Automatic Generation of Compound Word Lexicon for Hindi Speech Synthesis

Deepa S.R.*, Kalika Bali†, A.G. Ramakrishnan, Partha Pratim Talukdar†

†Hewlett-Packard Labs
24 Salarpuria Arena, Hosur Road, Bangalore, India
{kalika.bali, partha.talukdar}@hp.com

*Birla Institute of Technology & Science
Pilani, Rajasthan, India
f2000073@bits-pilani.ac.in

Indian Institute of Science
Bangalore, India
ramkiag@ee.iisc.ernet.in

Abstract

This paper addresses the problem of Hindi compound word splitting and its relevance to developing a good quality phonetizer for Hindi Speech Synthesis. The constituents of a Hindi compound word are not separated by space or hyphen. Hence, most of the existing compound splitting algorithms can not be applied to Hindi. We propose a new technique for automatic extraction of compound words from Hindi corpus. Preliminary tests conducted on the algorithm have shown a split rate of 92 to 96% of the input compound words. Of these splits, around 83 to 87% are correct splits. A few modifications have been suggested, which will improve the accuracy of the splits. Finally, we observe an improvement of 1.6% in Hindi Grapheme-to-Phoneme (G2P) conversion as a result of using a phonetized compound word lexicon, created by the above technique.

1. Introduction

Compound words are formed when two or more words are concatenated into a single word. Compounding is a highly productive word formation technique in Hindi. Compounding is also a common phenomenon in Greek, German, Finnish etc. Table:1 gives examples of a few Hindi compound words.

Compound Word	Constituents
loksabhA	*lok sabhA*
hawAijahAj	*hawAi jahAj*
kAlApAnI	*kAlA pAnI*

Table 1: Examples of Hindi Compound Words

Compound word lexicons play an important role in various domains of language technologies *e.g.* speech recognition (Ordelman et al., 2003), (Adda-Decker, 2003), machine translation (Koehn and Knight, 2003). Identifying compounds is also necessary for assigning stress patterns correctly as stress patterns for compounds differ greatly from equivalent single word units.

In a Text-To-Speech (TTS) synthesis system, the G2P module converts normalized input orthographic text into underlying phonetic representation. Accurate phonetic transcription is highly desired for natural sounding speech synthesis. Compound word splitting plays a crucial role in Hindi G2P conversion, specifically for solving the schwa deletion problem (described in Section 2.).

2. Schwa Deletion

Schwa deletion is a unique problem encountered in Hindi G2P conversion. Each consonant in Devanagari, the script used to write Hindi, is associated with an inherent schwa which is not represented in orthography. In some cases, this associated schwa is deleted depending on certain morpho-phonological factors and in others, it is retained . The written text, however, does not provide a direct clue about the deletion or retention of the schwa, thereby making it a challenging problem to address. Hindi word *kalam* "pen", for example, is represented in orthography using the consonantal characters for *k, l, m*. The schwas (*a*) are inserted by the speaker while speaking out the word. Vowels other than schwa are explicitly represented in orthography.

A set of rules for schwa deletion in Hindi are reported in (Narasimhan et al., 2001). The rules are based on the morpheme boundaries present in words. Word internal morpheme boundaries can be detected using a morphological analyzer. Unfortunately, high quality Hindi morphological analyzers are currently not available. Unavailability of such analyzers restricts the applicability of these rules.

Another set of rules were implemented in the Dhvani (Dhvani, 2001) speech synthesis system. These rules work well for simple words but fail in the case of compound words. For example, Hindi compound word *loksabhA* "lower house of parliament" is obtained by joining two words: *lok* "people" and *sabhA* "gathering". In orthography, the word is represented using consonant and vowel forms of *l, o, k, s, bh, A*. Application of schwa deletion rules on this word would produce [l o k a s bh A] which is incorrect. Correct phonetic transcription [l o k s a bh A] is obtained if the two words are analysed separately. Hence, a lexicon of compound words along with their correct phonetic representation is very important for accurate phonetic conversion of input text.

3. Previous Work

Corpus driven method for compound splitting using a parallel corpus is reported in (Brown, 2002). Compound splitting and recombination for reducing out of vocabulary (OOV) words in large vocabulary speech recognition systems is reported in (Larson et al., 2000). An approach to learn compound splitting rules from monolingual and parallel corpora and its impact on statistical machine translation systems are reported in (Koehn and Knight, 2003).

Statistical compound extraction techniques are reported in (Zhang et al., 2000), (Su et al., 1994). These methods require the constituents of a compound word to be space separated. Such methods are not applicable for Hindi compounds since constituents in a Hindi compound are not separated by space. To overcome this problem, a compound splitting algorithm has been developed.

4. Compound Splitting Algorithm

The compound extraction algorithm takes as input a text corpus and generates a lexicon with the compounds split into its constituent parts. The algorithm starts with the assumption that independently occurring words are valid atomic words.

A trie-like structure is used to store and efficiently match the words. Without loss of generalizability, the following description considers a compound word to be made up of two constituents. A *potential* compound word is detected if the currently processed word is part of a word already present in the trie or a word in the trie is a substring of the current word. For compounds with more than two components, the algorithm can be iteratively applied generating all constituent components. For each word (α), there are several possibilities:

- *Case 1:* α is not present in the trie and is also not a constituent of any of the words currently present in the trie.
- *Case 2:* α is already present in the trie as an independent word.
- *Case 3:* α is actually the initial constituent of a compound word currently present in the trie as an independent word.
- *Case 4:* α is actually the second constituent of a compound word currently present in the trie as an independent word.

In *Case 1*, the algorithm inserts α into the trie. In *Case 2*, no change is needed since α is already present in the trie as an independent word. In *Case 3*, the algorithm generates the second constituent (β) for each of the potential compound words where α is the first constituent. After the generation of β's, the end node corresponding to α in the trie becomes a leaf node. For each of the generated β's , the algorithm checks for its presence in the trie. If β is present in the trie, the algorithm marks the combination (α β) as a compound word. If β is not present in the trie, the combination (α β) is inserted into the suspicion list. The algorithm in its present form does not locate *Case 4*, since the processing is performed left to right.

Before processing each α, the algorithm checks for its presence in the suspicion list. If α is present in the suspicion list as the second constituent, the entry (γ α) is removed from the suspicion list and is marked as a compound word. After this, α is matched against the trie contents and actions are taken as per the case (*i.e. Case 1, 2, 3, 4*) to which α belongs.

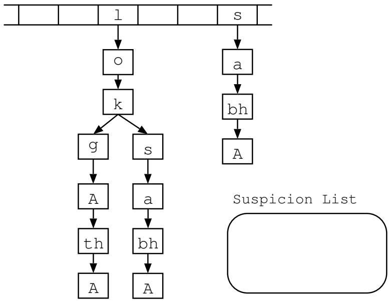

Figure 1: Before processing the word *lok*

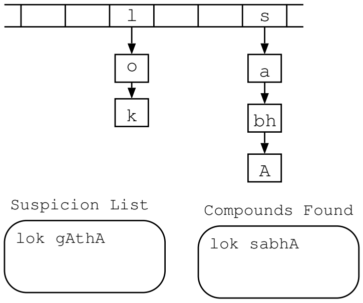

Figure 2: After processing the word *lok*

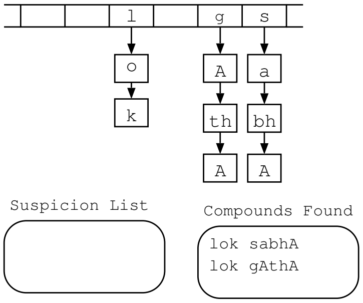

Figure 3: After processing the word *gAthA*

To increase the number of potential compounds in the suspicion list, the algorithm can be run with reversed words. The forward pass (*i.e.* left to right processing of a word in its original form) is enough to split a compound word if all its constituents are present in the corpus as independent words.

To illustrate the algorithm, let us consider a sample lexicon with words in the sequence *lokgAthA* "folk tales", *loksabhA* "lower house of parliament", *sabhA* "gathering", *lok* "people", *gAtha* "tale". If both constituents of a compound are present in the corpus, then the order of reading the compound and its constituent parts is irrelevant. Fig. 1 represents the trie contents after the words *lokgAthA*, *loksabhA* and *sabhA* have been read in. Suppose the word *lok* is input after this. The algorithm first checks for it in the suspicion list. After not finding it there, the word is matched against the contents of the trie generating the constituent forms *gAthA* and *sabhA*. *sabhA* is already present in the trie but *gAthA* is not yet read in. Hence, *lok sabhA* is marked as a compound word. But since *gAthA* is not present in the trie, *lok gAthA* is inserted into the suspicion list. This state is shown in Fig: 2.

Suppose the algorithm comes across the word *gAthA* later. *gAthA* is present in the suspicion list as the second constituent. Hence, *lok gAthA* is removed from the suspicion list and is marked as a compound word. Then the algorithm matches *gAthA* in the trie and since it is not present, it is inserted into the trie.

Hence, at the end of the algorithm, the best possible atomic word forms are present in the trie with compounds decomposed into their constituent parts.

4.1. Hindi Post-processing

4.1.1. Stray Characters & Affixes

The algorithm described above is vulnerable to stray characters and affixes present in the corpus. Such characters can trigger wrong splits and also increase false positive rate. For example, *pra* is a prefix in Hindi. Ideally, *pra* can not occur as an independent word. However, due to typographical errors or for other reasons, presence of such character combination in a text segment cannot be ruled out. If *pra* is present, the algorithm wrongly splits *pranAm* into *pra* and *nAm* "name" (assuming *nAm* is also present in the text). Actually, *pranAm* is not a compound word. Hence, special care should be taken so that the algorithm doesn't consider affixes as valid words. A possible solution to this problem can be the use of a list of affixes. The algorithm can check the affix lexicon for the presence of each constituent and mark the word as a compound word only if none of the constituents is present in the affix list.

4.1.2. Length Based Heuristics

Length based heuristics can be used to keep away stray characters from being considered as valid words by the algorithm. But this strategy does not work very well for Hindi. In Hindi, words with very few characters have the potential to form compounds words. For example, the two character word *nav* "new" can form an array of compound words such as, *navvarsh* "new year" and *navjivan* "new life". In the current system, only single character words are treated invalid.

4.1.3. Consonant-Vowel (CV) Pairs

Consonant-vowel (CV) combinations can occur independently in Hindi text but they cannot be a constituent of a compound word. Hence, the algorithm should check that none of the generated constituents is a CV pair. Examples of valid Hindi CV pairs are *ne* and *ki*.

4.1.4. Problems Related to Word-forms

Another problem associated with Hindi compound splitting is related to root word and its different forms. Let us take the example of the Hindi root *hawA* "wind" and one of its word-form *hawAi* "windy" which is a constituent of the compound word *hawAijahAj* "aeroplane". Constituents of the compound *hawAijahAj* are *hawAi* and *jahAj*. Let us consider the case when the algorithm reads the words in the sequence *hawAijahAj*, *hawA*, *hawAi* and *jahAj*. After reading the first two words, the algorithm will split it into the parts *hawA* and *ijahAj* and insert the combination into the suspicion list since the existence of the second constituent is not yet known. *ijahAj* is not a valid Hindi word. Hence, even though both the constituents are present, the algorithm will fail in splitting the compound *hawAijahAj* into its correct constituent parts. A possible solution to this problem can be the detection of different splitting points and subsequently selecting the best split based on probabilistic measure. This feature, however, is currently not incorporated into the current system.

5. Experimental Results

In the first experiment, the compound extraction algorithm was used to generate a compound word lexicon. The experiment was carried out to observe the number of compound words extracted by the algorithm from a given text segment.

Total Words In Text Segment	2420329
Total Unique Words	246325
Total Unique Words Marked as Compound	48428
Proportion of Compound Words (as detected)	19.66%

Table 2: Performance of our split algorithm on a text segment

A second experiment was carried out to study compound splitting accuracy. A compound word lexicon was generated using the algorithm described above. Final trie contents were also dumped since these words are the atomic word forms. Each word in the compound word lexicon forms one of the constituents of a compound word. The contents of the trie were merged with the compound word lexicon generating a combined lexicon. This experiment, in a way, tested the quality of the combined lexicon in terms of its coverage of independent word forms. Two sets of compound words (50 compound words each) were manually prepared. These lists were prepared without any knowledge of the words present in the generated compound word lexicon. A variant of the algorithm described above was used in this case. The algorithm first loaded all the words in the combined word lexicon. After this, words from the test files were matched against the loaded words and were split accordingly. The algorithm was allowed to cause splits only in the test words. The results are shown in Table:3.

Compound words for which all the constituents satisfied different criteria for independent words are included in the High Confidence List. Compound words in the low confidence list are the words for which the algorithm could not find one of its constituents in the corpus. These are the words which are finally present in the suspicion list. In Table:3, compound split precision and compound extraction rates are calculated based on the words included in

	Set 1	Set 2
Total Words	50	50
Words in High Confidence List	42	43
Words in Low Confidence List	6	3
Compounds Correctly Split	40	40
Compound Split Precision Rate	83.3%	86.9%
Compound Extraction Rate	96%	92%

Table 3: Accuracy of compound word split using our algorithm.

both High & Low confidence lists. For example, in *Set 1*, the algorithm marked 48 of the total 50 words as compound words achieving a compound extraction rate of 96%. Out of these 48 (42 in high confidence list and 6 in low confidence list) marked compound words, 40 compound words are correctly split. Hence, the split precision rate is 83.3%.

The third experiment was carried out to study the improvement in Hindi Grapheme-to-Phoneme conversion resulting from the incorporation of the phonetic compound word lexicon into the Hindi G2P converter (Ramakrishnan et al., 2004). A section of the Emille corpus (Emille, 2003) was randomly selected. The selected text segment was phonetized using Hindi G2P converter developed as part of the LLSTI (Tucker, 2003) initiative. Words which were present in the phonetic compound lexicon were also analysed using rules and the two phonetic transcripts were manually compared. The results are shown in Table:4.

Total Words	3497
Words Phonetised Using Lexicon	252
Lexicon Correct but Rule Incorrect	65
Rule Correct but Lexicon Incorrect	10
Lexicon and Rule Both Correct	173
Lexicon and Rule Both Incorrect	4

Table 4: Result of Hindi G2P Conversion

Effectively, phonetization of 55 words improved after incorporating the phonetic compound word lexicon into the Hindi G2P. Hence, a net improvement of 1.6% in Hindi G2P conversion is observed as a result of using the compound word lexicon generated by the algorithm presented in this paper. Moreover, out of the total 252 words phonetised using the lexicon, an improvement of 21.8% is observed.

6. Conclusion

An effective algorithm has been proposed for splitting compound words in Hindi. The algorithm has been tested and found to be effective in splitting above 90% of the input compound words. Of these splits, around 85% are found to be correct. One of the possible approaches to increase the accuracy of the split is to allow for multiple splits (at different points in the same word) of every word, by not removing any suspect compound word from the trie. To get more potential compound words, the same algorithm can be applied a second time, after reversing each word, so that the second constituent of each compound word can be identified first. A near-exhaustive list of affix words of the language can be deployed to minimize or altogether eliminate wrong splits on account of prefixes and suffixes.

7. References

Adda-Decker, Martine, 2003. A corpus-based decompounding algorithm for German lexical modeling in LVCSR. In *ISCA Eurospeech*. Geneva.

Brown, Ralf D., 2002. Corpus-Driven Splitting of Compound Words. In *Proceedings of the 9th International Conference on Theoretical and Methodological Issues in Machine Translation*.

Dhvani, 2001. Dhvani - TTS System for Indian Languages. (http://dhvani.sourceforge.net).

Emille, 2003. The EMILLE (Enabling Minority Language Engineering) Project. (http://www.emille.lancs.ac.uk).

Koehn, Philipp and Kevin Knight, 2003. Empirical Methods for Compound Splitting. In *EACL 2003*. Budapest, Hungary.

Larson, Martha, Daniel Willett, Joachim Köhler, and Gerhard Rigoll, 2000. Compound splitting and lexical unit recombination for improved performance of a speech recognition system for German parlianmentary speeches. In *6th Int. Conference on Spoken Language Processing (ICSLP)*. Beijing, China.

Narasimhan, B., R. Sproat, and G. Kiraz, 2001. Schwa deletion in Hindi Text-To-Speech Synthesis. In *Workshop on Computational Linguistics in South Asian Languages*.

Ordelman, R.J.F., A.J. van Hessen, and F.M.G. de Jong, 2003. Compound decomposition in Dutch large vocabulary speech recognition. In *Eurospeech 2003*.

Ramakrishnan, A.G., K.S.R. Anjaneyulu, Kalika Bali, and Partha Pratim Talukdar, 2004. Towards the Development of a Hindi Speech Synthesis System (unpublished).

Su, Keh-Yih, Ming-Wen Wu, and Jing-Shin Chang, 1994. A corpus-based approach to automatic compound extraction. In *32nd Annual Meeting of the Association for Computational Linguistics*. New Mexico.

Tucker, Roger, 2003. The Local Language Speech Technology Intiative (LLSTI). (http://www.llsti.org).

Zhang, J., J. Gao, and M. Zhou, 2000. Extraction of Chinese compound words – an experimental study on a very large corpus. In *Proc. of the 2nd Chinese Language Processing Workshop, ACL*.