Plagiarism Detection Using the Levenshtein Distance and Smith-Waterman Algorithm

Zhan Su, Byung-Ryul Ahn, Ki-yol Eom, Min-koo Kang, Jin-Pyung Kim, Moon-Kyun Kim
Department of Artificial Intelligence, University of Sungkyunkwan
Cheoncheon dong, Jangan-gu, Suwon, Korea
shakira317@skku.edu, anbr0305@skku.edu, eomkiyol@empal.com, mini0220@hanmail.net, payon@hotmail.com, mhkim@ece.skku.ac.kr

Abstract

Plagiarism in texts is issues of increasing concern to the academic community. Now most common text plagiarism occurs by making a variety of minor alterations that include the insertion, deletion, or substitution of words. Such simple changes, however, require excessive string comparisons. In this paper, we present a hybrid plagiarism detection method. We investigate the use of a diagonal line, which is derived from Levenshtein distance, and simplified Smith-Waterman algorithm that is a classical tool in the identification and quantification of local similarities in biological sequences, with a view to the application in the plagiarism detection. Our approach avoids globally involved string comparisons and considers psychological factors, which can yield significant speed-up by experiment results. Based on the results, we indicate the practicality of such improvement using Levenshtein distance and Smith-Waterman algorithm and to illustrate the efficiency gains. In the future, it would be interesting to explore appropriate heuristics in the area of text comparison

1. Introduction

The term of plagiarism is defined [1] as "to take (ideas, writings, etc.) from (another) and pass them off as one's own." Detecting the presence of copied material in documents is a problem confronting many disciplines. For example, in education students may plagiarize, as may writers in academic journals; in the commercial world, copying is found in theft of copyright or intellectual property. Plagiarism is a serious problem in the academic world and prevention of it has very important significance.

In information theory and computer science, the Levenshtein distance is a string metric, which is one way to measure edit distance. The Levenshtein distance between two strings is given by the minimum number of operations, and that needed to transform one string into the other, where an operation is an insertion, deletion, or substitution of a single character. In this paper, we will use a diagonal line that derived from the Levenshtein distance in the plagiarism detection to decrease the scarcity of the dynamic programming array.

For implementation purposes, first we may represent the lexical entities by integers. For example, the text A girl, a girl, a smart girl with whom I want to marry, might be represented as the sequence < 1 2 1 2 1 3 2 4 5 6 7 8 9 >, with the correspondence between lexemes and integer tokens as shown in Table 1. Here we have chosen to discard punctuation symbols. Once the initial lexical analysis phase has been carried out on all submissions in a corpus, the comparison phase involves only the resulting symbol sequences or strings. Hence any of a wide variety of methods that can be used to detect similarities in two strings might potentially be used as an aid to collusion detection.

The Smith-Waterman algorithm, a method of biology has been used in the text plagiarism detection, is similar to Levenshtein distance, but the difference is that if the result of comparison is the same, the cost will add one. The issue of algorithm efficiency is of real practical significance here. If we wish to apply methods of string comparison effectively to a corpus of n documents, then we have little option but to make n (n – 1) / 2 pairwise document comparisons. So we will decrease the likely scarcity of the dynamic programming array and improve the algorithm efficiency.

The remainder of this paper is structured as follows: Section 2 is devoted to a description of use of a

The 3rd Intetnational Conference on Innovative Computing Information and Control (ICICIC'08)
978-0-7695-3161-8/08 $25.00 © 2008 IEEE

diagonal line in the text comparison, which is derived from the Levenshtein distance to improve the likely scarcity of the dynamic programming array. In Section 3, we indicate the simplified Smith-Waterman algorithm, and how to use it to detect the similarities. Section 4 presents the experiment results, and we summarize our results and conclusions in Section 5.

Table 1: A mapping of lexemes to tokens

a	1
girl	2
smart	3
with	4
whom	5
I	6
Want	7
to	8
marry	9

2. Levenshtein distance and the use of a diagonal line

A commonly-used bottom-up dynamic programming algorithm for computing the Levenshtein distance involves the use of an (n + 1) × (m + 1) matrix, where n and m are the lengths of the two strings. This algorithm is based on the Wagner-Fischer algorithm for edit distance. Here is pseudocode for a function Levenshtein distance that takes two strings, s of length m, and t of length n, and computes the Levenshtein distance between them:

```
int LevenshteinDistance (char s[1…m], char t[1…n])
   // d is a table with m+1 rows and n+1 columns
   declare int d[0…m, 0…n]
   for i from 0 to m
     d[i, 0] := i
   for j from 0 to n
     d[0, j] := j
   for i from 1 to m
     for j from 1 to n
       if s[i] = t[j] then cost := 0
             else cost := 1
       d[I, j] := minimum(
            d[i-1, j] +1, // deletion
       d[i, j-1] +1, //insertion
            d[i-1, j-1] +cost // substitution)
    return d[m, n]
```

Now we begin with an example, there are two text files below:

1. X = a b c x d e f g h i
2. Y = a b c d e f g h

Suppose that the rows of the dynamic programming array represent X and the columns represent Y, and suppose that the threshold for a significant match is 3. Figure 1 shows the resulting dynamic programming table.

	a	b	c	x	d	e	f	g	h	i
a	0	1	2	3	4	5	6	7	8	9
b	1	0	1	2	3	4	5	6	7	8
c	2	1	0	1	2	3	4	5	6	7
d	3	2	1	1	1	2	3	4	5	6
e	4	3	2	2	2	1	2	3	4	5
f	5	4	3	3	3	2	1	2	3	4
g	6	5	4	4	4	3	2	1	2	3
h	7	6	5	5	5	4	3	2	1	2

Figure 1: The dynamic programming array for strings X and Y

On applying the algorithm, we find that the return value 2 is <= 3, but by anglicizing the two text files, we can find the diagonal line portion has the higher similarities, which we flagged it in red.

Proof:

from the theory of the Levenshtein distance, we can get this rule:

Suppose the beginning of the comparing node is the same, if the next comparing node is also the same, the cost is 0, so all of these values make of a diagonal line. The rectangle this diagonal line divides represented the similarities. In order to decrease the computation, we divide the dynamic programming array into some grids according to the rectangles. Considering the psychological factors, a plagiarism text is impossible the same with two portion of one text, so we can cut off the table like Figure 2. That means the rows and columns concluding rectangles which is made of the diagonal line will not be computed.

	a	b	c	x	d	e	f	g	h	i
a		1	2							
b	1		1							
c	2	1								
d						2	3	4	5	6
e					2		2	3	4	5
f					3	2		2	3	4
g					4	3	2		2	3
h					5	4	3	2		2

978-0-7695-3161-8/08 $25.00 © 2008 IEEE

Figure2: Divide the table

All above means that when we compare some text files:

First, we should represent the lexical entities by integers, and initiate the table by comparing the integers. If they are the same, the node is red.

Second, connect the red nodes which can made up a diagonal line, mark the rectangles the diagonal lines belongs to, and compute the number of the nodes as similar degree.

Third, divide the table into grids and make the rows and columns concluding rectangles as portions that need not compute.

3. Simplified Smith-Waterman algorithm

The Smith-Waterman algorithm [3] is a classical method of comparing two strings with a view to identifying highly similar sections within them. It is widely-used in finding good near-matches, or so-called local alignments, within biological sequences. But now, Smith-Waterman algorithm has been used in the text plagiarism detection, and our paper will simplify it.

We define S_{ij} to be the parameter of the matrix, and after simplifying it, the recurrence relation for S_{ij} is

$$S_{ij} = \begin{cases} S_{i-1,j-1} + 1 & \text{if } X(i) = Y(j) \\ \max(0,\ S_{i-1,j} - 1,\ S_{i,j-1} - 1,\ S_{i-1,j-1} - 1) & \text{otherwise,} \end{cases}$$

and $S_{i0} = S_{0j} = 0$ for all i, j.

We also use an example to explain it, there are two text files:

1. X = p q s j t u v
2. Y = p q r s j t u v

Suppose that X represent the rows of the dynamic programming array and Y represent the columns. For easily describing, here we only use the Smith-Waterman algorithm to compute, and the Levenshtein distance method is not included.

Figure 3 shows the resulting dynamic programming table.

		1	2	3	4	5	6	7
		p	q	s	j	t	u	v
1	p	1						
2	q		2	1				
3	r		1	1				
4	s			2	1			
5	j			1	3	2	1	
6	t				2	4	3	2
7	u				1	3	5	4
8	v					2	4	6

Figure3: Using simplified Smith-Waterman algorithm

The largest number among the red one is the degree of similarities. The numbers are larger, the similarities are higher.

Generally, when we computer the similarity of the texts, we will first use the Levenshtein distance method to divide the table, after dividing the table and then making some of the portions don't calculate, we applied simplified Smith-Waterman algorithm to the rest of the table, because of the less nodes of the table will be compute than Levenshtein distance.

4. Experiment results

We have run experiments on a batch of text files. The experiment is named Adam. Adam is a collection of 100 individual pieces of coursework on artificial intelligence topics, among which including 30 pseudo-plagiarized documents made artificially by means of copy, cut, paste, and mix actions.

We implemented three different programs to prove the results. The first (general) made no attempt to exploit scarcity, only calculating each entry in the dynamic programming table. The second (Levenshtein distance) used the diagonal line to compute it. The third (Smith- Waterman) applied the simplified Smith-Waterman algorithm to locate the similarities. Each of them used the greedy strategy to select matches. An option was also provided to employ a dictionary of synonyms, which the synonymy got the same value.

For these comparisons, we set a threshold value of 10 for significant matches, and flagged all pairs of files that reached a score of 70 or more. The overall average length of the files was 2578 tokens, and we discovered that using the Levenshtein distance method, 5.56% of the elements across all of the dynamic programming tables were non-zero, and the Smith-Waterman algorithm, 4.48% of the elements across all of the dynamic programming tables were non-zero. The results are summarized in Table 2, taken by the general and Levenshtein distance and Smith-Waterman algorithm programs (Algorithms G and L and SW) in each case. The table also reports the average score across all pairs of files. In view of the small proportion of non-zero elements in the dynamic programming table, we would expect a substantial speed-up in moving from the general to the Levenshtein distance version. Of course, the speed-up achieved is reduced somewhat by the preprocessing of the strings, and by the significantly more complex algorithm required to compute successive non-zero elements, as compared to just successive elements, in the table.

978-0-7695-3161-8/08 $25.00 © 2008 IEEE

Table 2 : Pairwise comparison of a batch of text files

Files	Ave. length	% non-zero	Ave. score	Alg G	Alg L
100	2578	5.56	28.2	582	103
Files	Ave. length	% non-zero	Ave. score	Alg G	Alg SW
100	2578	3.48	34.6	582	216

5. Summary and conclusion

We have described how the Levenshtein distance can be used to change the likely scarcity, which can improve both time and space efficiency, and how the simplified Smith-Waterman algorithm can be realized for significant local similarities. From the experiment results it can be proved that it is a powerful tool for the detection of plagiarism in practice.

Further efficiency gains in adapting the algorithm to this context would be valuable. The implementation of the improvement here might not be fast enough to an open web-based detection service, which has lots of date. In the future, it would be interesting to explore appropriate heuristics in the area of text comparison, which will decrease the disadvantages that the Smith-Waterman is too slow for large-scale application. It is also the importance of artificial intelligence research.

Acknowledge

This work was supported in parts by Ubiquitous Autonomic Computing and Network Project, 21th Century Frontier R&D Program, MIC, Korea.

References

[1] D.B. Guralnik and Ed. Cleveland, William Collins World Pub, 1976.
[2] X. Huang and W. Miller, "A time-efficient linear-space local similarity algorithm", Advances in Applied Mathematics, 12:337 –357, 1991.
[3] T.F. Smith and M.S. Waterman, "Identification of common molecular sub-sequences", Journal of Molecular Biology, 147:195-197, 1981.
[4] S.-Y. Noh, S. Kim, C. Jung, "A lightweight program similarity detection model using XML and Levenshtein distance", International Conference on Frontiers in Education: Computer Science and Computer Engineering, Las Vegas, Nevada, 2006.
[5] M. Joy, M. Luck, "Plagiarism in programming assignments", IEEE Trans. on Education 42, 1999.
[6] M. J. Wise, "YAP3: Improved detection of similarities in computer program and other text", 27th SIGCSE, Technical Symposium, PA, pp 130-134, 1996.
[7] D. S. Hirschberg, "A linear space algorithm for computing maximal common subsequences", Communications of the ACM 18, 1975.
[8] T. Lancaster. E_ective and E_cient, " Plagiarism Detection" PhD thesis, School of Computing, Information Systems and Mathematics, South Bank University, 2003.
[9] V.Bafna, B.Narayanan, and R.Ravi, "Non-overlapping local alignments (weighted independent sets of axis-parallel rectangles)", Discrete Applied Mathematics, 1996.
[10] Bao, J. P., J. Y. Shen, X. D. Liu, and H. Y. Liu, "A fast document copydetection model", Soft Computing 10, 41–46, 2006.
[11] Lane, P. C. R., C. Lyon, and J. A. Malcolm, "Demonstration of the Ferret, plagiarism dectector", Proceedings of the 2nd International Plagiarism Conference, 2006.

978-0-7695-3161-8/08 $25.00 © 2008 IEEE