Pre-calculated Spelling Correction Algorithm (PSC)

[1]Liny Varghese, [2]Supriya M.H, [3]K Poulose Jacob
[1,2,3]Cochin University of Science and Technology, Kochi, Kerala, India

Abstract
Spell correction algorithms provide a way to correct misspelled words. This paper proposes a modified symmetric delete spell correction algorithm which uses delete and transposes algorithm with pre-calculated dictionary.

Keywords
Spell correction, Levenshtein distance, Symmetric Delete Algorithm, Pre-calculated Dictionary

I. Introduction

Spell checking algorithms are used to correct the spellings of a misspelled word in applications like search engines, word processing, fraud detection etc. In spell Checking Algorithms, every term is checked against a dictionary and if the term is not found in the dictionary, then most similar terms to that word from dictionary are shown as spelling suggestions. This study proposes a new algorithm for spelling correction.

II. Literature Review

From the literature reviewed, it is found that three algorithms are mostly used in spell correction and they are:

1. Damerau-Levenshtein distance [1-2] is the distance between two strings by counting the minimum number of operations needed to transform one string into the other, where an operation is defined as an insertion, deletion, or substitution of a single character, or a transposition of two adjacent characters. Smaller the distance, similar the words are.
2. Peter Norvig Algorithm [3] generates all the possible combinations of the input term with an edit distance <=2 and then search each term with the dictionary. It is better than the first method, but still expensive and language dependent.
3. Symmetric Delete Algorithm [4] generates terms with an edit distance <=2 (deletes only operation) from each dictionary term and add them together with the original term to the dictionary. This has to be done only once during a pre-calculation step. Then it generate terms with an edit distance <=2 (deletes only operation) from the input term and search them in the dictionary. This algorithm claims that it is 1000 times faster than the second algorithm. Here the algorithm is language independent.

The Peter Norvig Algorithm is generating all the possible combinations of query term dynamically, i.e at the time of spell correction and then it needs to search each generated word against the dictionary. The time complexity of this algorithm depends on the number of characters in the query term and the alphabet. There will be 2n+2an+a-1 generated words where 'n' is the word size and 'a' is the alphabet size, with edit distance d=1. For eg: if word size=8 and alphabet size is 30 and edit distance is 1 then there will be 525 words in the search term list.

The Symmetric Delete Algorithm considers only delete operation in edit distance algorithm. Other operations like insert, transpose and substitute are skipped. The algorithm generates edit distance<=2 words of both dictionary and query string. Although generating edit-distance words of dictionary is a pre-calculation step, generating edit distance words of query string is a run-time process as in the case of algorithm(2). During the search time, the number of words generated is nCd where n=word size and d=edit distance. For eg: if word size=8 and alphabet size is 30(alphabet size is not important since only delete is used) and edit distance is 1 then there will be 8 new words in the search term list.

III. Proposed Algorithm

The proposed algorithm "Pre-calculated Spelling Correction algorithm (PSC)" uses only the pre-calculated dictionary with terms of edit distance <=2. Delete and transpose operations are considered for calculating the edit distance. Two hash tables are used here to store original dictionary words and newly generated words (with edit distance between 1 and 2). In hash Table 1, each row entry records the original word and its frequency, which is computed using a language model. The words with edit distance <=2 and links to the original words are stored in the hash Table 2 (See Table 1 and Table 2).

Table 1: Main Dictionary

Original word	Frequency
Term 1	f1
Term 2	f2
…	..
Term N	fn

Table 2: Generated words dictionary

Generated Words	Original words		
New Term 1	Term 1	Term 4	Term 6
New Term 2	Term 11	Term 51	Term 16
…	….	…	…
New Term M	Term 13	Term 42	Term 63

The query term is first checked against the main dictionary. If there are no matching words, continue the search to the generated words dictionary. If a record is found, then retrieve the correct word using the link to main dictionary. If more than one match is there, retrieve all matched words in main dictionary and output the word with highest frequency for spell correction.

The number of words in the search term list is only one; since no other words are generated for the search term as in the case of other two algorithms.

The main dictionary is same as in the case (2). But the generated word dictionary is much larger than Symmetric Delete Algorithm because transposition operation is also used along with delete operation in the new algorithm. For each original word in the main dictionary, with an edit distance=1 and a word length=n, there will be n words (due to deletions) and n-1 words (due to transpositions). i. e n+n-1=2n-1 new words in the new dictionary for each original word.

For example take a word 'exit'(n=4)

Deletions: xit, eit, ait, exi , Transpositions: eit,eixt,exit

Main dictionary: exit

Generated words dictionary: xit, eit, ait, exi, xeit, eixt, exti

IV. Pseudo Code

```
Prerequisite:
        Pre-calculate main dictionary and store it in hash table 1;
        Create hash table 2 with generated words
Code:
        Search keyword in Hash table 1
If found{
        Search term is correct, no spelling correction is required
}
else {
search in hash table2
If not found {
        No matching words in dictionary
}
Else{
        If number records matched =1
                Retrieve the original word from Hash table 2
        Else {
                Retrieve all original words;
                Output the word with highest frequency
        }
}
}
```

V. Computational Complexity

The time complexity of this algorithm is constant time i.e. O(1) due to the factor that the dictionary is stored on a hash table which has an average search time complexity of O(1). Hence it is also independent of the dictionary size (but depending on the average term length). If more memory is provided, collisions can be avoided - in the case of more than one match.

VI. Conclusion and Future Work

The methods used in algorithms used earlier are improved in the proposed algorithm to get better results. The major differences are in the dictionary size and the need to calculate new terms from search term. Both the dictionaries are pre-calculated and dictionary size increase is not a matter at this time. But the second aspect will reduce the search time of spell checker, since there is no need to generate the new words for search term at run time.

References

[1] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008.

[2] Damerau–Levenshtein distance. (n.d.). Retrieved January 25, 2017, [Online] Available: https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance

[3] How to Write a Spelling Corrector. (n.d.). Retrieved January 25, 2017, [Online] Available: http://norvig.com/spell-correct.html

[4] 1000x Faster Spelling Correction algorithm (n.d.). Retrieved January 25, 2017, [Online] Available: http://blog.faroo.com/2012/06/07/improved-edit-distance-based-spelling-correction/

[5] Fast approximate string matching with large edit distances in Big Data (n.d.). Retrieved January 25, 2017, [Online] Available: http://blog.faroo.com/2015/03/24/fast-approximate-string-matching-with-large-edit-distances/

Liny Varghese, working as Information Scientist at Cochin University of Science and Technology [CUSAT], Kerala since 2003. She did her M.Tech in Computer and Information Science from CUSAT in 2003. Her research interests include data mining, information retrieval, software configuration management and big data frameworks.

Dr. Supriya M.H. obtained her B.Tech from Regional Engineering College, Calicut. She completed her Master's and Ph.D. from Cochin University of Science and Technology, Kochi. She has more than 18 years of experience in teaching, 4 years in industry and is currently the Professor and Head of the Department of Electronics, CUSAT. With more than 130 publications and one patent to her credit, she has won several best paper awards and has contributed various chapters in well reputed books.

Dr. K. Poulose Jacob, Professor of Computer Science, Cochin University of Science & Technology; presently Pro VC, Cochin University of science and technology. Dr. Jacob has been teaching at the Cochin University since 1980. A National Merit Scholar all through, he is a graduate in Electrical Engineering and postgraduate in Digital Electronics. He obtained Ph D in Computer Engineering for his work in Multi-Microprocessor Applications. His other research interests are Information Systems Engineering, Intelligent Architectures and Networks. He has more than 50 research publications to his credit, and has presented research papers in several International Conferences in Europe, USA, UK and other countries. He is a Permanent Professional Member of the ACM and a Life Member of the Computer Society of India.