Development of Font Independent Spell Checker for Hindi

Rajneet Kaur & Dr. Dharam Veer Sharma

Department of Computer Science, Punjabi university, Patiala, India
Email: kaur.rajneet13@gmail.com, dveer72@hotmail.com

ABSTRACT

This Research is an attempt to highlight the need and awareness of Font Independent Spell Checker for Hindi. Before this research many Spell Checkers for Hindi are available but all are font dependent i.e. they work on single font of Hindi script. As Hindi Script have many fonts. Each Font needs a separate Spell Checker, Because of character mapping differences. There was no standardized mapping technique available. To overcome this problem, a need to develop a Spell Checker for Hindi Language, which is independent of fonts, arises. This task is accomplished by providing standardization while mapping. This standardization of mapping is provided by Unicode Encoding Technique.

Keywords: *NLP, Spell Checker, Font Independent Spell Checker for Hindi, Font Conversion, Font Classes & character Mapping, Key Mapping.*

I. Introduction:

Natural Languages Processing (NLP) is an interaction between computers and human languages. These are called natural because we do not invent them consciously, these are originated regionally. The Constitution of India designates the official language of the Government of India as Hindi (written in Devanagari script), and English. Each language has many fonts in which that language can be written. Similarly Hindi language also have many fonts i.e. Amar, AmaHindi, BharatVani Wide Font, Walkman-Chanakya-905, DevLys 010, Aakriti, Rekha. NLP has many applications like automatic summarization, coreference resolution, Machine translation, morphological segmentation, optical character recognition (OCR), Speech Recognition, word segmentation, speech processing etc.

Spell Checker: Spell checker in any language is software which identifies errors as well as corrects them by providing all possible suggestions. Ranking of suggestions is done by algorithms. They can either be combined with other applications like android phones, email or word processers or may be distributed individually. Spell checker plays an important for catching typos, but they do not help much when misspelled word creates a valid word; for example, you type too instead of to**.**

Practically, by using Spell Checker response time and error correction rate gets improved.

Two main functions of Spell Checker

Error Detection: The very first function of every spell checker in any language in to detect errors. Errors may be arising intentionally or unintentionally. Errors may occur due any of following reasons.

- ➢ Due to poor writing skills of writer.
- ➢ Lack of poor command over the working language.
- ➢ To complete the task in short period of time.

Error Correction: After detecting the error, second main function of the spell checker is to correct founded errors. Errors are corrected by suggesting few similar words. Error Correction involve following steps:

- ➢ Generate suggestions.
- ➢ Ranking of suggestions.

Errors in Spell Check

➢ **Real Word Errors**: These are acceptable valid words in the dictionary. These are acceptable in the lexicon but unsuitable for sentence formation. Real word errors detection is resolved by advanced statistical and Natural Language Processing techniques. For example:

सीता और राम भाई बहन है
सीता ओर राम भाई बहन है

In above both sentences, pronunciation of both words और and ओर is same. But,

Meaning of और is ---------Or
Meaning of ओर is ------- Direction

➢ **Non Real Word Errors:** These are non-acceptable invalid words in the dictionary. Syntax of these words is incorrect and not found in the dictionary. They occur due to lack of knowledge of spelling of the correct word or Human typing (wrong key press). Typographic errors are non-real world errors.

Typographic Errors: These errors arise unintentionally. Because writer knows correct spelling of the word but while typing he typed inaccurate word. Typographic errors are categorised into following categories.

i. **Insertion Error** Errors of this type arises when an extra letter in inserted unintentionally. In the following example an extra ा is typed. कामा for काम

ii. **Transposition Error**: Errors of this type arises when two adjacent letters are swapped with each other. In the following example ल and म get interchanged with each other. Transposition errors either are Real world or Non Real World. कलम for कमल

iii. **Deletion Error:** It occurs when at least one letter is deleted in a word. Letter may delete from front, end, middle or from any position. In the following example ल is deleted from the word कलम

iv. **Substitution Error:** Errors of this type arises when one or more letters are substituted by some another letter [1]. In the above following examples (ं) for (ँ), ये for ए are the various substitution pairs.

हिंदी for हिन्दी , लिये for लिए

The main reasons for substitution errors are:

a. Words which are generally used in many forms. For example:

लिये for लिए, बहुयें for बहुएँ

b. Vowels which have similar sounds. For example:

ि for ी, े for ै, ं for ँ

c. (ङ, ञ, ण, न, म) are pancham Varna

Combination of consonant set with pancham Varna is represented by an anuswar over the consonant preceding it. For example: चञ्चल for चंचल, पण्डित for पंडित

v. **Split word Error:** When a space is inserted by mistake within the word. [1] In some cases, split word errors can also give rise to real word errors. For example:

आपकी for आप की

हमेशा for हमे शा

vi. **Cognitive errors:** These types of errors arise occur when a writer does not know or has forgotten the correct spelling of a word due to lack of knowledge about the Language

ग्यान for ज्ञान

vii. **Phonetically Similar Character Error**: These errors arise when the writer substitutes a phonetically correct but orthographically incorrect sequence of characters for the required word [1]. Following categories of phonetically errors are discussed.

Class 1:	न for ण, ग for घ, ज for झ, ब for व
Class 2:	ज़ for ज, ग़ for ग, ड़ for ड, क़ for क, फ़ forफ
Class 3:	चञ्चल for चंचल,पण्डित for पंडित
Class 4:	ु for ू,ं for ँ,े for ै, ि for ी, ो for ौ

Error Detection Approaches:

Dictionary Look up: Dictionary is a database of words. Its size may vary (small or large) according to number of words. This technique is used to check the availability of each input word in dictionary. If the word is found in the dictionary, the word is correct; otherwise it is not available in the dictionary. Software generates an error list even if it is a correct word but not found in the dictionary. Too small dictionary size gives many false rejections and too large can accept a high number of valid low-frequency words. Dictionary needs to be updated timely [3]. Fast Accessibility from dictionary is achieved using Hash Table. To retrieve the word from dictionary, a Hash Address is computed which is obtained from Hash Table. An error is indicated, if the word stored at Hash address is not available or mistyped.

N-Gram: N-gram is a procedure to check misspelled words in the document. N grams are of n-letters sub sequence where n can be one, two, three and so on. Unigrams are one letter n-grams whereas bi-grams are two letter n-grams and Trigrams are three Letter n-grams. The binary bi-gram is easiest two dimensional arrays which represent all possible two letter combinations of the alphabet. The value of each letter is either 0 or 1. These arrays are non-positional binary n-grams because the position of the n gram within a word is not shown. It is not comparing each entire word in a document to a dictionary. In the n-gram procedure there will be pre compiled word unigram and syllable bigram, trigram frequencies [3]. The n-gram statistics is relatively inexpensive to construct without deep linguistic knowledge and it can identify many letter errors.

Morphological analysis: This technique is developed by Fritz Zwicky. This technique is useful for Real world problems. In natural language processing the study of structure and formation of word is known as Morphology [7]. A term **Morphem,** which means **minimal unit of meaning.** For example: **Unableness**

Un + able + ness

Un means "not" used as **Prefix.**

Able means "capable of performing all the requisite duties" used as **Stem**.

Ness is used as **Suffix**

Morphological analysis is a process to verify the Actual (root) word. if the suffix satisfy the actual word then the word treated as correct, otherwise it is incorrect word and stopped.

Error Correction Approaches:

a. Split word Error correction:

If (according to dictionary check) two consecutive strings are non-words then we can merge them and check the merged string in the dictionary [2]. If it is a valid word, then split word error has been detected and corrected.

b. Merged word Error Correction:

Suppose a string is not a valid word, if portion of this (from left side) is a valid word and the rest is also a valid word. Then it is considered that a merged word error has been detected [2]. It is corrected by inserting a space in between these two words.

c. Minimum Edit Distance: The minimum number of editing operations (Insertion, Deletion, Substitution, Transpositions) needed for converting one string of characters into a valid word. This algorithm compares misspelled word with the dictionary, having m words. For this m number of comparisons takes place. During comparison process by Minimum edit distance algorithms need m comparisons between misspelled string and the dictionary of m words. Edit distance is also computes for choosing the correct one. The Symmetric Delete algorithm reduces the complexity of edit distance algorithm and dictionary lookup. For example:

Suppose typed word is **शमशाम**. By Edit distance it will be generate following words by deleting character one at time. This word will check in dictionary and similar to this dictionary will find **शमशान** word. Then Edit Distance will also be applied to this word. And suggestions for that word are given following.

Table 1: Minimum Edit Distance

Options generated for User typed word "**शमशाम**" By Edit distance	Options generated for Dictionary word "**शमशान**" By Edit distance
मशाम (by deleting "श")	**मशाम** (by deleting "श")
शशाम (by deleting "म")	**शशाम** (by deleting "म")
शमाम (by deleting "श")	**शमाम** (by deleting "श")
शमशम (by deleting "ा")	**शमशम** (by deleting "ा")
शमशा (by deleting "म")	शमशा (by deleting "न")

Now last generated option of user typed word is same as that of last option of Dictionary word. Now one Suggestion will be root of options of dictionary word, i.e. English. Minimum edit distance has different algorithms like **Levenshtein algorithm**, **Hamming**, **Longest Common Subsequence**.

i. The Levenshtein algorithm: The Levenshtein algorithm (also called Edit-Distance) calculates the least number of edit operations that are necessary to modify one string to obtain another string. The most common way of calculating this is by the dynamic programming approach. Levenshtein distance (LD) is a measure of the similarity between two strings, source string (s) and the target string (t) [5]. The distance is the number of deletions, insertions, or substitutions required to transform s into t. **Example 1:**

- If s is "test" and t is "test", then LD (s, t) = 0, because no transformations are needed. The strings are already identical.
- If s is "test" and t is "tent", then LD (s, t) = 1, because one substitution (change "s" to "n") is sufficient to transform s into t.

Example 2:

The Levenshtein distance between "**rosettacode**", "**raisethysword**" is 8;

Example 3:

Table 2: Levenshtein Distance

L	E		V	E	N	S	H	T	E	I	N
0	=	+	0	=	=	=	-	=	=	=	=

M	E	I	L	E	N	S		T	E	I	N

Match	"="
Substitution	"0"
Insertion	"+"
Deletion	"-"

In Example 3 LD =4

ii. **The Hamming algorithm:** This algorithm refers to measure the distance between two strings of equal length. The Hamming distance allows only substitution; hence, it only applies to strings of the same length. The hamming distance between **Put** and **But** is 1 (changing 'P' to 'B') and the Hamming distance between शाम and राम is 1 (changing 'श' to 'र').

iii. **The Longest Common Subsequence algorithm:** (**LCS**) **Problem** is the problem of finding the longest <u>subsequence</u> common to all sequences in a set of sequences (often just two sequences).

LCS Problem Statement: Given two sequences, find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous. For **e**xample:

- LCS for input Sequences "ABCDGH" and "AEDFHR" is "ADH" of length 3.
- LCS for input Sequences "AGGTAB" and "GXTXAYB" is "GTAB" of length 4.
- LCS for input Sequences 6750ABT4K9 and 0069TYA5L9 is"650AT9" of length 6.

d. **Similarity Key technique:** Similarity Key techniques define a string similarity measure based on features which are extracted from the input words. This technique is used to map each string into a key such that identical spelled strings will have same keys. Transformation of dictionary words and word to be tested take place into similarity keys. All words either they are from dictionary or being testes share same key. Soundex and Speedcop are examples of Similarity key technique.

i. **Soundex Algorithm**: As the name suggests soundex made up of Sound + Index. The Phonetic analysis is deals with how the sounds are produced when we talk and how words are related to sounds. Applying phonetic analysis to natural languages is involves making computer to understand how sounds are produced and analyzes by algorithm. Based on the phonetic sound of words Soundex indexes them. Grouping of Words with similar pronunciation but different meaning is performed. It is language specific mainly for names. e.g. Chebyshev ---→ Tchebycheff Chebyshev is a name which can spelled differently in different countries[6]. Soundex Indexing is performed using Hashing and same value is given to similar-sounding producing words.

ii. **The SPEEDCOP System**: It is a method of correcting spelling errors like typing errors in a huge database. Each word is processed with the key, which is computed for every word in the dictionary.

e. **Rule Based Technique:** Every language has its own set of grammar rules which are used to communicate via writing or speaking. Rule Based Techniques are algorithms that attempt to analyze the error in mistyped words, which do not follow grammar rules.

Problem: The major problem is that a Spell Checker developed for AnmolHindi font will work only for that font; it will not work for other font like Kruti Dev 010, because of mapping of keys. In different fonts, keys are also mapped differently for Hindi characters. For example

While we press **f+g+U+n+h** in **Kruti Dev 010** font, it types **fgUnh** (in Hindi)

While we press **f+g+U+n+h** in **AnmolHindi** font, it types **fgUnh** (in Hindi)

Table 3: Problems of Legacy fonts

S.No.	Font Name	Word	Series of keys pressed
1	Kruti Dev 010	fgUnh	f+g+U+n+h
	AnmolHindi	fgUnh	f+g+U+n+h
2	Kruti Dev 010	ty	t+y
	AnmolHindi	Ty	t+y
3	DevLys 060 Thin	"kjr	Shift '+k+j+r
	AnmolHindi	"kjr	Shift '+k+j+r

Therefore to overcome this problem, Spell Checker for Hindi language using UNICODE encoding technique had been developed. By using UNICODE encoding technique a single Hindi Spell checker can be used for many fonts (Embedded) of Hindi.

Characteristics of Hindi Complexities: There are many complexities in writing character in Hindi Language.

1. Following words are called Conjuncts. Conjuncts are formed form two or more characters. For example

i. द + ् + ध = द्ध ii. द + ् + म = द्म iii. द + ् + भ = द्भ

2. In Unicode ◌ा matra cannot be applied before any character

i. **kr** ii. **kC**

3. In Unicode Single character cannot have two or more matras at the same time. Formation of following words is not possible in Unicode.

i. क् + ◌ा ii. क् +ि

Objectives:

i. To develop or acquire a database of Hindi words in Unicode, which will serves as a repository for spell checker.
ii. To identify the most commonly used font for Hindi typing
iii. To develop an algorithm for conversion of Hindi legacy font to Unicode and vice-versa
iv. To cross check the font conversion to ensure that it produces the desired results.

Methodology:

A dictionary has been created for this software; it contains 120 Hindi legacy fonts. When user types a word, if the typed word is not in Unicode then firstly spell checker converts it into Unicode, later spell checker moves to match input word in dictionary,

- If word is found in dictionary, no error is generated by the spell checker but if input word is not available in dictionary then it is treated as an error. Further error is corrected by replacing the word with suggestions provided by spell checker. But before replacing that word with correct one, font is again converted back to its source font.
- Facility to add font is provided, if system does not support particular font.
- If the word is available in dictionary then spell checker skips that word and move on to next word in text.

Procedure for Font Conversion:
Spell Checker checks whether the input word is in Unicode or ASCII. If that input word founds in ASCII then it get converted into Unicode because the whole dictionary is available in Unicode format only. Then dictionary look up technique works on that word. At last selected word from dictionary is converted back to its source code.

Steps followed by spell checker:

- Dictionary Creation
- Pre-processing
- Error Detection
- Displaying the suggestion list
- Replacing the error with most suitable suggestion
- Display correct output.

II. Creation of Font Independent Database: For creation of font independent spell checker for Hindi, firstly database has been created of more than 1, 20,000 Hindi words. After that database of different Hindi fonts have been prepared with the help of Unicode characters. In this database all the fonts have been categorised into different classes. Categorization of classes is done on the basis of similarity of key mapping among different characters. After categorization of classes character mapping is done.

- **Font Classes:** Database has 120 fonts. Similar fonts are categorised into 65 classes on the basis of *key mapping*. Single class may contain more than one font. For example AmrHindi, AmarHindi, AnmolHindi and ApniHindi all these fonts are belong to single class because the characters of all these fonts have same key mapping.
- **Character Mapping:** As different fonts have different keyboard layouts, because all fonts are key mapped differently. For example, In AmrHindi font when small letter ‘x’ is pressed x appears and when capital letter ‘X’ is pressed X appears and In Kruti Dev 010 font, when small letter ‘x’ is pressed **x** appears and when capital letter ‘X’ is pressed **X** appears **.**

Table 4: Character Mapping

Font Name	X	X (capital)
AmrHindi	**x**	**X**
Kruti Dev 010	**x**	**X**

III. Font Conversion: In Indian Languages (Hindi, Bangla, Gurumukhi, Sanskrit, and Punjabi) different fonts have different keyboard layouts, so the inbuilt font conversion utility of Windows cannot be used to convert the font of an Indian Language text. The problem discussed in table 2, arises due to differences in character mapping among fonts. This issue is resolved by Font Conversion technique. Font Conversion is used to convert specified source font into specified destination font, by this technique input text of one font can be converted into another one without any loss of text or information. To accomplish Font Conversion Unicode Encoding Technique has been used as an intermediate.

Unicode is a worldwide standardised Encoding technique [4]. To have specified results Algorithm of conversion from Unicode to ASCII and vice versa has played a very powerful role in this whole Project.
Font Conversion can perform operation like conversion of one font to another font, like other English font word processors for e.g. MS-Word, Open Office. Microsoft Word and other word processors support English to English font conversion because of same key mapping. But these word processors do not support multiple Indian Languages' font to font conversion because of different key mapping.

Algorithm of Conversion from Legacy to Unicode font:
1.) First of all, identify the legacy font.
2.) Identify its class using that font from database.
3.) Using that class and its character codes get corresponding Unicode characters.
4.) Refine Unicode word.
5.) Exit.

Refinement of word in Unicode:
Hindi in Unicode is dynamic not positional. E.g. To write प्रसिद्ध in Unicode following sequence is followed.

Table 5: Refinement of word in Unicode.

1	2	3	4	5	6	7	8
प	्	र	स	ि	द	्	ध

Algorithm of Conversion from Unicode to Legacy font:
1.) First of all, identify the Unicode font.
2.) Identify its class using that font from database.
3.) Using that class and its character codes get corresponding legacy characters.
4.) Refine legacy word.
5.) Exit.

Refinement of word in Legacy:
Hindi in Legacy is positional not dynamic. E.g. To write i~fl) in Legacy following sequence is followed.

Table 6: Refinement of word in Legacy.

1	2	3	4	5
I	~	F	L	)

IV. Font Independent Spell Checker for Hindi:

Font independent spell checker for Hindi works on all Hindi fonts. It checks whether that font is available in the database or not. If font is not available in the database, then the spell checker provides a facility to the user to add new font in the database. The whole dictionary is in Unicode format. If the input word is not in Unicode format, first of all Spell Checker converts it into Unicode and then look for that word in the dictionary. If word is available in the dictionary, spell checker skips that word and repeat same procedure to next token. If word is not found in the dictionary, suggestions come up. User replaces that word with one of the Unicode suggestions. In case source (input) font is ASCII, in that case selected correct word from Unicode suggestions also gets converted into source font.

The basic architecture of a font independent spell checker for Hindi is segregated into two modules, first is error detection module which detects all the misspelled words in given text document , it also checks whether the given input token is in Unicode or non-Unicode format because whole dictionary is stored in Unicode format. If input text is not in Unicode Format then conversion takes place from non-Unicode to Unicode. Second is error correction module which provides list of suggestion for the misspelled words and rank the suggestion list, the detailed functionality of a font independent spell checker for Hindi is to be explained using below flowchart.

Architecture of Font Independent Spell Checker for Hindi

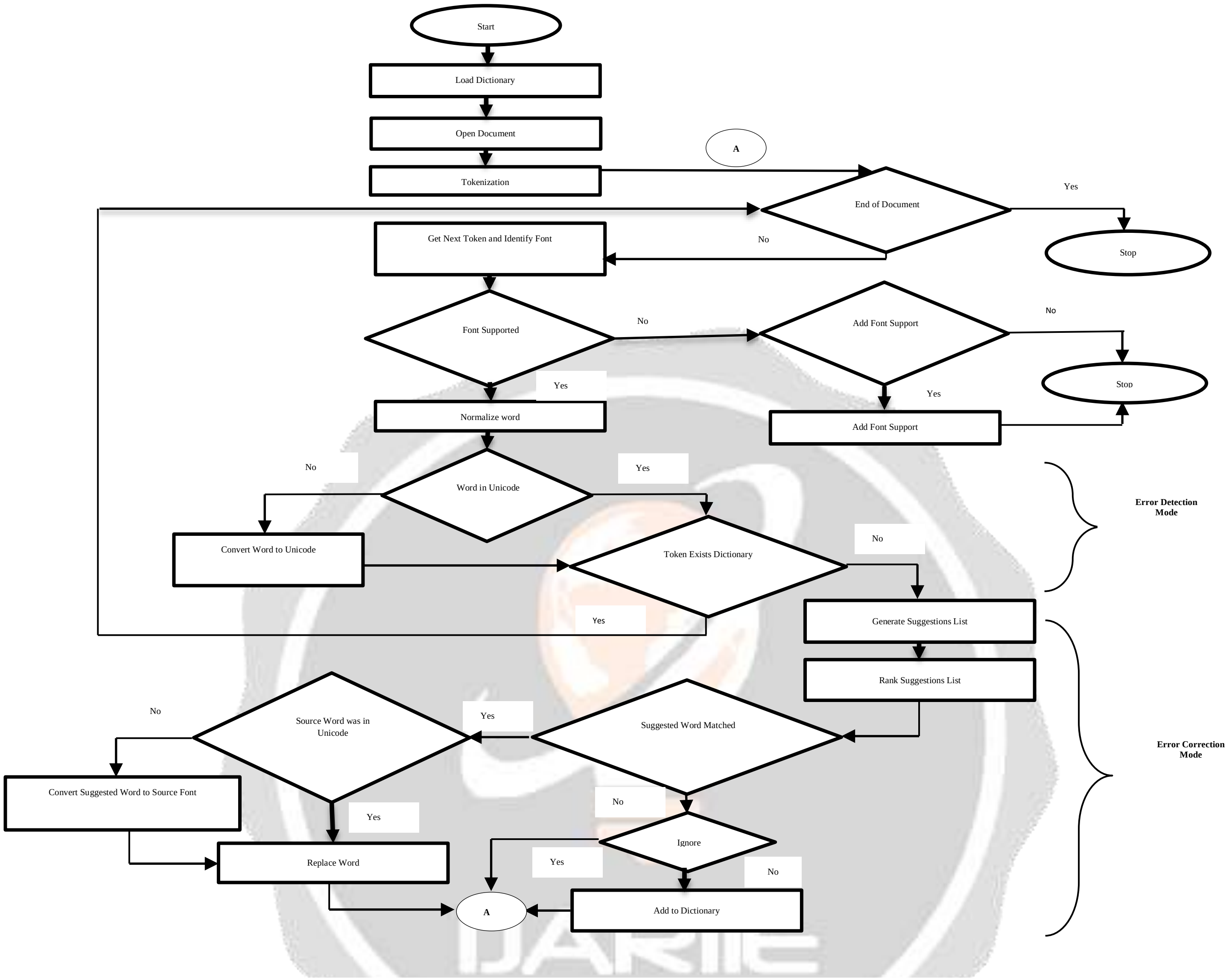

Fig. 1: Basic Architecture of Font Independent Spell Checker for Hindi

The working of basic architecture of Font Independent Spell Checker for Hindi is as follows:

1.) First of all, tokenization is done for the input, so that all the words could be processed.

2.) Check whether it is an end of the document, if it is yes then it will stop the process. Otherwise it will go to get one token and identifies its font.

3.) If the font of selected token is supported then it will go to step 4, otherwise it will give option to user whether he wants to add that font to dictionary. If the user wants to add the font then module for addition of font support in the database will be activated and it will add the font and exit. Otherwise, exit directly.

4.) Normalize the token this technique is used to convert the format of the token same as that of dictionary words. If this step is skipped then the correct words would be treated as a misspelled word. There are some characters whose length is one but, in other encoding the character length for the same character is two. For example

1.) श + ् + र = श्र　　　2.) स + ् + र = स्र　　　3.) क + ़ = क़

5.) If the token's font is in Unicode then move ahead else convert the token to Unicode.

6.) Check for presence of the Unicode token in dictionary. If the token is present in the dictionary then it's the correct token and go to step 2.

7.) If the word is not present in the dictionary, then it is detected as misspelled word and the program moves to the error correction mode. In the error correction mode, a list of suggestions is generated to replace the misspelled word.

8.) In next step, Ranking of the suggestion list is done so that the most appropriate suggestions appear on the top of the suggestions list.
9.) Check for correct suggestion from the suggested words list. If it is not found then either ignore the current token or add it to dictionary and go to step 2.
10.) If correct suggestion is found then check whether source token was in Unicode.
11.) If the source token was in Unicode, then the user can either replace the misspelled word or make no change and go to step 2.
12.) If the source token was not in Unicode, then convert the suggested word from Unicode to source font format and replace the misspelled token and go to step 2.
13.) Repeat the steps 2 to 12 for every token of the document.
14.) Exit.

Conclusion

Before this Research, a lot of work has been done in Hindi Spell Checker. But all that Spell Checkers were font dependent. Font dependent means, if a Spell Checker is developed for Mangal font then it will work only for that font. It does not work for any other font. Font Independent Hindi Spell Checker as the name indicates is an independent of all Hindi Fonts like Mangal, Krutidev, AmrHindi, Amar, Ravi etc. This is accomplished by the Unicode Encoding technique.

Basic Features of Font Independent Spell Checker are:

- The first main feature of this dissertation is the conversion of ASCII to UNICODE and vice-versa.
- Second feature is user gets same font of token to replace as that of its input string.
- All Dictionaries is in UNICODE.
- In case font is not installed, Facility to add font is provided.
- Its database contains about 1, 20,000 words in dictionary.
- These words are collected through different sources including websites and digital documents.
- This software supports 120 different Hindi Fonts.
- These fonts classified into 65 classes based on their character mapping.

Future Scope

- Problem of the errors Insertion, Deletion, Substitution, Transposition, Merge are handled safely except Split Error.
- For small words (having 2 to 4 characters), the number of generated suggestions are very large. This is due to Edit distance Algorithm.
- This software can be enhanced by making it Bilingual.

REFERENCES

[1] Shivani (2013) “Hindi Spell Checker”, M.Tech Synopsis Report, Punjabi University, Patiala.
[2] M.Ritika and K. Navjot (2013) “A Survey of Spelling Error Detection and Correction Techniques” International Journal of Computer Trends and Technology- volume4Issue3
[3] http://nlp.stanford.edu/IR-book/html/htmledition/edit-distance-1.htm
[4] Nirma Garg (2010) “Font Independent Spell Checker using Gurmukhi script”, M.Tech Thesis, Punjabi university, Patiala.
[5] http://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Fall2006/Assignments/editdistance/Levenshtein%20Distance.htm
[6] http://nlp.stanford.edu/IR-book/html/htmledition/phonetic-correction-1.html
[7] https://www.cs.bham.ac.uk/~pjh/sem1a5/pt2/pt2_intro_morphology.html