Recursive Suffix Stripping to Augment Bangla Stemmer

Md. Hanif Seddiqui*, Abdullah Al Mohammad Maruf*, and Abu Nowshed Chy†
* Department of Computer Science and Engineering
University of Chittagong
Chittagong-4331, Bangladesh
†Department of Computer Science & Engineering
Toyohashi University of Technology
Toyohashi, Aichi, Japan
hanif@cu.ac.bd, maruf.cu.cse09@gmail.com, and nowshed@kde.cs.tut.ac.jp

***Abstract*—Stemming is an operation on a word that simply extract the main part possibly close to the relative root, *we define as a lexical entry rather than an exact morpheme*, by eliminating the suffixes, as they mean, without doing a morphological analysis. Stemming is a widely used, however, a basic tool for a language processing in the area of Information Access (IA). Previous research on Bangla stemming has shown the capability of eliminating a single suffix on a 'longest match' basis from a word. Our proposed system has augmented the research through introducing a recursive process to eliminate multiple suffixes from a single word to retrieve more relevant relative root. Our proposed algorithm contains three approaches in recursive suffix stripping based stemming. We define them as *conservative*, *aggressive*, and *rule-based* approaches. In our proposed method, an inflectional word is stemmed in all possible ways by the recursive suffix stripping algorithm before identifying the final stem using the conservative, the aggressive and the rule-based approaches. Our experiments and evaluation on a relatively larger text corpora show the strength and efficiency of our proposed algorithm with 92% accuracy.**

***Keywords*—Bangla Stemmer, Natural Language Processing (NLP), Information Access, Information Processing**

I. Introduction

Natural language often contains inflectional terms or variants coming off a single lexical element, called stem. A stem is often associated with an article, number, and other suffixes to form morphological variances in a language. Stemming is an operation that splits a terms into its constituent main part and a single or a number of suffixes. Splitted main part is not necessarily be the lexical root rather than it is another lexical word close to the root. We call this main part of a word as a relative root. Therefore, stemming is a different technique than morphological analysis that splits a term into its constituent root part and affix without doing complete morphological analysis.

Terms with common stems tend to have similar meaning, which makes stemming an attractive option in information retrieval (IR) domain. It improves the performance through augmented term-weighting, document-weighting, query-document matching, document indexing, and retrieval with query terms. Bangla is a highly inflectional language that contains tens of morphological variances on a single stem. However, a few study has addressed this issue in Bangla Information Retrieval.

There are a number of stemming techniques to retrieve word-stem in English. One of the simplest technique utilize a list of frequent suffixes to reduce their relative roots [1], [2]. A more detailed evaluation of stemming algorithms [3] has revealed the fact of significant improvement in the recall due to stemming techniques against English documents. Other languages have similar stemmer to assist their IR area of research. For the Dutch language, the suffix stripping is evaluated to find out the effectiveness of stemming [4]. A detailed survey has been conducted on existing techniques of stemming Indonesian words to their morphological roots [5]. Moreover, Punjabi language stemmer [6], Malaysian Malay language stemmer [7] has been focusing on their languages. A more detailed survey on stemming of Asian languages has been addressed in [8]. Ramanathan and Rao (2003) propose a lightweight stemmer for Hindi which has performed longest match stripping with a handcrafted suffix list [9]. Other stemming algorithms for Information Retrieval (IR) applications have been developed for different languages including Latin [10], Indonesian [11], Swedish [12], Dutch, German and Italian [13], French [14], Slovene [15], Turkish [16] and so on.

The Bangla stemming algorithms strip the suffixes using a predetermined suffix list. Dasgupta and Ng [17] addresses unsupervised morphological parsing to segment words into prefix, suffix, and stem. Islam et al. [18] focus on a 'longest match' basis, whereas Majumder et al. [19] uses clustering approach to strip suffixes. Bangla is such an inflectional language that has a tendency of containing more than one suffices in a single stem. Therefore 'longest match' may be inadequate to eliminate all suffixes from morphological variances. To overcome this limitation, we propose a recursive process to extract all possible stems. In our proposed stemming technique, a candidate word is stemmed using the minimal list of non-redundant suffixes recursively, which produces a list of all possible stems for further analysis to detect a proper stem.

The rest of the paper is organized as follows. **Section II** compares our idea with other existing related work to articulate a research gap. **Section III** describes our proposed algorithm along with some comprehensive examples of Bangla stemming. **Section IV** includes experiments and evaluation to show the effectiveness of our proposed stemming technique against a relatively large corpus. Concluded remarks and some future directions of our work is elaborated in **Section V**.

II. Related Work

We have a few stemmer in Bangla: Dasgupta and Ng's morphological parser [17], a light weight stemmer of Islam et al [18]. and Yet Another Suffix Stripper (YASS) of Majumder et al [19].

Dasgupta and Ng (2006) proposed unsupervised morphological analysis parsed the root from Bangla words. It tried to segmented words into prefixes, suffixes and stems without language-specific prior knowledge of morpho-tactics and morpho-phonological rules. Authors defined the parser in two steps: inducing prefixes, suffixes and roots from a vocabulary of a large, unannotated corpus, and segmenting morphological variances based on these induced morphemes. The authors evaluated their system against 4,110 human-segmented Bangla words, and they claimed that their algorithm obtained F-scores of 83% that substantially outperformed Linguistica, one of the most widely-used unsupervised morphological parsers, by about 23%. [17]

Islam et al. (2007) proposed a fancy lightweight stemmer for Bangla and its use in spelling checker with a similar approach as proposed in [9]. The authors developed a suffix list and their proposed algorithm stripped the suffixes using the predetermined suffix list. It stripped a morphological variant based on the 'longest match' basis. They had collected a total of 72 suffixes for verbs, 22 for nouns and 8 for adjectives for Bangla language. The proposed stemming algorithm was primarily used for operating on inflections reducing derivationally related terms to the same stem. [18]

Yet Another Suffix Stripper (YASS) was a statistical approach developed by Majumder et al. (2007) [19]. The proposed algorithm used clustering techniques based on string distance measure. The authors claimed that it required no prior linguistic knowledge to improve recall of Information Retrieval (IR) systems against Indian languages. YASS was based on string distance measure. It managed to cluster a lexicon from a text corpus into homogeneous groups. Each group was expected to contain similar morphological variants of a single root word. They used Graph-theoretic clustering algorithm in their experiments.

III. Proposed Method for Bengali Stemming

Bangla words may contain multiple suffixes in a single stem. For example, the word "nirapottahInotai" contains three different suffixes 'hIno', 'ta', 'i'. In these cases, a 'longest suffix based' single stripping may not be sufficient. Therefore, we propose a recursive suffix stripping. We have developed a novel stemming technique based on three different approaches: conservative, aggressive, and rule-based. In conservative approach, we always ensure the stripped word to be in the predefined root list. However, aggressive approach always tries to eliminate all possible suffixes available in the word. The rule-based approach considers those suffixes whose identified patterns are different than the stripping patterns. Each of them has been elaborated in the following subsections. Before going into details of our algorithmic approach, we concentrate on the study of Bangla suffixes.

A. Suffix Study

Bangla is a highly inflectional language with many inflected forms of verbs, noun, adjective, and adverbs [18]. However, we have observed the suffixes from three different aspects. Some suffixes identify a specific class of words, like verbs. Some words contain pseudo-suffixes, which are not actually suffixes, rather they are part of the words. We have identified this type of suffixes and their stems or relative roots. We take these suffixes into consideration while working with conservative technique. We have considered 105 suffixes for our conservative approaches.

Moreover, we have observed a good variety of words, coming from noun origin or different language family contains 218 different suffixes for their variations due to articles, numbers, parts-of-speech conversion, and so on. These suffixes are used in aggressive approach of stemming.

Apart from this, there are a few suffixes, which have different nature than the suffixes described above. In case of these suffixes, we use a pattern in suffix identification, however, we hardly strip the whole suffixes. We strip quite differently from the word. For example, 'borrShay'. Here we identify 'ay', but we strip only 'y'. We use these suffixes in the rule-based approach. For the time being, we have used three different suffixes in this rule-based approach.

B. Recursive Suffix Stripping Algorithm

As we already described a word 'nirapottahInotai', which contains three suffixes. Some other word may contain a single, or two suffixes. In order to remove the undefined number of suffixes, we have introduced a recursive **Algorithm 1** to strip the suffixes repeatedly from the morphological variances to extract stem.

Algorithm 1: rsStripping($word$, $suffixes[]$, $stems$): An algorithm to strip all possible suffixes from a word

```
Input: word:to be stripped; suffixes[]: suffixes to be
       stripped; stems: vector of all possible stems
Output: Candidate stems list
1  if (word.length()<3) then
2      return;
3  for each suffix ∈ suffixes do
4      if (word.endsWith(suffix)) then
5          stem=word.strip(suffix)
6          if (stem.length()<2) then
7              continue;
8          if (lastChar(stem)==0x09CD) then
9              continue;
10         stems.add(stem)
11         rsStripping(stem, suffixes[], stems)
12 end
```

According to Recursive Suffix Stripping **Algorithm 1**, we take two data as input, ($word$ to be stripped and list of suffixes) and one reference address (call-by-reference) that contains a possible list of $stems$. The base criteria of the recursive function is stated at line 1. Then a simple iteration starts for all suffixes (refer to line 3). If the $word$ contains a suffix at the end (see line 4), algorithm strip it from the word to make a stem (line no. 5). However, if the length of the stem becomes less than 2 (as line 6), or the last character is 'hosonto' (refer to line 8) then the stem is not acceptable and starts processing for the next suffix. Otherwise, the stem is added to the stems vector (line no. 10) as a candidate stems and the function starts recursively for the stem word (see the line no 11).

C. Conservative Stemming

In conservative stemming, we use the Recursive Suffix Stripping **Algorithm 1** with the predefined list of suffixes. Then the **Algorithm 1** produces a vector or a list of candidate stems. Here, we only consider the stems, which are available in the root repository. This approach is called conservative as it only accepts the predefined set of forms (or roots). Among the short-listed stems, the stem with the smallest length is returned. Currently, we have 459 verb stems and 28 other stems in the root repository.

D. Aggressive Stemming

Aggressive stemming approach uses the same Recursive Suffix Stripping **Algorithm 1** with a different set of predefined suffixes. Similar to the conservative approach, the **Algorithm 1** produces a list of candidate stems. Unlike conservative approach, we consider all the stems and find the smallest length stems. In fact, the largest suffix has been removed aggressively. That is why, this approach is called as an aggressive approach.

E. Rule-based Stemming

Unlike conservative or aggressive approach, rule-based stemming is primarily applied for a separate list of suffixes. In reality, suffixes used in this list usually found at the end of a word. Therefore, we apply the first iteration of the algorithm as the rule-based approach, however, it becomes nothing but an aggressive approach after the first iteration. Evidently, rule-based stemming approach takes the essence of the aggressive approach.

Algorithm 2: ruleBasedStripping($word$, $rsuffixes[][2]$, $asuffixes[]$, $stems$): An algorithm to strip all possible suffixes from a word

```
Input: word:to be stripped; asuffixes[]: aggressive
       suffixes to be stripped; rsuffixes[][2]: suffixes
       for rule-based approach; stems: vector of all
       possible stems
Output: Candidate stems list
1  if (word.length()<3) then
2  |  return;
3  for each suffix-pair ∈ suffixes do
4  |  if (word.endsWith(suffix[i][0])) then
5  |  |  stem=word.strip(suffix[i][1])
6  |  |  if (stem.length()<2) then
7  |  |  |  continue;
8  |  |  if (lastChar(stem)==0x09CD) then
9  |  |  |  continue;
10 |  |  stems.add(stem)
11 |  |  rsStripping(stem, asuffixes[], stems)
12 end
```

According to Rule-based **Algorithm 2**, we take three data as input, ($word$ to be stripped, $rsuffixes[][2]$ be the suffixes of $n \times 2$ array for rule-based approach, and $asuffixes$ be a list of suffixes for aggressive approach) and one reference address (call-by-reference) that contains possible list of $stems$. The base criteria of the recursive function is stated at line 1. Then a simple iteration starts for each suffix-pair (refer to line 3). If the $word$ contains a suffix $suffix[i][0]$ at the end (see line 4), algorithm strip $suffix[i][1]$ from the word to make a stem (line no. 5). However, if the length of the stem becomes less than 2 (as line no. 6), or the last character is 'hosonto' (refer to line 8) then the stem is not acceptable, and starts processing for the next suffix-pair. Otherwise, the stem is added to the stems vector (line no. 10) as a candidate stems and the function calls Recursive Suffix Stripping **Algorithm 1** for aggressive suffixes that start recursion to find stem (see the line no. 11).

F. Proposed Stemming Algorithm

We have already discussed all of the components of our stemmer: the conservative approach, the aggressive approach, and the rule-based approach. Now, we put all the components together to extract a stem from the word. As an aggregation, we maintain three separate suffix-sets, $conservative\_suffixes[]$, $aggressive\_suffixes[]$ and $rule\_suffixes[][2]$ and $roots$ as a vector repository of some selected roots.

Algorithm 3: getStem ($word$): An algorithm to find stem of a given word

```
Input: word:to be stripped; conservative_suffixes[]:
       suffixes usually associated with verb and
       dictionary words; aggressive_suffixes[]:
       suffixes that can be associated with any words;
       rule_suffixes[][2]: suffixes that have
       distinguished identification part and strip part;
       stems: vector of a stem(s); stem: a relative root
       of a word; roots: vector of selected roots
Output: Stem of a given word
1  rsStripping (word, conservative_suffixes[], stems);
2  stem=conservativeStemming(stems,roots);
3  if (stem!=null) then
4  |  return stem;
5  ruleBaedStripping(word, rule_suffixes[][2],
     aggressive_suffixes[], stems)
6  stem=aggressiveStemming(stems)
7  if (stem!=null) then
8  |  return stem;
9  rsStripping (word, aggressive_suffixes[], stems);
10 stem=aggressiveStemming(stem);
11 if (stem!=null) then
12 |  return stem;
13 else
14 |  return word;
```

According to the Recursive Suffix Stripping **Algorithm 1**, line no. 1 of the **Algorithm 3** produces a vector of stems. The function $conservativeStemming()$ at line 2 checks availability of every $stem \in stems$ in the $roots$. If it finds any match, it returns the smallest stem, otherwise returns null. Line 3 states the fact straight forward. It is like greedy method, out of the three approaches, if any earlier approach finds some solution, the rest is discarded (as line no. 3, 7, and 11). $ruleBasedStripping()$ function also generates a vector of stems (line no. 5). The role of the $aggressiveStemming()$ is nothing but finding the smallest stem in $stems$ blindly. Therefore, the smallest stem is extracted in line 6. However, if the $stems$ is empty, then stem must be empty or null, which means that the $ruleBasedStripping()$ is unsuccessful in finding any stem. According to line 9, the $rsStripping()$ function is applied to generate a list of candidate $stems$, then the $aggressiveStemming()$ produces the smallest stem unless vector $stems$ is not empty. If all the steps above fail, the algorithm returns the original word as $stem$ as stated in line 14.

IV. Experiments and Evaluation

To evaluate the effectiveness of our proposed Bengali stemmer, we create a larger dataset by crawling news article from a popular Bengali daily news paper, "The Daily Protho Alo" [1]. The dataset contains almost 0.78 million words with 62 thousand distinct words. Our stemmer reduced this 62 thousand words into 32 thousand stems.

TABLE I. Basic Statistics about Dataset

Topic	Quantity
Number of documents	3772
Number of total sentences	64972
Number of distinct sentences	58058
Number of total words	779064
Number of distinct words	62367
Number of extracted stems	32074

In Information Access (IA) research, a document is considered as a vector of distinct words. Therefore, our proposed stemmer certainly reduces the vector dimensions by an almost half, which may increase the performance twice faster. Moreover, among the 31074 extracted stems 29479 stems are closely accurate, which means that almost 92% extracted stems are accurate with a minimal root repository. This experimental result shows its strength and efficiency against other available systems.

V. Conclusion and Future Direction

In this paper, we proposed an efficient and effective method for Bengali stemming based on three different approaches (conservative, aggressive, and rule-based) during stemming. Firstly, conservative stemming addresses mainly verbs and exceptional words heavily affected by stemmer. In this technique, 459 verb roots and 28 exceptional word roots are collected to form a small repository. The conservative stemming approach only considers the stems available in the repository. As the name suggest, the fundamental criteria of the conservative approach are that it does not allow stems beyond the predefined root-set. In the aggressive stemming approach, suffixes are eliminated repeatedly until the last suffix being removed. As the name suggests it finds the smallest length stems through eliminating the largest suffix from the word aggressively. Other than these two techniques, rule-based approach focuses on some right most suffixes, where stripping suffixes are different than the identified patterns. Rule-based approaches is a bit modified aggressive approach. With the essence of these three techniques, our novel stemming algorithm becomes unique and efficient for Bangla. We have experimented against a relatively larger dataset containing 0.78 million words in a 3772 real news documents dataset. The dataset contains heterogeneous words, of which many have been imported from different languages. In spite of the heterogeneous nature of our dataset, we achieved 92% of accuracy. However, because of the aggressive approach, we have observed several over stemming, which can be our future direction. Moreover, we will also focus on more purified suffixes and enlarged root repository containing most of the exceptional words.

[1] http://www.prothom-alo.com

Appendix

Our stemmer can be downloaded from our laboratory page http://skeim.org/projects-download/. It is a runnable *.jar. Anyone can utilize this software not only his Information Access (IA) research project, but also make use of it as an application. The short description on operation has been depicted in Table II. The basic command reflects:

$$java\ -jar\ jSkeimStemmer.jar\ <parameter>\ \ <data>$$

The basic operations has been articulated in Table II.

References

[1] J. Lovins, "Development of a stemming algorithm," *Translation and Computational Linguistcs*, vol. 11, no. 1, pp. 22–31, 1968.

[2] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[3] D. Hull *et al.*, "Stemming algorithms: A case study for detailed evaluation," *JASIS*, vol. 47, no. 1, pp. 70–84, 1996.

[4] W. Kraaij and R. Pohlmann, "Viewing stemming as recall enhancement," in *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 1996, pp. 40–48.

[5] M. Adriani, J. Asian, B. Nazief, S. Tahaghoghi, and H. Williams, "Stemming indonesian: A confix-stripping approach," *ACM Transactions on Asian Language Information Processing (TALIP)*, vol. 6, no. 4, pp. 1–33, 2007.

[6] V. Gupta and G. Lehal, "Punjabi language stemmer for nouns and proper names," in *proceedings of the 2nd Workshop on South and Southeast Asian Natural Language Processing (WSSANLP) IJCNLP*, 2011, pp. 35–39.

[7] S. Tai, C. Ong, and N. Abullah, "On designing an automated malaysian stemmer for the malay language (poster session)," in *Proceedings of the fifth international workshop on on Information retrieval with Asian languages*. ACM, 2000, pp. 207–208.

[8] V. Gupta and G. Lehal, "A survey of common stemming techniques and existing stemmers for indian languages," *Journal of Emerging Technologies in Web Intelligence*, vol. 5, no. 2, pp. 157–161, 2013.

[9] A. Ramanathan and D. Rao, "A lightweight stemmer for hindi," in *Proceedings of Workshop on Computational Linguistics for South-Asian Languages, EACL*, 2003.

[10] M. Greengrass, A. Robertson, S. Robyn, and P. Willett, "Processing morphological variants in searches of latin text," *Information research news*, vol. 6, no. 4, pp. 2–5, 1996.

[11] S. Bressan, "Indexing the indonesian web: Language identification and miscellaneous issues," *Recall*, vol. 80, p. 100, 2000.

[12] J. Carlberger, H. Dalianis, M. Hassel, and O. Knutsson, "Improving precision in information retrieval for swedish using stemming," in *Proceedings of NODALIDA 13th Nordic Conference on Computational Linguistics*, 2001, pp. 21–22.

TABLE II. The Basic Operation Manual

Parameter	Data	Output	Comments
-w	nirapottahInotai	nirapotta	Only the final stem has been returned as output.
-wa	nirapottahInotai	The first [ ] contains candidates of conservative stemming, if appropriate stem is not found. Second [ ] shows the list of candidate stems from aggressive approach.	This can be used for analysis purpose.
-f	name of the file which contains Bangla text	The first line contains the original text, whereas the second line represent the one by one corresponding stemmed text.	The final stem of each word has been returned sequentially as output.
-fa	name of the file which contains Bangla text	Full analysis of each word available in the file as described in row 2.	This can be used for analysis purpose.

[13] C. Monz and M. De Rijke, "Shallow morphological analysis in monolingual information retrieval for dutch, german, and italian," in *Evaluation of Cross-Language Information Retrieval Systems*. Springer, 2001, pp. 262–277.

[14] I. Moulinier, J. A. McCulloh, and E. Lund, "West group at clef 2000: Non-english monolingual retrieval," in *Cross-Language Information Retrieval and Evaluation*. Springer, 2000, pp. 253–260.

[15] M. Popovic and P. Willett, "The effectiveness of stemming for natural-language access to slovene textual data," *Journal of the American Society for Information Science*, vol. 43, no. 5, p. 384, 1992.

[16] F. Ekmekcioglu, M. Lynch, and P. Willett, "Stemming and n-gram matching for term conflation in turkish texts," *Information Research News*, vol. 7, no. 1, pp. 2–6, 1996.

[17] S. Dasgupta and V. Ng, "Unsupervised morphological parsing of bengali," *Language Resources and Evaluation*, vol. 40, no. 3-4, pp. 311–330, 2006.

[18] M. Islam, M. Uddin, and M. Khan, "A light weight stemmer for bengali and its use in spelling checker," in *Proceedings of the 1st International Conference on Digital Communication and Computer Applications (DCCA07), Irbid, Jordan*. BRAC University, 2007.

[19] P. Majumder, M. Mitra, S. Parui, G. Kole, P. Mitra, and K. Datta, "Yass: Yet another suffix stripper," *ACM transactions on information systems (TOIS)*, vol. 25, no. 4, p. 18, 2007.