Error-tolerant Finite-state Recognizer and String Pattern Similarity Based Spelling-Checker for Bangla

Md. Munshi Abdullah, Md. Zahurul Islam, Mumit Khan
BRAC University, Dhaka, Bangladesh
asad.anto@gmail.com, zahurul@bracu.ac.bd, mumit@bracu.ac.bd

Abstract

A crucial figure of merit for a spelling checker is not just whether it can detect misspelled words, but also in how it ranks the suggestions for the word. Spelling checker algorithms using edit-distance methods tend to produce a large number of possibilities for misspelled words. We propose an alternative approach to checking the spelling of Bangla text that uses a finite state automaton (FSA) to probabilistically create the suggestion list for a misspelled word. FSA has proven to be an effective method for problems requiring probabilistic solution and high error tolerance. We start by using a finite state representation for all the words in the Bangla dictionary; the algorithm then uses the state tables to test a string, and in case of an erroneous string, try find all possible solutions by attempting singular and multi-step transitions to consume one or more characters and using the subsequent characters as look-ahead; and finally, we use backtracking to add each possible solution to the suggestion list. The use of finite state representation for the word implies hat the algorithm is much more efficient in the case of non-inflected forms; in case of nouns, it is even more significant as Bangla nouns are heavily used in the non-inflected form. In terms of error detection and correction, the algorithm uses the statistics of Bangla error pattern and thus produces a small number of significant suggestions. One notable limitation is the inability to handle transposition errors as a single edit distance errors. This is not as significant as it may seem since the number of transposition errors are not as common as other errors in Bangla. This paper presents the structure and the algorithm to implement a practical Bangla spell-checker, and discusses the results obtained from the prototype implementation.

1. Introduction

Bangla is a complex language with a large lexicon, and it is primarily a close-phonetic oriented language. A morphologically-based Bangla spelling checker using string pattern comparison often produces huge number of possible solutions due to the use of compound characters in Bangla. So, phonetic based spell checking is the better approach for Bangla. Puspa [1] is one such phonetic spelling checker for Bangla, which, like any phonetic based spell checkers, has its limitations. It often fails to suggest for multiple errors in both the root and the suffix of a word. It also produces huge number of suggestion for detectable multiple error situations. But, as it is necessary to have minimal dictionary size for a spell-checker, it is also important that it produces small number of relevant suggestions for a word with around 50% error. Error tolerant finite state recognition uses its underlying finite-state recognizer to relate an erroneous word with is most probable solution set. Our goal is to develop a system that can handle non-word errors in Bangla efficiently and quickly. In our proposed system, all the possibilities are addressed while checking a non-word error. The reason is that every character may be correct and at the same time may be wrong, and so the better approach is to consider all possibilities and put a threshold value to control the result size. The threshold refers to the minimum edit-distance. The primary state table generation ensures that only the words available in the dictionary can be produced and nothing else. The motivation behind the study is to come up with a strong spell-checking algorithm using the finite state ma-chine to produce better result from the existing spell-checkers. In this paper, we tried to show the improvements achieved by using Finite State approach and the suggested algorithm and listing the results yielded by our working prototype. Our strategy is to map all the roots and their inflected forms in a state table and use it with the algorithm to check different types of error firstly as a single character error, as an earlier study [2, 3] suggested that error length of 1 to 2 is about 74% of all Bangla non-word errors, using the current character and a look-ahead character. If it turns out to be a multi-character error, multiple transitions for both the current character and the look-ahead character will be used to check multiple deletion or deletion-insertion error. If a solution can not be found then the current states will be saved and all the previous characters are inserted, and a multi-transition to consume the current character from the start state will be attempted. The final attempt may lead to a distant solution but as it will have a large edit-distance; it may eventually be left out of the

solution list. On the other hand, saving the cur-rent state before the desperate approach actually gives the ability to even check 3 and more character error with the subsequent characters. This should cover the vast majority of the spelling errors as 1 to 3 character errors make up 90.8% of all Bangla non-word errors [2, 3].

2. Related Work

There has been some activity in designing spelling checkers for Bangla and other South Asian languages. One recent example is the work done by UzZaman and Khan on Bangla spelling checker that uses double-metaphone called Puspa speller [1]. An approximate string matching algorithms [3] and a direct dictionary look up method [4] have been used so far for the detection of typographical errors and cognitive phonetic errors. Other many work have done on this field. One significant work has done on Error-tolerant Finite State Recognition with Applications to Morphological Analysis and Spelling [5].

3. Error Tolerant Finite State Recognition and Mapping

We can define error-tolerant finite state recognizer as a high error tolerant recognizer of a specific string set supported by the underlying finite state representation of that specific string set [5]. The important thing is to generate the finite state machine to recognize only the specific string set (All the Entries of Bangla Dictionary) only and nothing else. The term error-tolerance in the other hand requires an error metric for measuring how much two string deviate from each other. Transposition is rather uncommon in Bangla text and thus Levenshtein distance (accessible at: http://en.wikipedia.org/wiki/Levenshtein_distance) is the measure of the deviation of two strings. For the demonstration purpose we will consider only the currently recognized set at Table 1 by our working prototype. Figure 1 explains the underlying finite state recognizer's principle by mapping some of the roots and its inflections. One thing should be noted is that only the existing strings are recognized and nothing else, even though there are other valid strings that are not in the list. The state table can produce by any of the commonly used finite state machine tools such as Xerox FST tool (accessible at: www.xrce.xerox.com/competencies/content-analysis/fssoft/docs/fst-97/xfst97.html) and AT&T FSM library (accessible at: http://www.research.att.com/~fsmtools/fsm/). The mapping can be extended for every transition representing a property, a feature not required for our implementation at present. For this particular case study, we have used only 10 root words and a handful of inflections, but it is enough to test the expected improvements made to the approach to be used as a non-word spell checker. The extensive mapping is not used but kept as a future extension to the spell checker to have the ability to work on real-word errors.

4. Spelling-Check and Errors

The spelling checker being developed deals with only non-word errors. The algorithm has been optimized on the basis of statistics to handle the most probable errors efficiently and yet leave the scope to handle the more unusual cases. The algorithm handles substitution and insertion errors somewhat in a similar manner. It uses the look-ahead character to map the next move. On the other hand, deletion error is handled through the current character. One advantage in checking spelling using this algorithm is that at every encounter of an erroneous character, it saves the current state and thus uses it with later a character which reduces the scope of misdetection in a multi-character error situation. At the same time it does not leave any possibility unexplored and often produces "absurd" results, but if we take a closer look, the absurd strings turn out to be probable solutions in terms of string pattern similarity. The list of types of Bangla error found in Table 3 and the rate of their occurrences are the main motivations behind the spell checking process [3]. The algorithm does not handle transposition error as a single edit distance error to reduce complexity. This is reasonable since the number of these errors is rather small.

Table 1: List of verb roots & some inflected form

কর	বল	ধর	খা	যা
করিয়েছিলেন	বলিয়েছিলেন	ধরিয়েছিলেন	খাইয়েছিলেন	
করেছিলেন	বলেছিলেন	ধরেছিলেন	খেয়েছিলেন	গিয়েছিলেন
করছিলেন	বলছিলেন	ধরছিলেন	খাচ্ছিলেন	যাচ্ছিলেন
করিয়েছেন	বলিয়েছেন	ধরিয়েছেন	খাইয়েছেন	গিয়েছেন
করেছিলে	বলেছিলে	ধরেছিলে	খাইয়েছিলে	গিয়েছিলে
করছিলে	বলছিলে	ধরছিলে	খাচ্ছিলে	যাচ্ছিলে
করছেন	বলছেন	ধরছেন	খাচ্ছেন	যাচ্ছেন
করেছেন	বলেছেন	ধরেছেন	খেয়েছেন	গিয়েছেন
করিয়েছিলে	বলিয়েছিলে	ধরিয়েছিলে	খাইয়েছিলে	গিয়েছিলে
করেছিল	বলেছিল	ধরেছিল	খাচ্ছিল	যাচ্ছিল
করিয়েছিল	বলিয়েছিল	ধরিয়েছিল	খাইয়েছিল	গিয়েছিল
করেছে	বলেছে	ধরেছে	খেয়েছে	গেছে
করছে	বলছে	ধরছে	খাচ্ছে	যাচ্ছে
করছ	বলছ	ধরছ	খাচ্ছ	যাচ্ছ
করল	বলল	ধরল	খেল	গেল
করিয়েছে	বলিয়েছে	ধরিয়েছে	খাইয়েছে	গিয়েছে

Table 2: List of Noun roots & some inflected form

মানুষ	গরু	ছেলে	মেয়ে	টেবিল
মানুষরা		ছেলেরা	মেয়েরা	
মানুষেরা				
মানুষগুলা	গরুগুলা	ছেলেগুলা	মেয়েগুলা	টেবিলগুলা
মানুষগুলি	গরুগুলি	ছেলেগুলি	মেয়েগুলি	টেবিলগুলি
মানুষগুলো	গরুগুলো	ছেলেগুলো	মেয়েগুলো	টেবিলগুলো
মানুষদের		ছেলেদের	মেয়েদের	

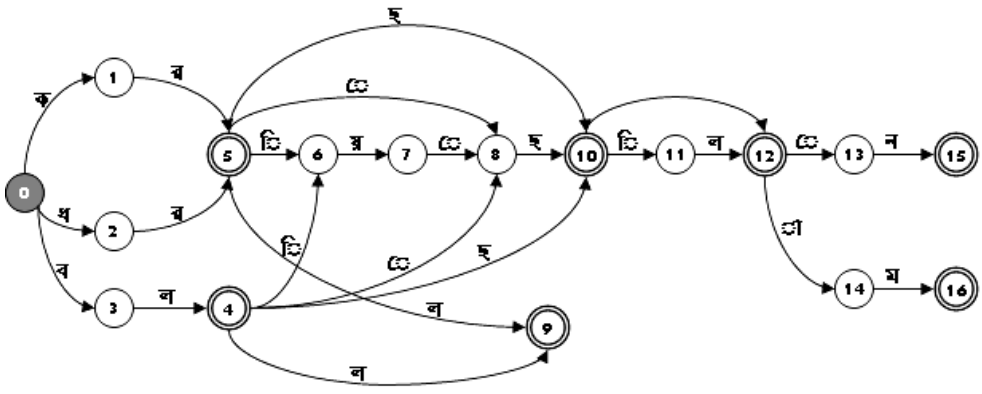

Figure1: Finite State Recognizer to Recognize Some Strings from Table 1

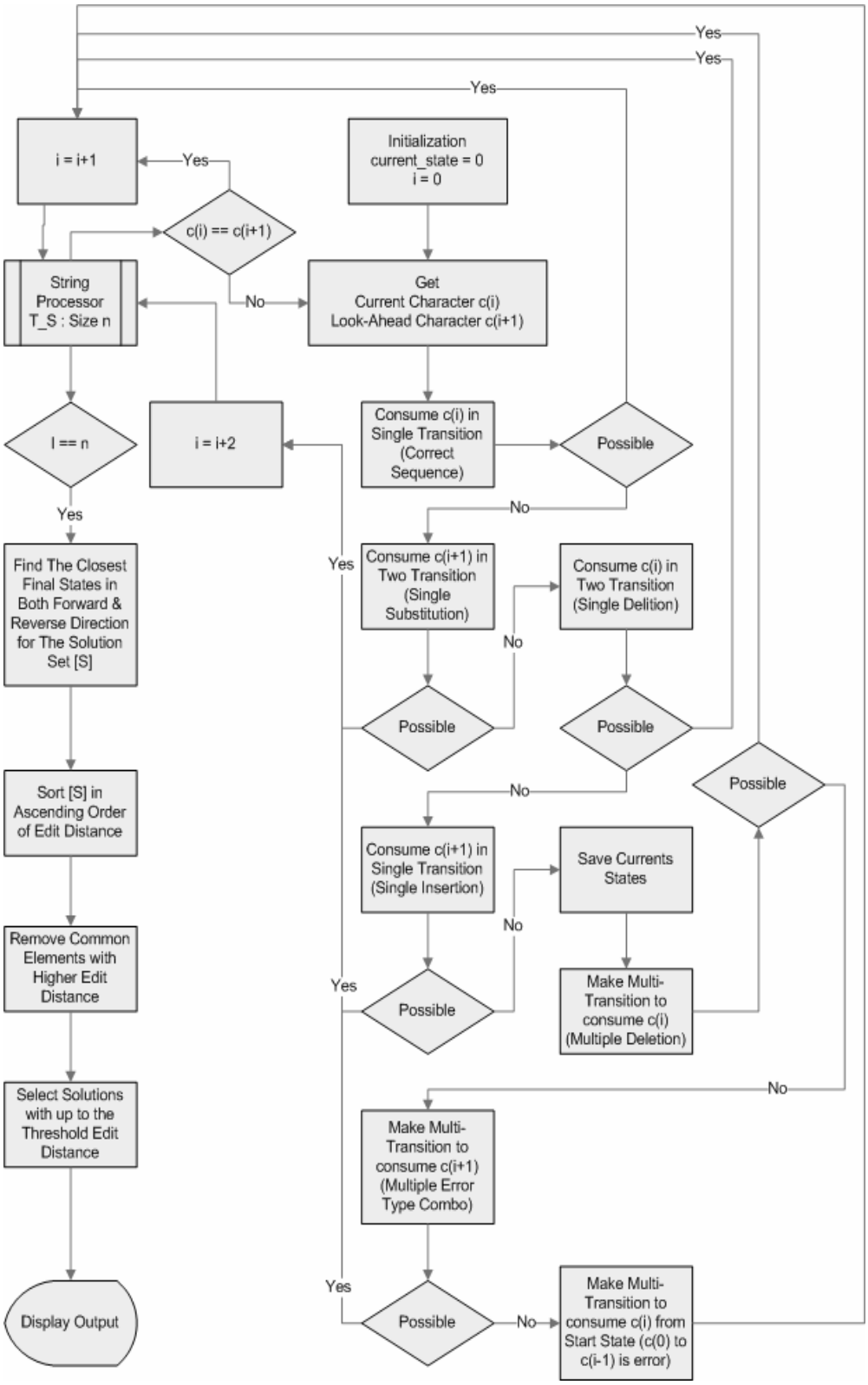

Figure 2: Spelling-Checker Algorithm Flow Diagram

Table 3: Types of non-word error in Bangla

Type of error	Percentage
Substitution	66.90
Deletion	17.87
Insertion	9.60
Transposition	5.63

5. Algorithm and Prototype Implementation

Figure 2 explains the working principles of the underlying algorithm. While the state tables are meant to be automatically generated from a set of finite state rules, the tables used in the prototype implementation have been hand-coded. The algorithm gets the input from the String Scanner, and then uses the Finite-State Recognizer Table to update the set of possible solutions. As the algorithm uses all the possibilities to reach the solution, sometimes our algorithm produces the same string pattern with different edit distance. The finalizing selection process addresses this problem by selecting the final solution set.

The algorithm works on the basis of error occurrence statistics. As suggested by [3], single character error is the highest among all Bangla text errors (41.36%), and so it has been handled exclusively. Moreover, the expectation of the presence of a particular type of errors in Table 3 also taken under consideration. After the initialization process, we checked if each character occurrence can be consumed by a single transition. If so then we are scanning a correct word. We have a basic check in the main block for repetition by putting an error value in our State-Table for every state. At each failure of such the sequence to assume possible solution is as follows:

We check for Substitution Error (possible expectancy 66.90%) by trying to consume the look-ahead character in two transitions. If possible then we switch our next test character to be the immediate next character of the look-ahead character. The recorded edit distance of this move is 1.

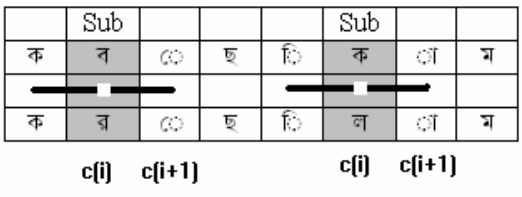

Figure 3: Substitution Error Handling.

a. If we failed in section (a), we check Deletion Error (possible expectancy 17.87%) by trying to consume the current character by two transitions. If possible then we switch our next test character to be the look-ahead character. The recorded edit distance for this move is 1.

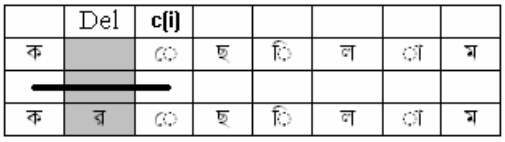

Figure 4: Deletion Error Handling.

b. If we failed in section (b), we check Insertion Error (possible expectancy 9.60%) by trying to consume the look-ahead character by a single transition. If possible then we switch our next test character to be the immediate next character to the look-ahead character. The recorded edit distance for this move is 1.

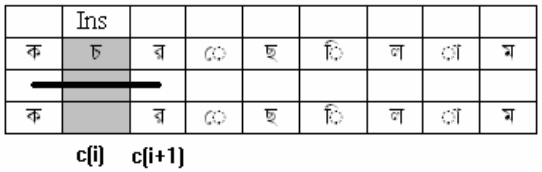

Figure 5: Insertion Error Handling.

c. If we failed in section (c), we first save the current state for later use and check Multiple Character Deletion Error by trying to consume the current character by a multiple transitions. If possible then we switch our next test character to be the look-ahead character. The recorded edit distance for this move is the sum of the number of transitions and 1 and it is updated to the saved state as well.

d. If we failed in section (d), then we assume a combinational multiple errors and try to consume the look-ahead character in multiple transitions. If possible then we switch our next test character to be the immediate next character to the look-ahead character. The recorded edit distance for this move is the sum of the number of transitions and 1.

e. If all attempts are failed we make a bold assumption that all the characters before the current character are insertions and make a multi-transition from the start state to consume the current character and the try the next character. The recorded edit distance for this move is the

sum of the number of transitions and the position of current character.

It should be noted that we save the current state just after failing with single error checking. This will eventually help us to lead to the correct solution as multiple moves with the next test character will be made from that state. All these checks will be made with all the elements of the solution set thus no possibilities will be left off and a synchronous edit distance update is made to keep things in order. Another significant benefit from this approach is the reduced number of pattern matching. The algorithm will make deterministic moves rather than trying blind guesses, which this reduces the processing time needed for each check and increases the speed.

After checking the whole string a finalizing step is used in which the possible output set is checked, sorted and selected by threshold matching. All the possibilities are finalized by making them reach a final state by the minimum number of moves and it is tried in both forward and reverse direction. Due to the mix of singular and multiple moves the prototype often produces same patterns with different edit distance. Similar patterns with higher edit distance are removed after sorting the list. Then the elements having an edit distance higher than a predefined threshold value are truncated. Thus the result is shown.

6. Performances and Evaluation

The evaluation metric used for testing the performance is based on the [6] parameters for isolated-word error correction:

- Lexicon size;
- test set size;
- correction accuracy for single error misspellings;
- correction accuracy for multi-error misspellings; and
- type of errors handled (phonetic, typographical, OCR generated etc.).

These metrics have been used by another notable effort in Bangla spelling checker as well [7]. There were 3 files in the test set, each with 97 words. One file contained all correct strings, another contained strings with single error of any of the possible types, and the third file contained string with multiple, ranging from 2 to 9, character errors.

The lexicon used in this study does not contain the actual words, rather consists of a state table with 50 states for the 97 strings used in the test case. The results show 92% accuracy for single character error in different positions. The algorithm fails to find the suggestion in cases where the transposition error occurs at either end of the word, and that accounts for much of the remaining 8% in our results. In case of multiple character errors, the accuracy is approximately 70%. It should be noted that some of the input words were randomly generated nonsense words, which often led to the algorithm producing absurd suggestions in return. An attempt was made to check the suggestion generation capabilities of the algorithm for any word, including those that may never occur in Bangla.

One observation is the important result is the very small incremental time needed to find single-error misspellings over the correct ones. The state transition approach keeps the transition time somewhat the same and that should be regarded as a significant improvement in the performance of the spell checker.

7. Future Work

One notable improvement would be to introduce transposition errors as a single edit distance errors. If that is done, a simple check of single transition possibility based on the current character and the look-ahead character will yield a possible suggestion. Faster and optimized state table generation can further improve the performance. Extending this algorithm to handle real-word errors can be achieved by augmenting the information kept for the mapping of the states, which is currently used for state status determination only. Memory requirement is a concern in the current implementation, which can be significantly improved by using different representations such as node-edge representation for the state table.

8. Conclusion

This paper presents a finite state machine based spelling checker for Bangla that shows good promise in terms of suggestion generation and runtime performance. The method described optimizes its performance by using statistical information on the error patterns of Bangla text. While it is quite possible to use this method on other languages without such statistics, the performance will definitely suffer for the lack of it.

9. Acknowledgement

This work has been supported in part by the PAN Localization Project (www.panl10n.net) grant from the International Development Research Corporation,

Ottawa, Canada, administered through the National University of Computer and Emerging Sciences, Pakistan.

10. References

[1] N. UzZaman and M. Khan, "A Comprehensive Bangla Spelling Checker" In the *Proceeding of International Conference on Computer Processing on Bangla (ICCPB-2006)*, Dhaka, Bangladesh, 17 February, 2006.

[2] P. Kundu and B.B. Chaudhuri, "Error Pattern in Bangla Text", *International Journal of Dravidian Linguistics*, 28(2), 1999.

[3] B. B. Chaudhuri. 2001, "Reversed word dictionary and phonetically similar word grouping based spell-checker to Bangla text", *LESAL Workshop*, Mumbai, 2001.

[4] A.B.A. Abdullah and A. Rahman, "A Different Approach in Spell Checking for South Asian Languages", In the *Proceeding 2nd International Conference on Information Technology for Applications (ICITA)*, China, 2004.

[5] K. Oflazer, 1996, "Error-tolerant Finite-state Recognition with applications to Morphological Analysis and Spelling correction", *Computational Linguistics*, Volume 22, 1996, pp. 73 – 89.

[6] K. Kukich, "Techniques for automatically correcting words in text", *ACM Computing Surveys*, 24(4), 1992, pp. 377-439.

[7] A. Bhatt, M. Choudhury, S. Sarkar and A. Basu. 2005, "Exploring the Limits of Spellcheckers: A comparative Study in Bangla and English", *The Second Symposium on Indian Morphology, Phonology and Language Engineering (SIMPLE'05)*, Published by CIIL Mysore, Kharagpur, INDIA, 2005, pp. 60-65.

[8] N. UzZaman and M. Khan, "A Double Metaphone Encoding for Bangla and its Application in Spelling Checker", *IEEE International Conference on Natural Language Processing and Knowledge Engineering*, Wuhan, China, 2005.

[9] N. UzZaman and M. Khan, "A Bangla Phonetic Encoding for Better Spelling Suggestions", In the *Proceeding of 7th International Conference on Computer and Information Technology (ICCIT 2004)*, Dhaka, Bangladesh, December, 2004.

[10] S. Deorowicz and M.G. Ciura, "Correcting Spelling Errors by Modeling their Causes", *International Journal of Applied Mathematics and Computer Science*, Vol. 15, No. 2, 2005, pp. 275–285.