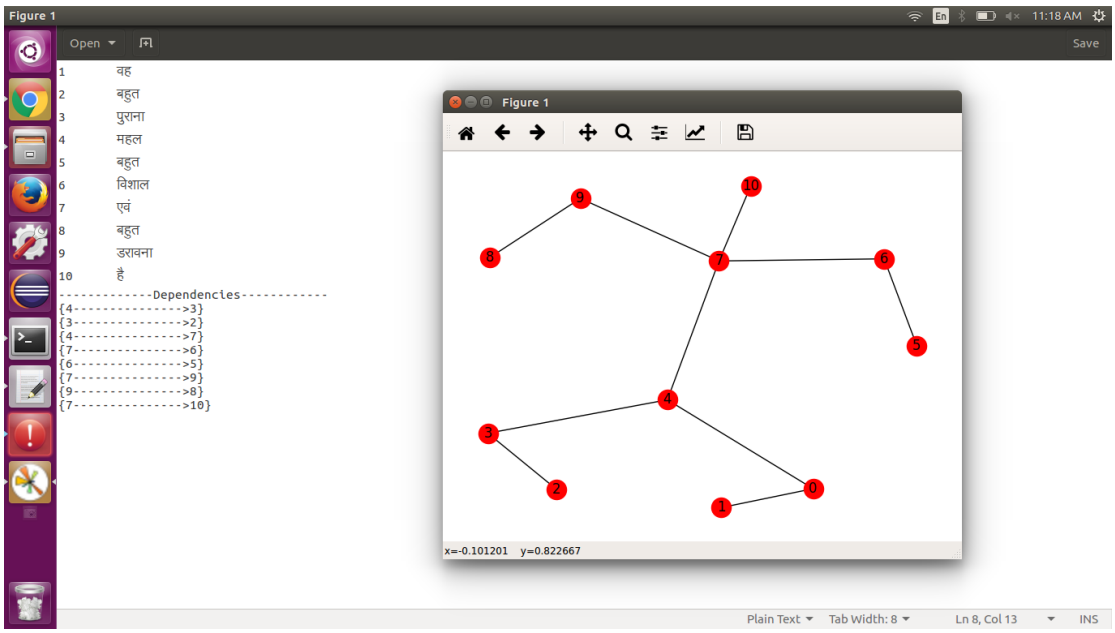


Figure 5.11: Output file having dependency graph with index file

The dependent sets obtained from the above implementation can now be helpful in generating scattered context rules. These scattered context production rules can be easily implemented over the dependencies between the words of Hindi dependencies. This work would also save us from generating complex context sensitive rules for the same problem.

Chapter 6
Conclusion and Future Scope

6.1 Conclusion

In this study, we have covered the use of scattered context grammar in representing Hindi language dependency. Although scattered context grammar have already been used in English linguistics as discussed earlier but the use of scattered context grammar in Hindi linguistics was a bit difficult task because of its rich morphology. We have tried to understand the dependencies among the Hindi words in the sentence with the help of dependency tree which is constructed using NetworkX.

With the help of scattered context grammar in Hindi dependencies, complex rules of context sensitive grammar can be avoided. Dependencies among the words can be represented using scattered context grammar which is having its generative power between context free grammar and context sensitive grammar. As this scattered context grammar forms the production rules for the singular sentences to plural sentences, user becomes aware of the grammatical error which may appear while translating sentence from singular to plural. Scattered context grammar used here is capable of handling both inter chunk and intra chunks dependencies quite effectively and the scattered context rules which are developed for nouns, adjectives and verbs are generalised rules and are not restricted to some specific Hindi sentences.

6.2 Future Scope

We believe there is a lot that is yet to be explored in this area. Following are the points that can be considered for further research on this topic:

i. We have ignored those sentences with cross serial dependencies which lead to the generation of more than one dependency tree. Such types off cross- serial dependencies can be studied to an extent higher.

ii. We have used the scattered context grammar in the case where the Hindi sentence is changed from singular to plural. This conversion generates some dependencies and hence arise the need of production rules. But these dependencies can also be generated when mode of the sentence is changed or the sentence is converted from active voice to passive voice.

Appendix A

Networkx as described in chapter 5 is used under Python platform to generate the dependency graph. Steps which are to be followed to generate the dependency graph are coded in python language as shown in figure A1.

```
import networkx as nx
import matplotlib.pyplot as plt
import pylab as p
import os
from nltk.corpus import indian


def traverseArray(arrWords,parentId,G):
        print("-------------MORE------------");
        for dictWord in arrWords:
                if(int(dictWord['parentid']) == parentId):
                        print("Parent = ",dictWord['parentid'],"Index = ",dictWord['index'],"Root = ",dictLine['rootword'],"word = ",dictLine
['word'] );
                        if(int(dictWord['parentid'])!= 0):
                                fileDependencies.write("{"+dictWord['parentid']+"--------------->"+dictWord['index']+"}\n");
                        G.add_edge(dictWord['parentid'], dictWord['index']);
                        G = traverseArray(arrWords,int(dictWord['index']),G);
        return G;



build_dir="/home/pratibha/Downloads/hindi-dependency-parser-2.0/"
cwd=os.getcwd()
parent=[];
arrLines = [];
try:
        os.chdir(build_dir)
        f=open("manisha.input.txt","w+")
        f1=open("all.txt","a+")
        string=raw_input("enter the text here\n")
        f.write(string)
        f1.write(string + '\n')
        f.close()
        f1.close()
        os.system("make manisha.output")
        os.chdir(build_dir)

        fileDependencies=open("dependencies.input.txt","w+")
```

Python Tab Width: 8 Ln 37, Col 1 INS

Fig A1: Python code generating dependency graph

Files, which are generated is the output of the python code shown in figure A2.

```
try:
        os.chdir(build_dir)
        f=open("manisha.input.txt","w+")
        f1=open("all.txt","a+")
        string=raw_input("enter the text here\n")
        f.write(string)
        f1.write(string + '\n')
        f.close()
        f1.close()
        os.system("make manisha.output")
        os.chdir(build_dir)


        fileDependencies=open("dependencies.input.txt","w+")
        with open("manisha.output") as f:
                for line in f:
                        print(line);
                        if(line != "\n"):
                                dictLine={}
                                dictLine['index'] = line.split('\t')[0]; # index of word
                                dictLine['word'] = line.split('\t')[1]; # index of word
                                dictLine['rootword'] = line.split('\t')[2]; # index of word
                                dictLine['type'] = line.split('\t')[3]; # index of word
                                dictLine['parentid'] = line.split('\t')[4]; # index of word
                                arrLines.append(dictLine);#Add dict in Array
                                fileDependencies.write(dictLine['index'] +'\t'+ dictLine['word']+ '\n')#write in file
                        else:
                                print("Manpreet");
                f.close()
        G=nx.Graph()
        # Add nodes and edges
        fileDependencies.write("-------------Dependencies------------\n");
        G = traverseArray(arrLines,0,G);
        fileDependencies.close();
        nx.draw(G, with_labels = True)
        plt.savefig('labels.png')
        plt.show();
        print("Dependency and Graph stored in"+ build_dir +" as dependencies.input.txt and labels.png")
finally:
        os.chdir(cwd)
```

Python Tab Width: 8 Ln 65, Col 1 INS

Fig A2: Python code generating dependency files

References

[1]. S. Greibach and J.Hopcroft, *"*Scattered context grammars*",* "*Journal of Computer and System Sciences*" 3.3, 1969, pp. 233-247.

[2]. "Scattered context grammars research" [Online] Available: https://labraj.feri.um.si/en/Scattered_Context_Grammars_research [Accessed on 08 May 2017]

[3]. B.R Ambati, "Hindi dependency parsing and treebank validation", *Master's, Hyderabad, India, 2011.*

[4]."Malt Parser" [Online] Available: *"*http://www.maltparser.org/intro.html" [Accessed on 30 May, 2017].

[5]."MST Parser" [Online] Available: "https://github.com/kzn/mstparser/blob/master/ALT_README" [Accessed on 31 May,2017]

[6]. K. Puneeth, "Dependency Parsing and Empty Category Detection in Hindi Language*", Diss. International Institute of Information Technology*, Hyderabad, 2016.

[7]. "Hindi Dependency Parser" [Online] Available: "https://bitbucket.org/sivareddyg/hindi-dependency-parser/src" [Accessed on 18 June, 2017].

[8]. K.Singla, A. Tammewar, N. Jain and S. Jain, "*Two-stage approach for Hindi dependency parsing using maltparser"* Training, *12041*(268,093), 2012, pp.22-27.

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

[10]. B. R. Ambati, T. Deoskar and M. Steedman, *"Using CCG categories to improve Hindi dependency parsing", ACL (2)*, 2013.

[11]. J.Bresnan, R.M. Kaplan, P.Stanley, and A.Zaenen. "Cross-serial dependencies in Dutch" *In The formal complexity of natural language*, Springer, Netherlands, 1982, pp. 286-319.

[12]. H. Fernau and A. Meduna, *"*A simultaneous reduction of several measures of descriptional complexity in scattered context grammars*", Information Processing Letters* 86.5, 2003, pp. 235-240.

[13]. H. Fernau and A.Meduna, *"*On the degree of scattered context-sensitivity*", Theoretical Computer Science*, 2003, pp. 2121-2124.

[14]. S.Husain, *"*Dependency parsers for Indian languages*", Proceedings of ICON09 NLP Tools Contest: Indian Language Dependency Parsing, 2009*.

[15]. A. Meduna and Z. Peter *Regulated Grammars and Automata*, Springer, 2014.

[16]. M.V Yeleti and K. Deepak. *"Constraint based Hindi dependency parsing", Proceedings of ICON09 NLP Tools Contest: Indian Language Dependency Parsing, India*, 2009.

[17]. 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 (TALLIP)* 16.3,2013, pp.17.

[18]. P. Xu, J. Ringgard, "Using a dependency parser to improve SMT for subject-object-verb languages", *Proceedings of human language technologies: The 2009 annual conference of the North American chapter of the association for computational linguistics*, Association for Computational Linguistics, 2009.

[19]. R. McDonald, P. Fernando, R. Kiril, and J. Hajič. "Non-projective dependency parsing using spanning tree algorithms", In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, 2005, pp. 523-530.

[20]. D. Zeman and P. Resnik, "Cross-Language Parser Adaptation between Related Languages*",* In *IJCNLP*, 2008, pp. 35-42.

[21]. B.R.Ambati, B.R., P. Gade, G.S.K Chaitanya, S. Husain and LTRC, *Effect of Minimal Semantics on Dependency Parsing*. In *RANLP*, 2009, pp. 1-5.

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

[23]. S.Husain, P.Gadde, B.Ambati, D.M. Sharma and R.Sangal, December. *A modular cascaded approach to complete parsing*, In *Asian Language Processing, 2009. IALP'09. International Conference*, IEEE, 2009, pp. 141-146.

[24]. S. Kolachina, P. Kolachina, M. Aggarwal and S.Husain, "Experiments with maltparser for parsing Indian languages*", Proc of ICON-2010 tools contest on Indian language dependency parsing, Kharagpur, India, 2010*.

[25]. 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, Association for Computational Linguistics*, 2012, pp. 49-56.

[26]. J. Hockenmaier and M. Steedman, "CCGbank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank", *Computational Linguistics*, vol. *33*(3), 2007, pp.355-396.

[27]. L. Rychnovsky*, "Parsing of context-sensitive languages",* 2007.

[28]. P.Gadde, K. Jindal, S.Husain, D.M.Sharma, & R.Sangal, "Improving data driven dependency parsing using clausal information*",* In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, Association for Computational Linguistics, 2010, pp. 657-660.

[29]. A. Bharati, S. Husain, D.M Sharma & 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.*

[30]. 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*, Association for Computational Linguistics, 2009, pp. 186-189.

[31]. R. Tsarfaty, D. Seddah, Y. Goldberg, S. Kübler., M. Candito, J. Foster & 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*, Association for Computational Linguistics, 2010, pp. 186-189.

[32]. K.Singla, A.Tammewar, N. Jain & S. Jain, *"*Two-stage approach for Hindi dependency parsing using maltparser*"*, *Training*, *12041*, 2010, pp. 22-27.

[33]. J. Nivre, *"*Parsing Indian languages with maltparser*"*, *Proceedings of the ICON09 NLP Tools Contest: Indian Language Dependency Parsing*, Hyderabad, India, 2009, pp. 12-18.

[34]. A. Bharati, S.Husain, M. Vijay, K. Deepak, D.M. Sharma & R. Sangal, *"*Constraint Based Hybrid Approach to Parsing Indian Languages*", PACLIC*, Hong Kong, 2009, pp. 614-621.

[35]. N. Kalra and A. Kumar. "State grammar and deep pushdown automata for biological sequences of nucleic acids", *Current Bioinformatics, India*, 11.4, 2016, pp. 470-479.

[36]. N. Kalra and A. Kumar. "Fuzzy state grammar and fuzzy deep pushdown automaton", *Journal of Intelligent & Fuzzy Systems, India*, 31.1, 2016, pp. 249-258.

[37]. N. Kumar "Nouns", Available at "http://blogs.transparent.com/hindi/how-to-make-plural-from-singular-noun-in-hindi/" [Online] Accessed on 30 June, 2017.

[38].Verbs, Available at "http://www.geocities.ws/lordvaruna/" [Online] Accessed on 30 June, 2017.

[39].Verbs, Available at "https://en.wikibooks.org/wiki/Hindi/Verbs" [Online] Accessed on 30 June, 2017.

[40]. N. Kumar, Available at http://blogs.transparent.com/hindi/top-10-verbs-in-hindi/ [Online] Accessed on 30 June, 2017.

[41]. Adjectives, Available at "http://www.dataspec.info/english/Adjective/index.htm" [Online] Accessed on 30 June, 2017.

[42]. Masculine Nouns, Available at "https://unacademy.com/lesson/masculine-and-feminine-part-1-in-hindi/756VU2FK" [Online] Accessed on 30 June, 2017.

[43]. A. Sonak "Mind ur Hindi", 2017," Available at: "http://www.mindurhindi.com/hindi-grammar-rule/adjectives-in-hindi/" [Online] Accessed on: 08 May, 2017.

[44]. "Hindi Verb Conjugation" Available at: "http://www.geocities.ws/lordvaruna/" [Online] Accessed on: 08 May, 2017

[45]. Participles, Available at "http://hindilanguage.info/hindi-grammar/verbals/participles/" [Online] Accessed on 30 June, 2017.

[46]. 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, Hyderabad, India*, 2010, pp.1-8.

[47]. A. K. Joshi and S. Yves. "Tree-adjoining grammars", *Handbook of formal languages*. Springer, Berlin Heidelberg, 1997, pp. 69-123.

[48]. A. Sonak, Available at "http://www.mindurhindi.com/learn-hindi-lessons/either-neither/" [Online] Accessed on 30 June, 2017.

[49]. P.Kosaraju, S.Husain, B.R.Ambati, D.M.Sharma, & R.Sangal, *"*Intra-chunk dependency annotation: expanding Hindi inter-chunk annotated Treebank*",* In Proceedings of the Sixth Linguistic Annotation Workshop, *and Association for Computational Linguistics*, Stroudsburg, PA, USA, *2012*, pp. 49-56.

[50]. Python, Available at "https://www.python.org/doc/essays/blurb/" [Online] Accessed on 30 June, 2017.

[51]. Networkx, Available at "https://networkx.github.io/" [Online] Accessed on 30 June, 2017.

[52]. Github, Networkx Tool, Available at "https://networkx.github.io/" [Online] Accessed on 24 June, 2017.

Research Publication

M. Singla and A. Kumar, "Application of Scattered context grammar for resolving dependency in Hindi Language", *International Conference on Information Communication, Instrumentation and Control (ICICIC),* IEEE – 2017.

[Accepted]

Video Link

The video can be seen on the link: https://youtu.be/Lj9E6bgLCzk

Thesis

ORIGINALITY REPORT

%13	%11	%5	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Regulated Grammars and Automata, 2014. Publication	%2
2	labraj.uni-mb.si Internet Source	%1
3	ltrc.iiit.ac.in Internet Source	%1
4	web2py.iiit.ac.in Internet Source	%1
5	www.scss.tcd.ie Internet Source	%1
6	www.umiacs.umd.edu Internet Source	%1
7	oldwww.iiit.ac.in Internet Source	%1
8	gitlab.lif.univ-mrs.fr Internet Source	%1
9	researchweb.iiit.ac.in Internet Source	%1

10	147.229.9.23 Internet Source	<%1
11	www.citeulike.org Internet Source	<%1
12	www.aclweb.org Internet Source	<%1
13	bitbucket.org Internet Source	<%1
14	maltparser.org Internet Source	<%1
15	www.geocities.com Internet Source	<%1
16	www.semanticscholar.org Internet Source	<%1
17	aclweb.org Internet Source	<%1
18	quranjooy.itrc.ac.ir Internet Source	<%1
19	rian.ie Internet Source	<%1
20	library.naist.jp Internet Source	<%1
21	140.116.245.248 Internet Source	<%1

22	ksync.enhydra.org Internet Source	<%1
23	Mondak, Jeffery J.. "Cultural heterogeneity in capitalist society: In defense of repetition on the billboard hot 100", Popular Music & Society, 1989. Publication	<%1
24	www.dmtcs.org Internet Source	<%1
25	dspace.thapar.edu:8080 Internet Source	<%1
26	Cavanagh, . "Fixed-Point Addition", Computer Arithmetic and Verilog HDL Fundamentals, 2009. Publication	<%1
27	www.seas.gwu.edu Internet Source	<%1
28	ceur-ws.org Internet Source	<%1
29	Meduna. "Context Conditions Placed on the Neighborhood of Rewritten Symbols", Grammars with Context Conditions and Their Applications, 05/27/2005 Publication	<%1
	Lecture Notes in Computer Science, 2003.	

EXCLUDE QUOTES	ON	EXCLUDE MATCHES	< 4 WORDS
EXCLUDE BIBLIOGRAPHY	ON		