Research on String Similarity Algorithm based on Levenshtein Distance

Shengnan Zhang[1] ,Yan Hu[1] , Guangrong Bian[2]
1. School of Computer Science and Technology, Wuhan University of Technology, Hubei Province, Wuhan, China
2. Department of Aviation Ammuniton, Air Force College of Service, Jiangsu Province, Suzhou, China
1653921195@qq.com, 15927221013@163.com, 44973901@qq.com

***Abstract*—The application of string similarity is very extensive, and the algorithm based on Levenshtein Distance is particularly classic, but it is still insufficient in the aspect of universal applicability and accuracy of results. Combined with the Longest Common Subsequence (LCS) and Longest Common Substring (LCCS), similarity algorithm based on Levenshtein Distance is improved, and the string similarity result of the improved algorithm is more distinct, reasonable and accurate, and also has a better universal applicability. What′s more in the process of similarity calculation, the Solving algorithm of the LD and LCS has been optimized in the data structure, reduce the space complexity of the algorithm from the order of magnitude. And the experimental results are analyzed in detail, which proves the feasibility and correctness of the results.**

Keywords—edit distance; the longest common subsequence; the longest common substring; similarity calculation

I. INTRODUCTION

String similarity has a very wide range of applications, such as text comparison, plagiarism detection, web search, etc. And its calculation method are many, such as the edit distance algorithm, the longest common substring algorithm, greedy string matching algorithm, Heckel algorithm, RKR - GST algorithm, etc. When comparing the similarity of the two strings, it is based on the different characteristics can be divided into based on literal similarity, based on statistical correlation, and the method based on semantic similarity [1].Among them, the classic and widely used is the edit distance algorithm based on literal similarity. In recent years, for different application requirements of the LD algorithm to improve the method of endless, Such as the literature[2] proposed a new formula to calculate the string similarity, which improved the accuracy of word matching; Literature [3] considering the exchange between the adjacent characters, minimize number of edit operation; Literature[4] that the text similarity is calculated from the entropy of the multi-common string, and the accuracy is improved; Literature [5] is proposed based on the LCS and edit distance algorithm research. According to edit distance and similarity problem, many improved algorithm from different aspects to improve the accuracy of the calculation results.

These approaches based on edit distance often consider only the number of edits and neglect the effect of the longest common subsequence (LCS) on the similarity of the strings, or consider the LCS but ignores the longest common substring (LCCS), The traditional similarity formula based on the edit distance cannot get reasonable results. For example, a string S = "abcd" and T1 = "dcba" and T2 = "cdab" compare the similarity, if not considering the LCS, due to LD are 4 will not be able to judge the T2 and S more similar; As another example, the string S = "abcdef" and T1 = "amcnf" and T2 = "abcmng" compare the similarity, because LD and LCS are 3, so even considered the LCS cannot judge the T2 and S more similar, Then we can use LCCS as the distinguishing factor, on this basis, this paper comprehensive consider LD, LCS and LCCS, presents a new string similarity algorithm.

II. STRING SIMILARITY ALGORITHM BASED ON EDIT DISTANCE

A. String Similarity Calculation Based on Edit Distance

The edit distance is proposed by Russian scientist Vladimir Levenshtein in 1965, character as the editing unit, which is the minimum number of operations (insert, delete, replace) that a string is turned into another string. It is often used in the similarity calculation of strings. Set two strings S and T with length m and n respectively, construct matrix LD [n + 1, m + 1], circulation calculating the value of each cell LD (i,j) in the matrix, a calculation formula is as follows:

$$LD(i,j) = \begin{cases} 0, & i = 0, j = 0 \\ j, & i = 0, j > 0 \\ i, & i > 0, j = 0 \\ Min, & i > 0, j > 0 \end{cases}$$

Min=min{LD(i-1,j)+1,LD(i,j-1)+1,LD(i-1,j-1)+f(i,j)},where f(i, j) =1 if the ith word of S is not equal to the jth word of T, otherwise f (i, j) = 0. Finally, the LD (n, m) of the rightmost corner of the matrix is the size of the desired edit distance.

LD itself can represent the similarity of the two strings, intuitively, the greater the LD, the smaller the similarity. Assuming that there are two strings S and T of length m and n respectively, using LD to express their edit distance, using Sim (S, T) to show their similarity [6], then there is:

$$Sim(S,T) = 1 - \frac{ld}{\max(m,n)} \qquad (1)$$

But the formula (1) does not have a good general applicability. For example, set the string: A:aec, B:bcaea, C:aeac,then LD (A, B) =3, LD (A, C) =3, max (A, B) =5, max (A, C) =4, by the formula (1): Sim (A, B) =1-3/5 is greater than Sim (A, C) =1-3/4, which indicates that the similarity of A and B is greater than A and C, but it is more similar to A and C obviously, and the improved similarity calculation formula is given below.

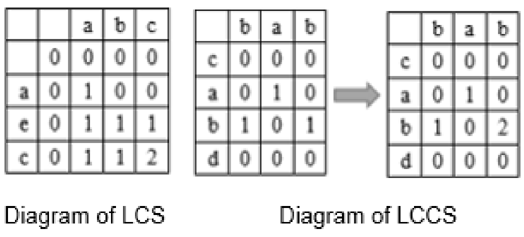
B. String Similarity Algorithm Based on LCS and LCCS

LCS is the longest common sub sequence of two strings, in the case of the relative position of the character is not required to be continuous, he reflects the similarity of the string, often used for similarity checks. The definition of the longest common sub string is similar to the LCS, except that it requires the longest common subsequence that must be consecutive, and denoted by LCCS (Longest Continuous Common Substring) in this paper for the sake of distinction. From the above analysis, it can be seen that the traditional method for similarity based on the edit distance is not good in accuracy, discrimination and applicability. In this paper, combined with the LCS and LCCS to improve the algorithm. First, the traditional LCS and LCCS algorithm [7] is introduced, and then the improved string similarity calculation formula in this paper is introduced.

There are many methods [8] to solve the length of LCS, such as the method of the exhaustive method: for all the subsequence of the T-string, check whether it is a subsequence of S to determine if it is a common subsequence, and then choose the longest, but for two strings of length m and n, the time complexity is O (2^n*2^m) at an exponential rate. Recursive method: simple programming and easy to understand, but with a large number of repeated recursive calls, efficiency is not high. Classical dynamic programming method: Simplification of complex problems, introduced an array to store all the solutions to the sub problems, eliminating the trouble of solving the same sub problems repeatedly. When seeking LCS, using L[i, j] records the length of the longest common subsequence of Tj and Si, and finally, the length of the longest common sub sequence of S and T is recorded in L[m,n], and the recursive relation is established as follows:

$$L[i,j] = \begin{cases} 0, & i = 0 \text{ or } j = 0 \\ L[i-1, j-1] + 1, & i, j > 0, Si == Tj \\ \max(L[i, j-1], L[i-1, j]), & i, j > 0, Si \,!= Tj \end{cases}$$

When computing LCCS [9], the first to construct a two-dimensional matrix L [m+1][n+1] and initialization, if Si=Tj, then L[i][j]=1; Finally, find the cells that value is1 on the oblique diagonal of the matrix and their corresponding string, then the longest string is LCCS. Considering the trouble of finding, when the matrix need to fill 1,let it directly equals to the value of upper left corner plus 1, then the largest element of this matrix is the length of the LCCS, the first character position of the LCCS is the coordinate output of the first cell that has changed. Fig. 1 shows an example of the process of seeking LCS and LCCS.



		a	b	c
	0	0	0	0
a	0	1	0	0
e	0	1	1	1
c	0	1	1	2

	b	a	b
c	0	0	0
a	0	1	0
b	1	0	1
d	0	0	0

	b	a	b
c	0	0	0
a	0	1	0
b	1	0	2
d	0	0	0

Diagram of LCS
Diagram of LCCS

Fig. 1. The process of solving the LCS and LCCS

When comparing the similarity between the two strings S and T, the size of the longest common subsequence is important in addition to the intuitionistic effect of the edit distance, then combining LD and LCS gives another similarity Calculated as follows:

$$Sim(S, T) = 1 - \frac{ld}{lcs + ld} \qquad (2)$$

Now, look at the example again, use the formula (2): S (A, B) =1-3/5 less than S (A, C) =1-3/6, which can correctly show that the similarity of A and B is less than A and C. The experiment shows that this formula is suitable and good to expand the general applicability of the traditional formula, but the situation two mentioned in the introduction has not been resolved. The following will be combined with the longest common substring (LCCS) to continue to improve the similarity calculation formula.

III. STRING SIMILARITY OPTIMIZATION ALGORITHM BASED ON LD, LCS AND LCCS

A. Optimization Algorithm of Edit Distance Based on Column Vector

The space complexity and time complexity of the traditional edit distance algorithm are O (m * n). Considering every time when calculating LD(i, j) only use LD(I,j-1),LD(i-1,j) and LD(i-1,j-1), that is, only the current column and the previous column are required to participate in the calculation, Therefore, this paper uses two column vectors instead of matrix, Save only the current state and the state of the last operation, Experiments show that in the case of Basically does not affect the results and time complexity, the spatial complexity is reduced to O(min (m, n)). The optimized edit distance algorithm is described as follows:

Input: string S, T

Output: the length of LD

1) set the length of S is *n*, the length of T is *m* (set n<m), if *m*=0, return *n*; if *n*=0, return *m*; create two vectors *v0[m*+1*]* and *v1[m*+1*]*;

2) Initialize the value of *v0 [m* + 1*]* to be 0 ... *m*;

3) Check each character in *S* (*i* from 1 to *n*);

4) Check each character in *T*(*j* from 1 to *m*);

5) Let *cost* be the edit cost, if *s[i]==t[j]*,the *cost*=0;if *s[i*]!=*t[j]*, then *cost*=1;

6) The value of *v1[j]* is the minimum of the following three:

a、Above the unit+1:*v1[j-1]* + 1

b、Next to the left side of the unit+1:*v0[j]* + 1

c、The upper left corner of the unit +*cost*:*v0[j-1]* + *cost*

7) After completing the iterations (3, 4, 5, 6), *v1[m]* is the value of the edit distance.

B. Optimization of The Longest Common Subsequence Algorithm Based on Dimension Reduction

Considering that only *L[i-1, j-1]*, *L[i, j-1]* and *L[i-1,j]* are directly related when calculating *L[i, j]*,that is, in calculating the sub problem of the row *L[i]*,it is only necessary to know the value of the row *L[i-1]* and the value of the previous element *L[i, j-1]* of the current *L[i, j]*, at this time the space complexity decreases from O (m * n) to O (n) without affecting the time complexity. Define an array *base[]* of length is *m+1* and initialized to *0*, a recursive preamble *front=0*, a current calculation element *pre* (meaning L[i, j]), after each calculation of L[i, j], the value of *front* is updated to *base [j-1]*, then *pre* update to *front*, and then continue to scan back, pay attention to update the last element. Iteration n times, and finally the value of *base[m]* is the value of LCS. The optimized LCS algorithm is described as follows:

```
Input: string S, T
Output: the length of LCS
1) Input two strings S, T;
2) Define an array base[] with size of T.length()+1, define a
   recursive preambl front=0, the current element pre = 0;
3) Initialize base[] to 0, base[i]=0;
4) Check each character in S(i from 1 to S.length());
5) Check each character in T(i from 1 to T.length());
6) Cyclic checking, if S[i-1]=T[j-1], pre=base[j-1]+1;
   otherwise pre = max (front, base [j]);
7) Cycle update base and front, base[j-1]=front, front=pre;
8) Update base[T.length()]=front ,Check base[j] (j from 1 to
   T.length()) and output the value of base[];
9) Cyclic execution step (4, 5, 6, 7, 8), in the end, the
   base[T.length()] is the value of LCS.
```

C. Optimization of String Similarity Calculation Method Based on The Ld, The Lcs And Lccs

Now use *l(S, T)* to represent the length of the LCCS of string S and T, after taking into account its influence, case two in the introduction has a solution. Now reanalyze the similarity of S and T1, T2, when the LD and LCS are the same, the similarity can be judged by comparing the LCCS. Calculated *l(S, T1)* =1 less than *l(S, T2)* =3, it can be concluded that the similarity between T2 and S is higher, which is consistent with the way people's subjective judgment. The problem now facing is how to merge *l(S, T)* into the similarity calculation formula and how to determine the similarity when *l(S, T)* is also the same. Here is an example, set the string S = "abcmg", T1 = "abcnp", T2 = "ebcmf", Now find the impact of the three key factors are the same size, also cannot calculate which group of similarity is higher, but the result More close to artificial subjective judgment is that S and T1 are more similar. This is based on the location of LCCS to determine, in this paper, the solution is to comprehensively consider the length and position of LCCS, and introduce a variable *p* to denote the first position of the longest common substring (LCCS) in the target string S. When both LCS and LD are the same, the greater the *l(S, T)*, the greater the similarity. When the LCCS length is also the same, taking into account the effect of the precursor character(a character before the current processing character) is bigger than the subsequent character(a character after the current processing character) on the similarity, then the smaller the *p*, indicating that the LCCS's first character position is more forward, the greater the similarity, this paper introduce *p/(l*m)* as a factor to check the impact of LCCS on the similarity, *m* is the length of the shorter string, μ is a variable that balances this factor, which can be artificially set according to the rigidity of LCCS requirements. Define the new similarity calculation formula as follows:

$$Sim(S,T) = \frac{lcs}{lcs + ld + \frac{p}{l * m} * \mu} \qquad (3)$$

At this point re-analysis of the above example, let μ=1, and use the formula (3) to calculate the similarity: Sim(S,T1)=0.592, Sim(S,T2)=0.584, which shows that the similarity of S and T1 is greater than the similarity of S and T2.Compared with formula (1)and(2),the result is more in line with the actual situation, the distinction is better, and the experimental results verify its universal applicability. Set the length of S and T are m and n respectively, According to the optimization algorithm steps above to obtain the edit distance (LD(S, T)) and the length of the longest common subsequence (LCS(S, T)).According to the steps of the LCCS algorithm to obtain the length of the longest common substring (LCCS(S, T)) and the first character position of the substring (*p)*, which filter out the meaningless full symbol comparison. According to the similarity formula proposed in this paper, using Java language, algorithm process is as follows:

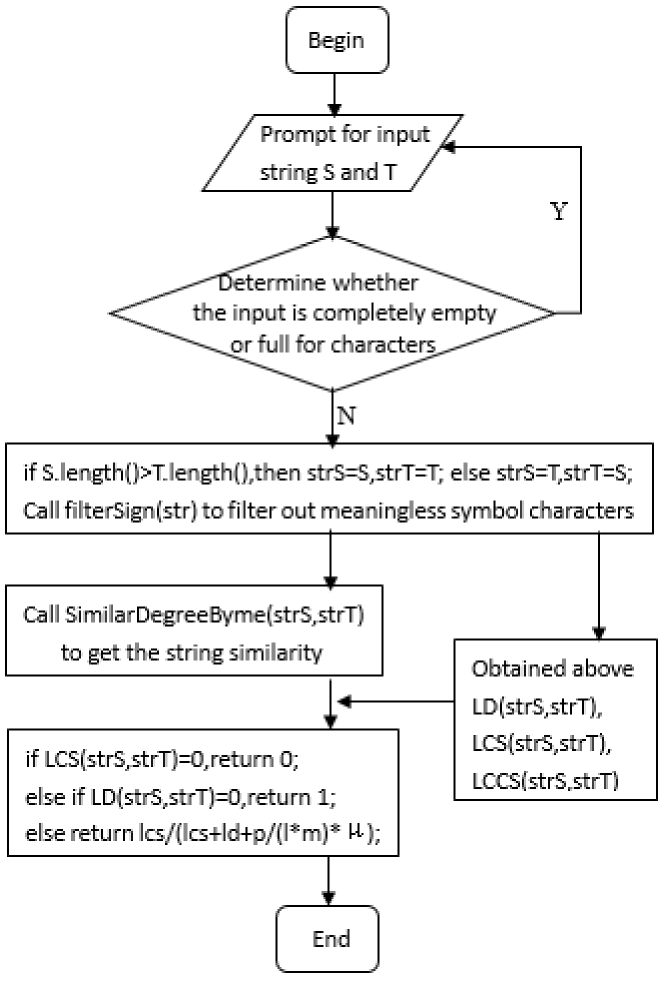

Begin
Prompt for input string S and T
Y
Determine whether the input is completely empty or full for characters
N
if S.length()>T.length(),then strS=S,strT=T; else strS=T,strT=S;
Call filterSign(str) to filter out meaningless symbol characters
Call SimilarDegreeByme(strS,strT) to get the string similarity
Obtained above LD(strS,strT), LCS(strS,strT), LCCS(strS,strT)
if LCS(strS,strT)=0,return 0;
else if LD(strS,strT)=0,return 1;
else return lcs/(lcs+ld+p/(l*m)* μ);
End

Fig. 2. The algorithm process

IV. EXPERIMENTAL ANALYSIS AND CONCLUSIONS

Two groups of data were used to analyze the accuracy, applicability and discrimination of the algorithm. The first set of data selects the source string S='expect', and Select a group of words that are close to S in the English dictionary to form a set of target strings. W={w1, w2, w3, w4, w5, w6, w7 }={'spectator', 'exercise', 'pecuniary', 'accept', 'excerpt', 'exempt', 'aspect' }. The second set of data selects the source string S='abcde01234', target strings listT={t0, t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13 }={'wqefghlm56234', 'e01abcd', 'fbcdm56789', 'fghlm51234', 'mgcde05678', 'w01234abc', 'mghln01234', 'abcde56789', 'fghde01234', 'abcde01567', 'fgcde01234', 'abcde012fg', 'fbcde01234', 'abcde01235' }.

A. Analysis of Similarity Results

Respectively using the three similarity calculation formula (1) (2) and (3) to calculate the similarity. The comparative analysis of the string similarity calculation results of the two sets of data are shown in Figure 3 and 4. In addition, in order to describe the experimental conclusion more detailed, easy to compare and illustrate, given the contrast of the first set of data, as shown in Table I.

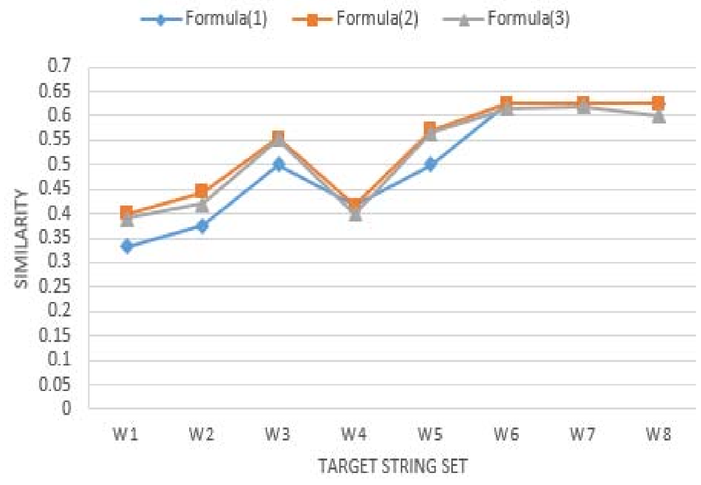


Fig. 3. Results of similarity of data set W

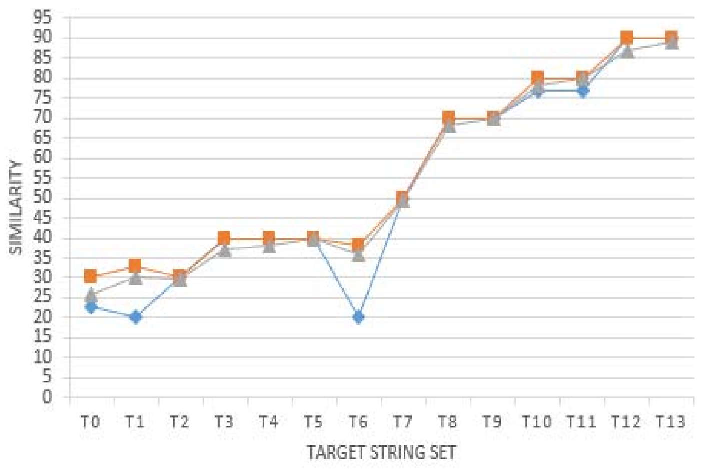


Fig. 4. Results of similarity of data set T

TABLE I. DATA CONTRAST

Groups	Target String	Formula(1)	Formula(2)	Formula(3)
1	spectator	0.333	0.400	0.395
2	exercise	0.375	0.444	0.440
3	expceted	0.500	0.555	0.552
4	bespectacled	0.417	0.417	0.412
5	exempt	0.500	0.571	0.565
6	earpecte	0.625	0.625	0.615
7	expedite	0.625	0.625	0.622
8	expdctse	0.625	0.625	0.620
	RANGE	0.292	0.225	0.227
	STANDARD DEVIATION	0.117885	0.097	0.097

B. The Experiment Conclusion

It can be seen from Figure 3, 4, In the inspection of the target string sets w1~w8 and t1~t13, the results of formula(2)and(3)is more reasonable and accurate than formula(1), which avoids the case of similarity is extremely unreasonable caused by the change of character sequence. The trend of the similarity results from formula (3) is basically the same as formula (2) and broadly between (1) (2), which shows the applicability and stability of the formula (3).When the two lines of formula (1) (2) coincide with each other, that is, the two methods cannot distinguish the similarity, while the line of formula (3) has a change of rise or fall, which can get a better distinction result. It can be seen from the groups 3and5 in table 1, when LD is the same, the formula (2) (3) has a more reasonable and more distinguishable result than (1), similarly, it can be drawn from groups 6 7and 8 that when LD and LCS are the same, the formula (3) can get the similarity data more rational and differentiated than (1) and (2). The data varies with the length and location of the LCCS, decreases with the increase of the value of $p/(l(S,T)*m)*\mu$. The formula (3) has a more moderate standard deviation and range relative to formulas (1) and (2), so that the results of the similarity is more reasonable.

In conclusion, the formula (3) has a good general applicability, and the result is more discriminative, accurate and reasonable than formulas (1) and (2).

V. CONCLUSION

In this paper, the calculation of string similarity is further improved according to the idea of raising, analyzing and solving problems. Taking into account the important influence of LCS and LCCS on the computation of string similarity to improve the algorithm, and extensive experimental show that optimized algorithm improv es the general applicability of the algorithm and the accuracy of the results, moreover, a better discriminative similarity result can also be obtained for that string with very small differences. In addition, the traditional calculation process of LD and LCS is optimized in terms of data structure, which further reduces the space complexity of the algorithm in the case that the time complexity is basically not affected.

REFERENCES

[1] Minghe, Guoliang, Dong, et al. String similarity search and join: a survey[J]. Frontiers of Computer Science Selected Publications from Chinese Universities, 2016, 10(3):399-417.

[2] Allison L． Dynamic Programming Algorithm(DPA) for Edit － Distance ［EB /OL］． ［2015 － 07 － 10］． http: / /www． allisons． org /ll /AlgDS / Dynamic /Edit /.

[3] JIANG Hua, HAN An-qi, WANG Mei-jia. Solution Algorithm of String Similarity Based on Improved Levenshtein Distance[J]. Computer Engineering, 2014, 40(1):222-227.

[4] Diao X C, Tan M C, Cao J J. New method of character string similarity compute based on fusing multiple edit distances[J]. Application Research of Computers, 2010, 27(12):4523-4525.

[5] Nguyen T T A, Conrad S. An Improved String Similarity Measure Based on Combining Information-Theoretic and Edit Distance Methods[M]// Knowledge Discovery, Knowledge Engineering and Knowledge Management. Springer International Publishing, 2014.

[6] Artur Niewiarowski, Marek Stanuszek，Mechanism of analysis of similarity short texts, based on the Levenshtein distance， studiainformatica，Vol 34, No 1 (2013).

[7] H Jiang，AQ Han，MJ Wang，Z Wang，WU Yun-Ling，Solution Algorithm of String Similarity Based on Improved Levenshtein Distance，《Computer Engineering》,2014,40(1):222-227.

[8] Liu J, Wu S. Research on longest common subsequence fast algorithm[C]// International Conference on Consumer Electronics, Communications and Networks. IEEE, 2011:4338 – 4341.

[9] Kostakis O, Papapetrou P. Finding the longest common sub-pattern in sequences of temporal intervals[J]. Data Mining & Knowledge Discovery, 2015, 29(5):1-3.