A Hybrid Methodology of Effective Text-Similarity Evaluation

Shu-Kai Yang(✉) and Chien Chou

National Chiao-Tung University,
No. 1001, Daxue Rd., Hsinchu 300, Taiwan (R.O.C.)
skyang@csie.nctu.edu.tw, cchou@mail.nctu.edu.tw

Abstract. In this paper, an effective methodology which hybridizes a LCS finding algorithm and SimHash computation is presented for evaluating the text-similarity of articles. It reduces the time-space scale needed by the LCS algorithm by breaking the articles into word subsequences of sentences, managing and pairing them by SimHash comparisons, and reaching the goal of evaluating long-length articles rapidly, with the similar parts and similarity score of compared articles figured out exactly.

Keywords: Text similarity · LCS · LSH · SimHash · Plagiarism detection

1 Introduction

The evaluation of text similarity is a classic problem in computer science. It is widely used in paraphrasing evaluation, plagiarism detection, version control, and the duplication checking of documents or web pages. The mainstream methodology of text-similarity evaluation is based on the longest common subsequence (LCS) finding algorithms [1]. If we treat an article as a sequence of words and punctuation marks, the problem of figuring out the similar parts of two articles is equal to finding the longest common subsequence of the two sequences. For example, here are two text sequences:

"that words is that words words that"
"is words that words that is"

The longest common subsequence (LCS) is

"is that words that"

In the six words of the second text sequence, there are four words in common with the first text sequence. We can say that the sequence has the 66% similarity with the first one. As shown above, exploiting a LCS finding algorithm in text-similarity not only evaluates the score but also figures out the similar parts of the text. But the disadvantage is, a LCS finding algorithm always takes squared complexity in both space and time. This makes the algorithm hard to evaluate long-length articles.

In this paper, presented an effective methodology that hybridizes Myers' LCS finding algorithm [2] and Similarity hash (SimHash) [3] computation, and develops the brand new framework to reduce the time-space scale needed by a LCS finding algorithm, and perform fast text-similarity evaluations to long-length articles. Similarity

C.-Y. Chang et al. (Eds.): ICS 2018, CCIS 1013, pp. 227–237, 2019.
https://doi.org/10.1007/978-981-13-9190-3_24

hashing (SimHash) is the way to compute the "fingerprints" of articles. It completely ignores the literal meaning of text data. It only emphasizes and gathers the statistics of the characteristics of bit patterns of the words in the text. With the SimHash computations and comparisons, it is easy to find similar articles in a text library, just like telling people by fingerprints. But SimHash cannot figure out the similarity score and similar parts of evaluated articles.

In the presented methodology, first parse the original and to-be-compared text, and get two sequences of words and punctuation marks. Then separate each sequence into the subsequences of sentences. Compute the SimHash of each subsequence, and find the most similar pairs between the subsequences of the original and to-be-compared text by checking the hamming distance of their SimHash values. Finally apply a LCS finding to the paired subsequences to figure out the similar parts and compute the similarity score of the whole text.

2 Previous Works

The most well-known algorithm used in text-similarity evaluation is Myers' algorithm and its variations [4]. Instead of iterative string comparisons, the algorithm transforms the subsequence finding problem into a path-finding problem on an edit graph as shown in Fig. 1. Considering the example text A and B to be evaluated, the goal of Myers' algorithm is to find the shortest path to change A to B.

A: *"that words is that words words that"*
B: *"is words that words that is"*

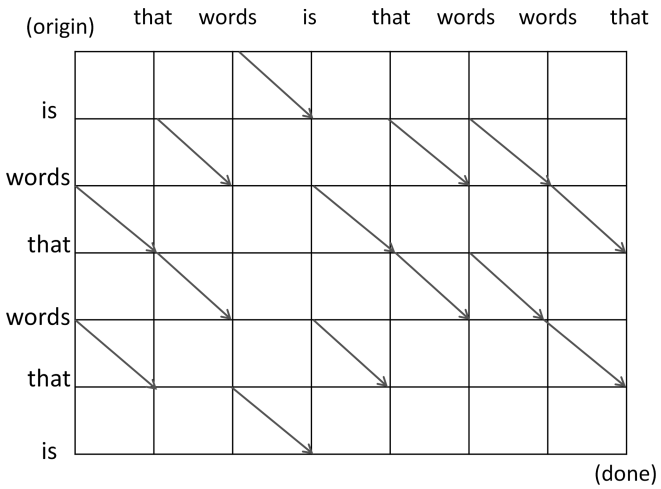


Fig. 1. The edit graph of the compared text

As shown in Fig. 1, the original text A is laid out as the horizontal dimension, the destination text B is laid out as the vertical dimension of the graph. In the graph, since the origin (0, 0) at the upper left corner, a horizontal step is equal to delete a word of A, a vertical step is equal to add a word of B. For example, the sub-path that moves from (0, 0) to (1, 1) go one step horizontally from (0, 0) to (1, 0), and one step vertically from (1, 0) to (1,1). The sub-path means that deleting "that" and adding "in" when changing text A to B.

Myers' algorithm also develops an optimized method for finding the shortest path since the origin to the lower right corner [5]. When the shortest path is found, the steps of changing A to B is known, the path is also called "edit script" from A to B, and the passed diagonal lines is exactly the "longest common subsequence (LCS)" of A and B.

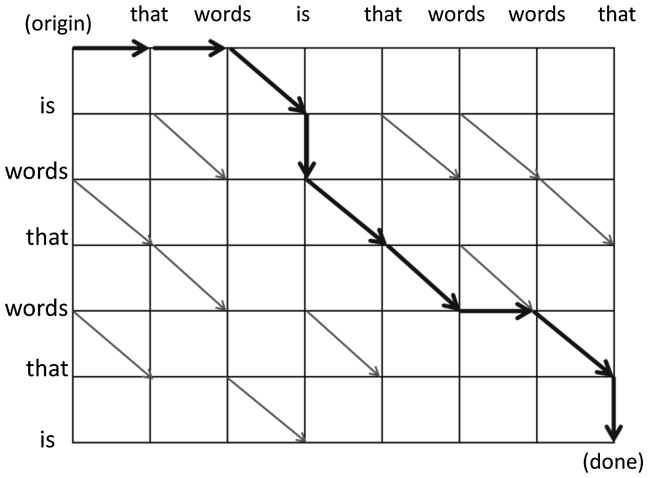


Fig. 2. The path found by Myers' algorithm.

Myers' path finding is basically a breadth-first-search (BFS). When growing the path of depth d to d + 1, the branches with the longer length are prior to the others. Passing the diagonal lines does not increase the depth but increases the length of current path. The shortest path found by Myers' algorithm is shown in Fig. 2, and we know the longest common subsequence (LCS) of A and B is:

"is that words that"

Although Myers' algorithm helps to skip some branches of searching, finding the shortest path is basically a squared complexity task in both space and time. If we are going to evaluate two articles of 10,000 words, we still have to prepare a grid graph with 100,000,000 cells, and lots of CPU time for the computation. Because a LCS finding algorithm like this detects the "subsequence" of text, plagiarism may evade the detection by swapping the order of sentences and paragraphs.

Similarity hashing (SimHash) is a counter-intuitive method of text-similarity evaluation [3]. Instead of traversing the text, it invents a rapid method compares the

composition of articles. The main idea of SimHash methodology is assuming the articles that talk about the same keywords with the same frequencies should be similar or equivalent articles, hence the SimHash computation tries to transform the words and their frequencies of occurrence to something like integer values or bit patterns for rapid comparisons.

Hash functions are the standard functions that map the inputted values to wide ranges evenly [6]. The mapped values always have the fixed length of data, even the length of the inputted data varies. For example, a typical hash function maps ASCII strings to 32-bit integers. A well-defined hash function has the characteristics:

1. A hash function maps a close value set to a wide range. A little change of the inputted value causes a very different hash value outputted by the hash function.
2. The hash values outputted by a hash function have the fixed data-length, such as 32 bits or 64 bits.
3. A hash function is not invertible. You cannot guess the inputted data according to its hash value.

Hash function are usually implemented as the shifting and XORing of the bits of the inputted byte stream. The original paper of SimHash has told us, you can apply any hash function that fits in with the described definitions when calculating the SimHash of an article. In the implementation of this paper, a 64-bit SDBM hash function is used.

For an article to calculate its SimHash, the hash values of every word in the article are calculated first. As shown in Fig. 3, we gather the statistics of each bit of the word hash values, and form the requested SimHash. For each *i*-th bit of the requested SimHash, it is 1 if all the *i*-th bits of word hash values have more 1 than 0, otherwise the *i*-th bit of SimHash is 0.

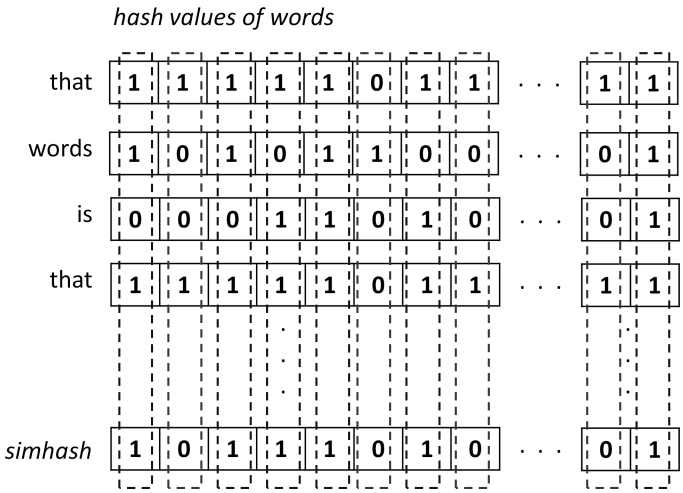


Fig. 3. A SimHash calculation example.

After the statistics shown in Fig. 3, each *i*-th bit of the result SimHash tells the characteristics of the article: how many words whose hash values have their *i*-th bits 1 or 0. If two compared articles have their SimHash values having more equal bits, the more similar those articles are. The SimHash calculation is one kind of local sensitive hashing (LSH), meaning that each bit of the hash value reflects a local change of the inputted source data. One 64-bit SimHash value remembers 64 characteristics of the article, and calculating the hamming distance (how many bits differ) of two SimHash tells the similarity of the two compared articles.

The SimHash methodology is very efficient to verify the similarity of two articles, so it is widely exploited in searching the plagiarism papers in the academic library, and in finding similar or duplicated web pages in a web-cache database. In the database, stored are the fixed-length SimHash values of articles, and we can rapidly find the plagiarized source by checking the hamming distance of the SimHash of the compared article and the SimHash values stored in the database without traversing the text.

Although the SimHash methodology is very efficient for long-length articles, the hamming distances neither tell the percentages of similarity nor figure out the similar parts of compared articles. In this paper, we hybridize the advantages of the LCS-based and SimHash-based algorithms, without their disadvantages.

3 The Hybrid Methodology

Considering the architecture of an article, sentences and paragraphs are swappable without changing the meaning of the original article, and the words inside a sentence are not swappable. The presented methodology therefore breaks the LCS-based text-similarity evaluation into three stages. In the first stage, the article is segmented into sentences. In the second stage, hash and pair the similar sentences by SimHash calculations. And in the third stage, LCS and edit scripts of the paired sentences are evaluated, and the similarity score of entire article can be calculated. The shortest edit script (SES) that scribes how to rewire the original text as the compared one is also acquired.

As shown in Fig. 4, for the two evaluated text A and B, they are parsed into sequences of words and punctuation marks, and the sequences are segmented in to subsequences, hashed, paired, and evaluated. Finally, the similarity of A and B is summed up by collecting the results of subsequences.

3.1 The Segmentation Stage

For further hashing and longest common subsequence (LCS) finding, the inputted text is parsed into sequences of words and punctuation marks. In the parsed punctuation marks, some are the separators of sentences, depending on languages. For example, English sentences are separated by periods, exclamatory marks, and question marks. And Chinese sentences are separated by comma and periods. After segmentation, one sequence of the inputted text is separated into subsequences of sentences.

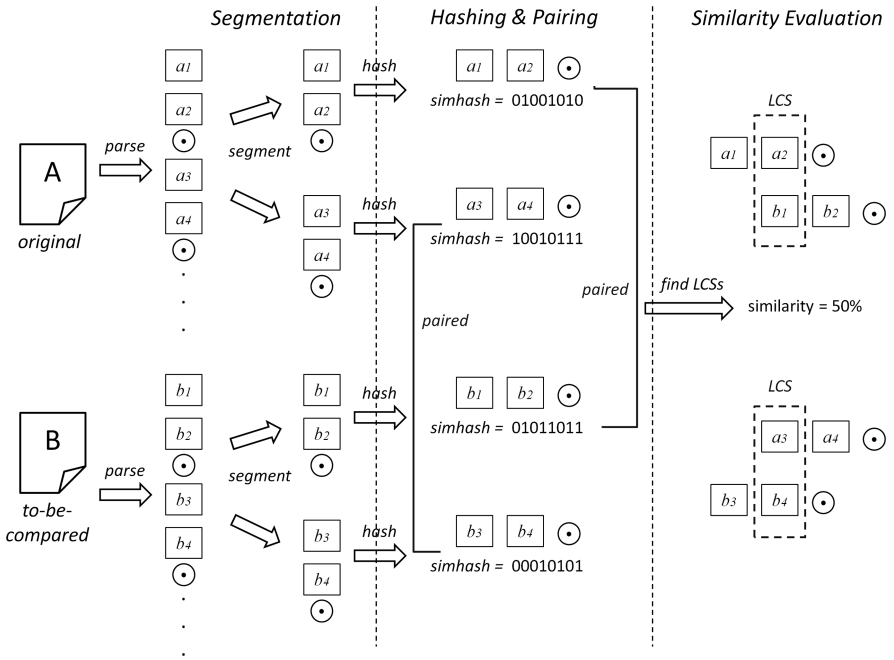


Fig. 4. Stages of the hybrid methodology.

3.2 The Hashing and Pairing Stage

For each word in the subsequences of sentences, compute their hash values by a 64-bit SDBM hash function, and form the similarity hash (SimHash) value of the sentence by the calculation introduced in [3]. As shown in Fig. 4, the hamming distances of the SimHash values of the sentences in A and B tells how similar there are. [3] has told us, if the hamming distance of two 32-bit SimHash values is greater than 3, they are not similar. In our implementation, if the hamming distance of two 64-bit SimHash values of a sentence in A and another sentence in B is greater than the threshold 12, they are determined completely different.

SimHash computation is a statisties-based methodology. Longer articles lead the results close to the expectations. [3] recommended the threshold value 3 for 32-bit SimHash values, so 6 can be a proper threshold for 64-bit SimHash values according to the bucketing principle. We take a doubled value 12 for handling short articles well.

For each sentence b in B, find the most similar sentence a in A by finding the minimal hamming distance to the SimHash values of b, and these two sentences a and b are paired for finding LCS in the next stage. If there is no such sentence a in A whose SimHash value has a hamming distance less than 12 to b, there is no similar sentence in A for the sentence b in B.

3.3 The Similarity Evaluation Stage

For every paired sentences *a* and *b*, find their longest common subsequence (LCS) by Myers' Algorithm [2]. Sum up the length of found LCS, and divide it by the total of words in the to-be-compared text B, we get the similarity of A and B. It is so intuitive that the score tells the percentage of B is plagiarized from A. The presented definition of similarity is

similarity (A, B) = sum of length of LCS/total words of B

where A is the original text and B is the to-be-compared paraphrase. The presented methodology has reduced the time-space complexity of LCS finding from the square of full-length of the text to the square of sentence length by breaking the task into three stages. It makes the methodology efficient enough to process long-length articles. It keeps the advantages of LCS finding that exactly finds the similar parts and the shortest edit Scripts (SES) of the compared text.

3.4 The Shortest Edit Script

A shortest edit script (SES) is a valuable by-product of Myers' Algorithm [2]. As shown in Fig. 2, the algorithm transforms the LCS finding problem to a shortest-path finding problem on an edit graph. Each horizontal or vertical step on an edit graph denotes a word deletion of adding, so that when the shortest path is found, the minimal steps to rewrite the original text A to the compared paraphrase B is also found. The minimal steps to rewrite the text are named the shortest edit scripts (SES) from A to B. See the mentioned example:

A: *"that words is that words words that"*
B: *"is words that words that is"*
LCS: *"is that words that"*

The shortest path on Fig. 2 is also the SES:

"~~that words~~ is +(words) that words ~~words~~ that +(is)"

In the presented methodology, it also collects the SES data of paired sentences, and tells the way to rewrite the original text to the compared one. The function is very useful in the paraphrasing practices of literature writing lessons.

3.5 Extending to Non-english Text

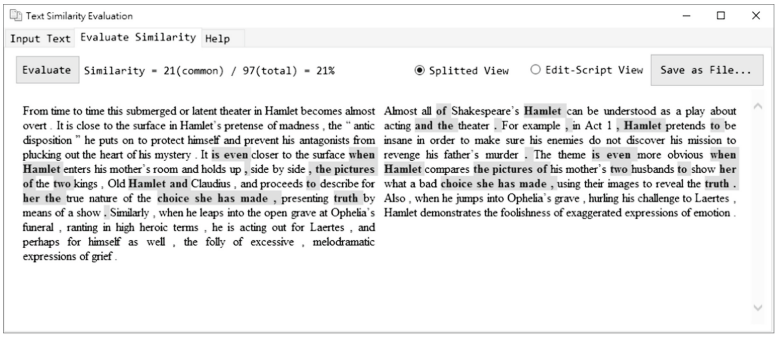
The presented methodology not only works well on English text, but also works on other languages. Here is the tip to make it applicable. One word in English is composed of multiple alphabets, but one word in another language, such as Chinese, is a single character, and a single character does not form a distinguished hash value in any hash function. For these cases, multiple words have to be grouped together for a hash value, and the rest parts of the presented methodology works fine. That is why we select 64-bit SDBM as the built-in hash function. It shifts the most bits of the inputted byte stream, and the fewest grouped characters are needed to form a distinguished hash value. In Chinese, it needs only four characters. This tip makes the presented methodology works well on Chinese, and Chinese-English mixed articles.

4 Implementation and Results

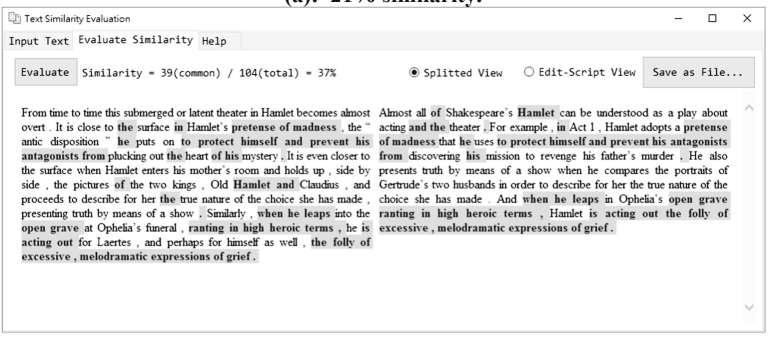
We have implemented the presented methodology as a.NET class library. It is applicable to both PC and web-based applications. The demonstrated is the results of the app processing the written text examples provided by Princeton University [7]. The original text is from the pages of an old book talking about Shakespeare. Here are some paraphrases detected different similarities. As shown in Fig. 5, the app shows the comparison sheets of the original text and the paraphrase, emphasizes the similar parts, and calculates the similarity score by counting the percentage of common words.

In Fig. 5(c) is the most similar paraphrase, and in Fig. 5(d) is the shortest edit script that rewrites the original text to the paraphrase. The app also saves to evaluation results as HTML pages with built-in JavaScript code. When you click on a similar sentence of the compared paraphrase, the window of original text scrolls to and highlights the sentence where the clicked sentence is plagiarized from.

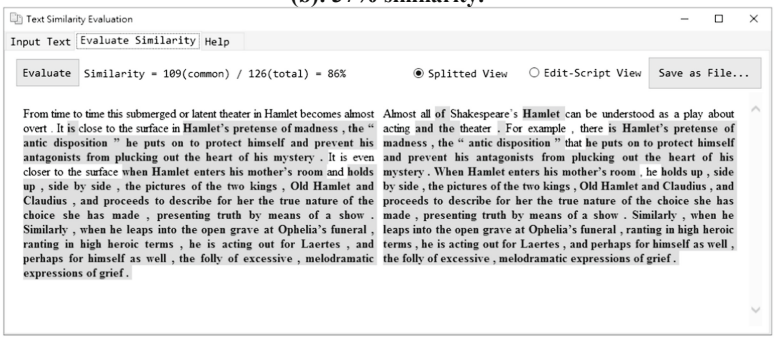
The demonstration in Fig. 6 shows that the presented methodology also works well on non-English text. The written text examples provided by National Taiwan University [8]. As shown in Fig. 6, inserting some redundancy words in sentences does not avoid the detection of text similarity.



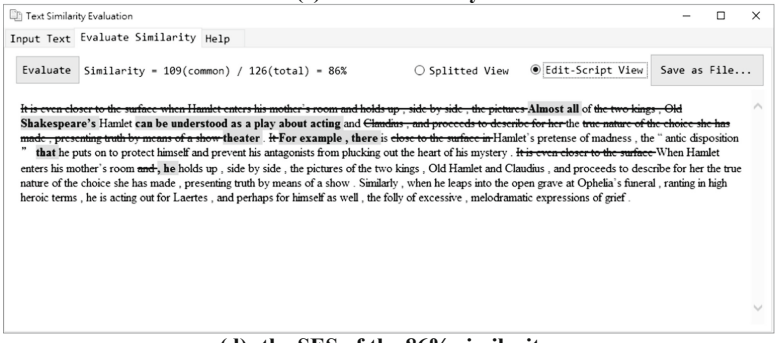
(a): 21% similarity.



(b): 37% similarity.



(c): 86% similarity



(d): the SES of the 86% similarity.

Fig. 5. The paraphrase detected 21%, 37%, and 86% similarity.

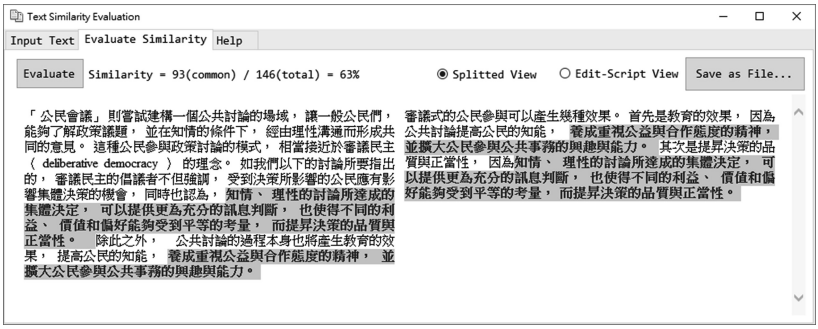


Fig. 6. The demonstration in Chinese.

5 Conclusion

We present a hybrid methodology of text similarity evaluation that keeps the advantages of LCS finding algorithm and SimHash calculation. It is effective for long-length text, and is keeping the rich information of detected similar parts, percentage of similarity, and the shortest path to rewrite the compared text. It is very applicable for the plagiarism detection, version control and duplication checking of documents, and for literature writing lessons. In the presented methodology, long-length articles are segmented into the subsequences of words and punctuation marks of sentences. The subsequences of the original text and to-be-compared text are paired by the minimal hamming distance of SimHash values. Then perform LCS finding in the subsequence pairs, the similar parts of paired sentences are found, and the percentage of text similarity is also acquired. The presented methodology has reduced the time-space complexity of LCS-based to the square of the sentence length. If works well on both English and non-English text. The cost of the methodology is so low that it can be used on PC or web applications, and it can be easily integrated in a text database for further applications.

References

1. Hunt, J.W., MacIlroy, M.D.: An algorithm for differential file comparison. Computing science technical report, #41, Bell Laboratories (1976)
2. Myers, E.W.: An O(ND) difference algorithm and its variations. Algorithmica **1**(1–4), 251–266 (1986)
3. Sadowski, C., Levin, G.: SimHash: Hash-based similarity detection. Technical report UCSC-SOE-11-07, University of California, Santa Cruz, February 2011
4. Indu, P., et al.: A comparative study of different longest common subsequence algorithms. Int. J. Recent Res. Aspects **3**(2), 65–69 (2016). ISSN 2349-7688

5. Hertel, M.: An O(ND) Difference Algorithm for C# (2006). https://www.mathertel.de/Diff/
6. Partow, A.: General Purpose Hash Function Algorithms. http://www.partow.net/programming/hashfunctions/
7. Examples of Plagiarism: Princeton University. https://pr.princeton.edu/pub/integrity/pages/plagiarism/
8. Lin, K.-M.: What is Plagiarism? Examples and Explanations. https://www.facebook.com/notes/657936507563326/