An Efficient Algorithm for Finding the Most Similar Sequence on the Manhattan Distance Model*

Hsiu-Chieh Hung[1], Chang-Biau Yang[1]†and Kuo-Si Huang[2]
[1]Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan
†cbyang@cse.nsysu.edu.tw
[2]Department of Business Computing
National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan

Abstract

Since the similarity (such as LCS, DTW) calculation of two given sequences usually requires quadratic time, sequential search is time-consuming for finding the most similar sequence in a database. This paper discusses the searching probability of a sequence and proposes a method for determining the searching order to reduce the searching time. On the Manhattan distance model, the searching probability of a query sequence is calculated for selecting the next compared sequence. Accordingly, the third and the subsequent compared sequences can be determined. Hence, a searching strategy of parameter $\langle 0.81, 1\rangle$ is proposed for finding the most similar sequence, where 0.81 and 1 indicate the multipliers of the edit distance between the reference sequence and the query sequence, respectively. Our searching strategy considers the triangle inequality of the edit distance for accelerating the searching efficiency by pruning some unnecessary sequences away.

Keywords: bioinformatics, sequence similarity, longest common subsequence, edit distance, prune and search, Manhattan distance model

1. Introduction

For finding the most similar sequence to a query sequence in a database, one can apply the *sequential search* to compare the query sequence with each sequence in the database. When comparing two sequences, one can simply use the *dynamic time warping* (DTW) to calculate the distance between them if they are time series [1–3]. For character sequences, we may apply the *longest common subsequence* (LCS) algorithm to compute their identities or similarities [4, 5], or calculate their edit distance [6, 7]. The LCS and the edit distance are two different measures to explore the similarity between character sequences. The above two measurements with the *dynamic programming* (DP) approach require $O(mn)$ time, where the lengths of two sequences are m and n, respectively. Given a query sequence, it takes plenty of time if we use the one-by-one sequential search for finding the most similar sequence in a database.

Many studies used the heuristic methods to find a feasible solution for speeding up the searching efficiency, such as the BLAST [8, 9] and the Clustal [10] in bioinformatics. For finding similar DNA sequences, BLAST first utilizes a heuristic method to find out some similar segments between two sequences, and then expands these segments to check whether the two sequences are similar [8, 9]. The Clustal software was created to solve the *multiple sequence alignment* problem [11], and it reduces the processing time by using the *evolutionary tree* [10, 12]. In addition, Park *et al.* proposed a method by combining the DTW and the bounding strategy with the *suffix tree* to improve the searching efficiency [13].

However, these studies did not provide any theoretical analysis to give the reason that we should apply the algorithms. As a result, we try to provide the theoretical analysis for our searching method. Furthermore, our aim is not only a feasible solution, we tend to focus on finding the optimal solution in a character-sequence database. The *Voronoi diagram* can solve this kind of problem for numeric data in the *Euclidean metric*, because it can divide the sphere of influence as the preprocessing and afterward find the best solution quickly [14]. However, it is not suitable for the character-sequence, because the sequence dimensions or lengths are variant and the distance measurement of character-sequence is obviously not in the Euclidean metric. Hence, we choose the edit distance as our distance measurement, and utilize the *triangle inequality* property on the *Manhattan distance* model for theoretical analysis. Afterwards, we propose the sequence searching orders, and prune unnecessary sequences away to improve the searching efficiency.

The organization of this paper is given as follows. Section 2 introduces the triangle inequality and the Manhattan distance as background knowledge, then describes and defines the terminologies used in this paper. Section 3 presents the searching strategy for finding the most similar sequence in a character-sequence database. The experimental results are shown in Section 4 to illustrate the efficiency of our searching strategy. Finally, the conclusion is given in Section 5.

2. Notations and Background

In this paper, the hth sequence in a character-sequence database S is denoted as S_h. Taking S_h as

*This research work was partially supported by the Ministry of Science and Technology of Taiwan under contract MOST 107-2221-E-110-033.

†Corresponding author.

an example, the notations used in this paper are listed as follows.

- $|S_h|$: the length of sequence S_h.
- s_h^i: the ith character or element of S_h.
- $[i..j]$: an index range from index i to j.
- $S_h[i..j]$: the substring of S_h from index i to j. Note that $S_h[i..j] = \emptyset$ if $i > j$.

The *edit distance* is used to measure the minimum difference of two sequences for converting one sequence to the other. The greater the edit distance, the more dissimilar they are. There are three kinds of traditional operations in the edit distance, including *insertion*, *deletion* and *substitution*. Assuming we want to convert sequence S_1 to S_2, one can align S_1 and S_2 first by placing some gaps to make an alignment of S_1 and S_2. And then it uses the three operations to calculate the edit distance of this alignment.

In this paper, the costs of edit distance for each insertion, deletion, and substitution are assumed to be 1, 1, and 2, respectively. With such cost assignment, in fact, one substitution can be viewed as one insertion and one deletion. That is, it is equivalent to that only insertions and deletions are considered.

Definition 1. *Given two sequences S_1 and S_2 of lengths m and n, respectively, the edit distance is the minimum cost for converting S_1 to S_2.*

An edit distance algorithm by using DP with both $O(mn)$ time and space was proposed by Wagner and Fischer [7], given in (1), where $\gamma(\cdot)$ is the cost function of an operation, $ED(i,j)$ denotes the edit distance of $S_1[1..i]$ and $S_2[1..j]$, and ε denotes the empty string.
$ED(i,j) =$

$$\min\begin{cases} i + \gamma(S_1^i, \varepsilon) & \text{if } j = 0,\\ j + \gamma(\varepsilon, S_2^j) & \text{if } i = 0,\\ \begin{cases} ED(i-1, j-1) + \gamma(S_1^i, S_2^j) \\ ED(i-1, j) + \gamma(S_1^i, \varepsilon) \\ ED(i, j-1) + \gamma(\varepsilon, S_2^j) \end{cases} & \text{otherwise.} \end{cases} \tag{1}$$

The *triangle inequality* is an important property for the edit distance. Chen and Ng applied the triangle inequality of the edit distance and the lower bound of time series on DTW, and they improved the efficiency in their *pruning strategy* [15]. This paper focuses on the character sequences, rather than the time series. For the character sequences, inspired by the DTW, we apply the triangle inequality property of three sequences and the lower bound of the distance between two sequences.

We first give a simple example to illustrate the triangle inequality property. Suppose that we are given three distinct sequences S_1, S_2 and S_3, where $d_{S_1,S_2} = 6$ and $d_{S_1,S_3} = 8$. We can grasp the range of the edit distance between S_2 and S_3 without comparing them directly due to the triangle inequality. We can derive $|d_{S_1,S_2} - d_{S_1,S_3}| \leq d_{S_2,S_3} \leq d_{S_1,S_2} + d_{S_1,S_3}$ since the edit distance satisfies the triangle inequality [16]. Since the costs of each insertion, deletion and substitution are 1, 1, and 2, respectively, every two neighboring possible edit distances between S_2 and S_3 differ by 2. That is, the value of d_{S_2,S_3} may be one of $|d_{S_1,S_2} - d_{S_1,S_3}|$, $|d_{S_1,S_2} - d_{S_1,S_3}|+2$, $|d_{S_1,S_2} - d_{S_1,S_3}|+4, \cdots,$ and $d_{S_1,S_2} + d_{S_1,S_3}$. Thus, we have the following theorem, whose formal proof is presented in [17].

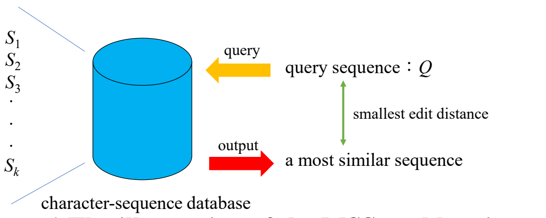


Figure 1 The illustration of the MSS problem in a character-sequence database.

Theorem 1. *Suppose that the costs of each insertion, deletion, and substitution operation are 1, 1, and 2, respectively. Given three sequences S_1, S_2 and S_3, the triangle inequality $|d_{S_1,S_2} - d_{S_1,S_3}| \leq d_{S_2,S_3} \leq d_{S_1,S_2} + d_{S_1,S_3}$ holds. Furthermore, $d_{S_2,S_3} \in \{|d_{S_1,S_2} - d_{S_1,S_3}|, |d_{S_1,S_2} - d_{S_1,S_3}|+2, |d_{S_1,S_2} - d_{S_1,S_3}|+4, \cdots, d_{S_1,S_2} + d_{S_1,S_3}\}$.*

The L_p-norm is a distance calculation model in the *Euclidean space*. It can be applied on multi-dimensional coordinates where $1 \leq p \leq \infty$. Equation (2) shows the formula of the L_p-norm, where $v = \{v_1, v_2, \cdots, v_t\}$ is a vector [18].

$$||v||_p = (|v_1|^p + |v_2|^p + \cdots + |v_t|^p)^{1/p} \tag{2}$$

The most common application is the L_2-norm, which is also known as the *Euclidean distance*. In addition, the L_1-norm $|v_1| + |v_2|$ is also known as the *Manhattan distance* on the two-dimensional space.

Finding the similar sequence in a database is a practical problem in bioinformatics. Many studies utilize the heuristic methods, but only a few focus on the *most similar sequence* (MSS) problem. The MSS problem deals with numeric data in many applications, such as time series [15]. In this paper, we are interested in character-sequence data. Formally, in the MSS problem, given k sequences in a database and a query sequence Q, we want to find the most similar sequence in the database for Q. Fig. 1 illustrates the problem concept.

3. Our Algorithm

3.1 First and Second Compared Sequences

First, a preprocessing is performed to obtain a pairwise distance table, which records the edit distance between each pair of sequences in the database. For example, suppose that a database consists of six sequences, $S = \{S_1, S_2, S_3, S_4, S_5, S_6\}$. Table 1 shows an example of the pairwise edit distances in the database. Suppose that there is no information provided between query sequence Q and each sequence in the database. We can select the first compared sequence randomly in the database. Assume that the first compared sequence is $\hat{S}_1 = S_2$. We calculate the edit distance between $\hat{S}_1$ and Q, $d_{\hat{S}_1,Q}$.

Assume S_X in S is the second compared sequence of Q. By Theorem 1, we obtain

$$|d_{\hat{S}_1,Q} - d_{\hat{S}_1,S_X}| \leq d_{S_X,Q} \leq d_{\hat{S}_1,Q} + d_{\hat{S}_1,S_X}. \tag{3}$$

Table 1 An example of pairwise distance table calculated in the preprocessing stage. Each value represents the edit distance of the corresponding two sequences.

	S_1	S_2	S_3	S_4	S_5	S_6
S_1	0	6	4	9	7	9
S_2	6	0	4	5	11	13
S_3	4	4	0	7	9	9
S_4	9	5	7	0	8	14
S_5	7	11	9	8	0	8
S_6	9	13	9	14	8	0

If $d_{\hat{S}_1,S_X} > 2 \times d_{\hat{S}_1,Q}$, we have $d_{\hat{S}_1,Q} < d_{S_X,Q} < 3 \times d_{\hat{S}_1,Q}$. As one can see, the solution for $d_{S_X,Q}$ is worse than $d_{\hat{S}_1,Q}$. Thus, we could prune the sequences whose edit distance with $\hat{S}_1$ are greater than or equal to $2 \times d_{\hat{S}_1,Q}$. As a result, we should choose the sequences as S_X such that $d_{\hat{S}_1,S_X} < 2 \times d_{\hat{S}_1,Q}$. $d_{\hat{S}_1,S_X} = 2 \times d_{\hat{S}_1,Q}$ is a bound for selecting the next compared sequence. For improving the search efficiency, we shall find better sequences to decrease the amount of comparisons rather than choosing S_X with this bound or randomly.

To achieve this goal, the detailed analysis of the searching order is presented in [17]. As a brief summary, we get the following equation:

$$d_{\hat{S}_1,S_X} \approx 0.81 d_{\hat{S}_1,Q}. \tag{4}$$

That is, we choose the second compared sequence S_X whose edit distance with the first compared sequence $\hat{S}_1$ is closest to $\lfloor 0.81 \times d_{\hat{S}_1,Q} \rfloor$.

3.2 Third and Subsequent Sequences

By analyzing the searching order, we take the sequence with the current smallest edit distance to Q as the reference sequence S_r. Let S_c be the current sequence compared with Q, whose $d_{S_c,Q}$ has been determined. Let S_X be the next sequence to be compared with Q. We use the multiplier factor 0.81 in (4) to select the third and the subsequent sequences as follows.

$$d_{S_r,S_X} = 0.81 \times d_{S_r,Q} \tag{5}$$

In other words, such S_X seems to have high probability to prune away unnecessary sequences. For convenience, the multiplier factor 0.81 is denoted as $mf_{0.81}$. For example, given an arbitrary value z, $mf_{0.81} \times z$ indicates $\lfloor 0.81 \times z \rfloor$. Also, we still use the triangle inequality to prune away unnecessary sequences.

More precisely, in the experimental results (shown in Section 4), our searching strategy is denoted as $\langle 0.81, 1 \rangle$. If $d_{S_c,Q} \leq d_{S_r,Q}$, then $S_r = S_c$ and we use the guideline $d_{S_r,S_X} = 0.81 \times d_{S_r,Q}$ to select the next compared sequence S_X; otherwise, S_r is not updated and the search guideline becomes $d_{S_r,S_X} = d_{S_r,Q}$.

3.3 The Proposed Algorithm

Fig. 2 shows the flow chart to demonstrate the process of our searching strategy $\langle 0.81, 1 \rangle$. The proposed algorithm for solving the MSS problem in a character-sequence database is described in Algorithm 1. R denotes the set consisting of the remaining sequences in the database that are still needed to compare with Q. An example of the searching process is shown in Table 2 and described as follows, whose inputs are given in Table 1.

Iteration 0: $R = \{S_1, S_2, S_3, S_4, S_5, S_6\}$ initially.

Iteration 1: Suppose that the first compared sequence $S_X = S_2$, which is selected randomly in R. Let $S_r = S_c = S_2$. Suppose $d_{S_X,Q} = d_{S_2,Q} = 6$ is obtained. Prune S_6 away from R since $d_{S_2,S_6} = 13 \geq 2 \times d_{S_2,Q} = 12$ and S_6 cannot be the answer.

Iteration 2: Find S_3 in R to be the next S_X since d_{S_r,S_3} is closest to $mf_{0.81} \times d_{S_r,Q} = \lfloor 0.81 \times 6 \rfloor = 4$. Calculate $d_{S_3,Q} = 4$ (supposed). After computation, $S_X = S_3$ becomes S_c. Because $d_{S_c,Q} \leq d_{S_r,Q}$, let S_c become new S_r. Prune S_5 away from R since d_{S_r,S_5} = $d_{S_3,S_5} = 9 \geq 2 \times d_{S_r,Q} = 2 \times d_{S_3,Q} = 8$. Then, $R = \{S_1, S_4\}$.

Iteration 3: The factor 0.81 is applied again. It takes S_1 as S_X, because $d_{S_r,S_1} = d_{S_3,S_1} = 4$ is closest to $mf_{0.81} \times d_{S_r,Q} = \lfloor 0.81 \times 4 \rfloor = 3$. Calculate $d_{S_1,Q} = 6$ (supposed). After computation, $S_X = S_3$ becomes S_c. It does not change S_r because $d_{S_c,Q} = 6 > d_{S_r,Q} = 4$.

Iteration 4: $1 \times d_{S_r,Q}$ is used to select the next S_X, since in the previous iteration, S_r is not updated. Hence, only S_4 is the next S_X. We calculate $d_{S_X,Q} = 3$. Set $S_c = S_X$. Becasue $d_{S_c,Q} \leq d_{S_r,Q}$, update $S_r = S_c$. Finally, S_r is returned as the solution.

Algorithm 1 Finding the most similar sequence in a character-sequence database.

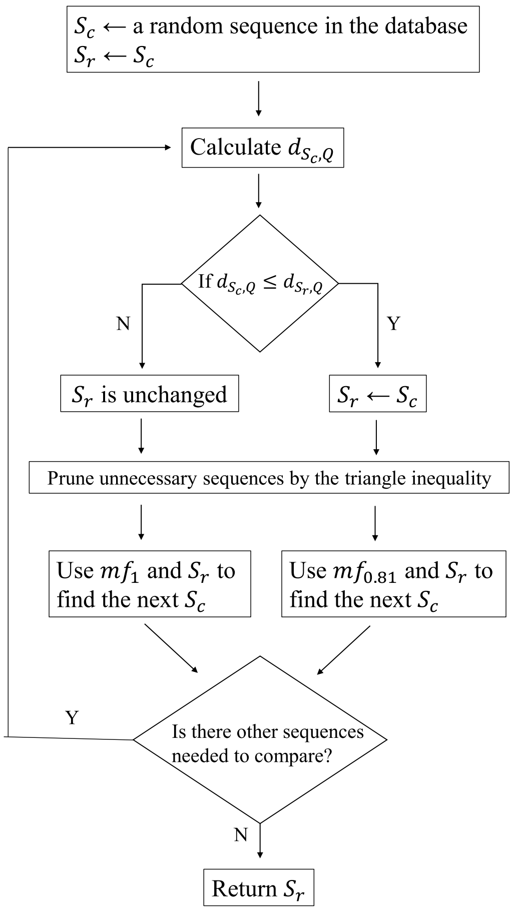
```
Input: $DB$: the character-sequence database;
    $d_{S_i,S_j}$: the pairwise edit distance in $DB$;
    $Q$: the query sequence
Output: The most similar sequence $S_r$ in $DB$
 1: A set $R$ is initialized to contain all sequences in $DB$.
 2: $S_X \leftarrow$ a sequence randomly chosen from $R$
 3: $S_c \leftarrow S_X$, $S_r \leftarrow S_c$, calculate $d_{S_r,Q}$
 4: while $R$ is not empty do
 5:     Calculate $d_{S_X,Q}$
 6:     Remove $S_X$ from $R$
 7:     Remove each $S_i$ from $R$ if $d_{S_X,S_i} \geq 2 \times d_{S_X,Q}$
 8:     $S_c \leftarrow S_X$, $d_{S_c,Q} \leftarrow d_{S_X,Q}$
 9:     if $d_{S_c,Q} \leq d_{S_r,Q}$ then
10:         $S_r \leftarrow S_c$
11:         Select $S_X$ such that $d_{S_r,S_X}$ is closest to $\lfloor 0.81 \times d_{S_r,Q} \rfloor$ if $R$ is not empty
12:     else
13:         Select $S_X$ such that $d_{S_r,S_X}$ is closest to $d_{S_r,Q}$ if $R$ is not empty
14: return $S_r$
```

4. Experimental Results

This paper takes the *expressed sequence tag* (EST) databases of mouse as the experimental dataset. The EST is a short subsequence of *complementary DNA* (cDNA). We randomly extract some sequences from them to form the experimental database DB in our experiments. The testing environment is a computer of 64-bit Windows-10 OS, with an Intel Core i7-2600

Table 2 An example of our searching process with the inputs in Table 1. The symbol "-" indicates empty.

iteration	S_r	$d_{S_r,Q}$	S_c	$d_{S_c,Q}$	mf	R
0	-	-	-	-	-	$S_1, S_2, S_3, S_4, S_5, S_6$
1	S_2	6	S_2	6	$mf_{0.81}$	S_1, S_3, S_4, S_5
2	S_3	4	S_3	4	$mf_{0.81}$	S_1, S_4
3	S_3	4	S_1	6	mf_1	S_4
4	S_4	3	S_4	3	-	-



$S_c \leftarrow$ a random sequence in the database
$S_r \leftarrow S_c$
Calculate $d_{S_c,Q}$
If $d_{S_c,Q} \le d_{S_r,Q}$
N
Y
S_r is unchanged
$S_r \leftarrow S_c$
Prune unnecessary sequences by the triangle inequality
Use mf_1 and S_r to find the next S_c
Use $mf_{0.81}$ and S_r to find the next S_c
Y
Is there other sequences needed to compare?
N
Return S_r

Figure 2 The process of our searching strategy $\langle 0.81, 1\rangle$.

3.40GHz CPU and 16GB RAM. Our algorithm is implemented with the Microsoft Visual C++ 2017.

In the experimental result, the terms are abbreviated including sequential search (Seq), random selection (RS), random generation (RG), and mutation (M). The tuple of six parameters, T($source$, num_{DB}, num_Q, $|\Sigma|$, $avg\_len(DB)$, $len\_scope(Q)$), is defined for illustrating the experimental results more clearly, where $source$ indicates the kind of data type, num_{DB} represents the number of sequences in DB, num_Q represents the number of query sequences, $|\Sigma|$ is the alphabet size, $avg\_len(DB)$ is the average sequence length in DB and $len\_scope(Q)$ is the length scope of each query sequence.

For example, T(mouse, 1000, 1000, 4, 381.7, *) denotes the sequences are randomly extracted from the EST database of mouse to form DB with $num_{DB} = 1000$, $|\Sigma| = 4$ and $avg\_len(DB) = 381.7$. If the query generator is random selection (RS), we randomly extract query sequences from the EST database of mouse with $num_Q = 1000$. If the query generator is mutation (M), we randomly select $\frac{num_Q}{10}$ sequences from DB and use each sequence to produce 10 query sequences by the mutation percentage. If the query generator is random generation (RG), we randomly generate query sequences with $num_Q = 1000$ and $|\Sigma| = 4$. The last symbol $*$ represents the length scope of each Q is variable because we use different query generators to generate query sequences.

The average execution time (in seconds) is the average of the required times for all query sequences under different conditions. The sequence comparison ratio (%) indicates that the average ratio of sequences in DB are really compared with the query sequence for finding the MSS. The average execution time (in seconds) is outside parentheses and the sequence comparison ratio (%) is inside the parentheses.

In Table 3, we extract 1000 sequences randomly from the original EST database of mouse to construct DB with average length 381.7. And it requires 485 seconds to calculate the pairwise edit distance in the preprocessing phase.

For investigating the efficiency of our searching strategy, we run seven searching strategies, including $\langle 0.81, 0.81\rangle$, $\langle 1, 0.81\rangle$, $\langle 0.7, 1\rangle$, $\langle 0.81, 1\rangle$, $\langle 0.9, 1\rangle$, $\langle 1, 1\rangle$ and $\langle 1.2, 1\rangle$. We observe that our searching strategy $\langle 0.81, 1\rangle$ has a significant improvement compared with the sequential search in Table 3. Furthermore, the strategy $\langle 0.81, 1\rangle$ is also faster than $\langle 0.81, 0.81\rangle$ and $\langle 1, 0.81\rangle$ when the query generator is random selection (RS) and the mutation percentages are 1%, 5%, 10% and 20%. The strategies $\langle 0.7, 1\rangle$, $\langle 0.81, 1\rangle$ and $\langle 0.9, 1\rangle$ have comparable performance. In the mutation percentages 1%, 5%, 10% and 20%, we also observe that the sequence comparison ratios of the strategies $\langle 0.9, 1\rangle$, $\langle 1, 1\rangle$ and $\langle 1.2, 1\rangle$ get higher and higher when the first parameter of the strategy becomes larger and larger. This growing trend of the sequence comparison ratio indicates that our searching strategy $\langle 0.81, 1\rangle$ has the minimal expected number of points.

The average numbers of sequences needed to compare with Q in each iteration of the searching strategies $\langle 0.81, 0.81\rangle$ and $\langle 0.81, 1\rangle$ in Table 3 are shown in Fig. 3. We observe that strategy $\langle 0.81, 1\rangle$ is more efficient than $\langle 0.81, 0.81\rangle$ when the mutation percentages are small because the average number of remained sequences in strategy $\langle 0.81, 1\rangle$ decreases more rapidly than $\langle 0.81, 0.81\rangle$.

In Table 4, it requires about 3.5 hours to calculate the pairwise edit distance in the preprocessing phase. The strategy $\langle 0.81, 1\rangle$ is the best strategy when the mutation percentages are less than or equal to 20%. Although the experimental result of the strategy $\langle 0.7, 1\rangle$ in mutation 1% is comparable to $\langle 0.81, 1\rangle$, the strategy $\langle 0.81, 1\rangle$ is still faster than $\langle 0.7, 1\rangle$ in the mutation 5%, 10% and 20%. In the mutation percentages 1%, 5%, 10% and 20%, the average execution times and the sequence comparison ratios of the strategies $\langle 0.9, 1\rangle$, $\langle 1, 1\rangle$ and $\langle 1.2, 1\rangle$ are increasing when the first parameter of the strategy becomes larger. This indicates that our searching strategy $\langle 0.81, 1\rangle$ has the minimal expected number of points again because it is better than $\langle 0.7, 1\rangle$, $\langle 0.9, 1\rangle$, $\langle 1, 1\rangle$ and $\langle 1.2, 1\rangle$. The average execution time in the mutation percentage 40% is less than 30% because the average length of 1000 query sequences is shorter in the mutation percentage 40%. The sequence comparison ratio in the mutation percentage 40% is also less than 30% because the query sequences are more similar to the sequences in DB.

5. Conclusions

When the edit distances of all sequence-pair in the database are known, this paper proposes an efficient searching strategy based on the triangle inequality property of edit distance for solving the MSS problem in a character-sequence database. The concept of the pruning step is drawing a circle of a double radius to identify the candidate sequences which should be compared with the query sequence. We present the strategy to select the next compared sequence for accelerating the searching speed. In the experiments, the searching strategy $\langle 0.81, 1\rangle$ works superiorly when the mutation percentages are small. That is, it can be applied more effectively if the query sequence is similar to a sequence in the database.

References

[1] E. J. Keogh and M. J. Pazzani, "Derivative dynamic time warping," in *Proceedings of the First Society for Industrial and Applied Mathematics International Conference on Data Mining*, vol. 1, (Chicago, IL, USA), pp. 5–7, 2001.

[2] E. Keogh and C. A. Ratanamahatana, "Exact indexing of dynamic time warping," *Knowledge and Information Systems*, vol. 7, no. 3, pp. 358–386, 2005.

[3] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[4] D. S. Hirschberg, "A linear space algorithm for computing maximal common subsequences," *Communications of the ACM*, vol. 18, no. 6, pp. 341–343, 1975.

[5] D. S. Hirschberg, "Algorithms for the longest common subsequence problem," *Journal of the ACM*, vol. 24, no. 4, pp. 664–675, 1977.

[6] W. J. Masek and M. S. Pateson, "A faster algorithm computing string edit distances," *Journal of Computer and System Sciences*, vol. 20, no. 1, pp. 18–31, 1980.

[7] R. Wagner and M. Fischer, "The string-to-string correction problem," *Journal of the ACM*, vol. 21, no. 1, pp. 168–173, 1974.

[8] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 2, pp. 403–410, 1990.

[9] S. McGinnis and T. L. Madden, "BLAST: at the core of a powerful and diverse set of sequence analysis tool," *Nucleic Acids Research*, vol. 32, pp. W20–W25, 2004.

[10] D. G. Higgins and P. M. Sharp, "CLUSTAL: a package for performing multiple sequence alignment on a microcomputer," *Gene*, vol. 73, no. 1, pp. 237–244, 1988.

[11] F. Corpet, "Multiple sequence alignment with hierarchical clustering," *Nucleic Acids Research*, vol. 16, no. 22, pp. 10881–10890, 1988.

[12] M. A. Larkin, G. Blackshields, N. P. Brown, R. Chenna, P. A. McGettigan, H. McWilliam, F. Valentin, I. M. Wallace, A. Wilm, R. Lopez, J. D. Thompson, T. J. Gibson, and D. G. Higgins, "Clustal W and Clustal X version 2.0," *Bioinformatics*, vol. 23, no. 21, pp. 2947–2948, 2007.

[13] S. Park, W. W. Chu, J. Yoon, and C. Hsu, "Efficient searches for similar subsequences of different lengths in sequence databases," in *Proceedings of 16th International Conference on Data Engineering*, (San Diego, USA), pp. 23–32, 2000.

[14] A. Aggarwal, L. J. Guibas, J. Saxe, and P. W. Shor, "A linear-time algorithm for computing the voronoi diagram of a convex polygon," *Discrete & Computational Geometry*, vol. 4, no. 6, pp. 591–604, 1989.

[15] L. Chen and R. Ng, "On the marriage of Lp-norms and edit distance," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases*, (Toronto, Canada), pp. 792–803, 2004.

[16] M. S. Waterman, T. F. Smith, and W. A. Beyer, "Some biological sequence metrics," *Advances in Mathematics*, vol. 20, no. 3, pp. 367–387, 1976.

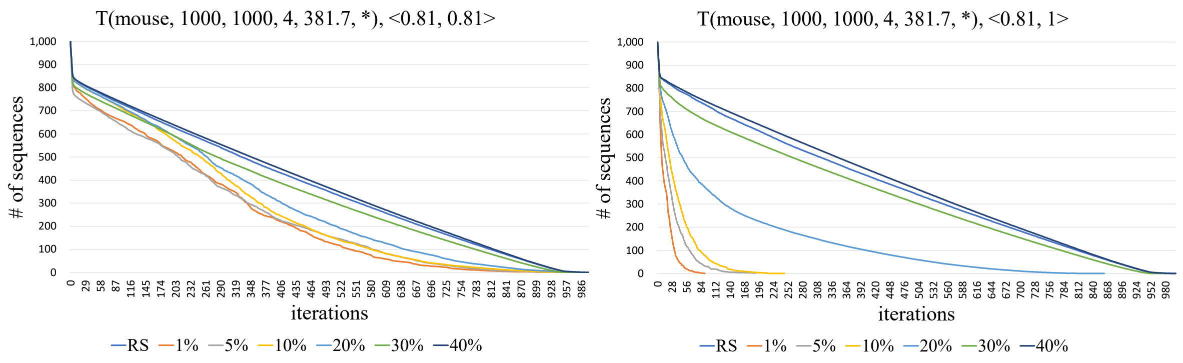
[17] H.-C. Hung, "A prune-and-search algorithm for finding the most similar sequence," Master's Thesis, Department of Computer Science and Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan, 2018.

[18] R. J. Hathaway, J. C. Bezdek, and Y. K. Hu, "Generalized fuzzy c-means clustering strategies using Lp norm distances," *IEEE Transactions on Fuzzy Systems*, vol. 8, no. 5, pp. 576–582, 2000.

Table 3 The execution time for the EST of mouse, where $num_{DB} = 1000$, $num_Q = 1000$, $|\Sigma| = 4$, $avg\_len(DB) = 381.7$ and the length scope of each Q is variable.

T(mouse, 1000, 1000, 4, 381.7, *)

		Seq	Our method						
			$\langle 0.81, 0.81\rangle$	$\langle 1, 0.81\rangle$	$\langle 0.7, 1\rangle$	$\langle 0.81, 1\rangle$	$\langle 0.9, 1\rangle$	$\langle 1, 1\rangle$	$\langle 1.2, 1\rangle$
RS		1 (100%)	0.8 (79.6%)	0.81 (79.4%)	0.78 (78.1%)	0.78 (78.1%)	0.78 (78.2%)	0.81 (78.2%)	0.78 (78.2%)
M	1%	0.92 (100%)	0.28 (35.8%)	0.15 (20.2%)	0.02 (1.7%)	0.02 (1.7%)	0.02 (1.8%)	0.02 (1.7%)	0.02 (1.8%)
	5%	0.93 (100%)	0.33 (38%)	0.23 (28.1%)	0.03 (2.9%)	0.03 (2.9%)	0.03 (3.1%)	0.03 (3.2%)	0.03 (3.3%)
	10%	0.98 (100%)	0.37 (41.2%)	0.27 (32.8%)	0.04 (4%)	0.04 (4%)	0.04 (4.1%)	0.04 (4.5%)	0.05 (4.9%)
	20%	1.04 (100%)	0.53 (50.4%)	0.51 (47.8%)	0.43 (38.1%)	0.43 (38%)	0.44 (38.3%)	0.45 (38.7%)	0.44 (38.8%)
	30%	0.93 (100%)	0.7 (74.4%)	0.7 (74.3%)	0.71 (74.3%)	0.7 (74.3%)	0.69 (74.3%)	0.7 (74.3%)	0.69 (74.3%)
	40%	1.03 (100%)	0.87 (83.1%)	0.88 (83.1%)	0.86 (83.1%)	0.86 (83.1%)	0.88 (83.1%)	0.86 (83.1%)	0.88 (83.1%)



(a) $\langle 0.81, 0.81\rangle$ **(b)** $\langle 0.81, 1\rangle$

Figure 3 The average number of sequences remained in each iteration of the searching strategies $\langle 0.81, 0.81\rangle$ and $\langle 0.81, 1\rangle$ in Table 3.

Table 4 The execution time for the EST of mouse, where $num_{DB} = 5000$, $num_Q = 1000$, $|\Sigma| = 4$, $avg\_len(DB) = 383.7$ and length scope of each Q is variable.

T(mouse, 5000, 1000, 4, 383.7, *)

		Seq	Our method						
			$\langle 0.81, 0.81\rangle$	$\langle 1, 0.81\rangle$	$\langle 0.7, 1\rangle$	$\langle 0.81, 1\rangle$	$\langle 0.9, 1\rangle$	$\langle 1, 1\rangle$	$\langle 1.2, 1\rangle$
RS		4.9 (100%)	3.67 (73.7%)	3.67 (73.7%)	3.61 (71.8%)	3.61 (71.9%)	3.62 (71.9%)	3.63 (72.2%)	3.62 (72.1%)
M	1%	4.69 (100%)	1.42 (32.7%)	1.1 (27.8%)	0.07 (1.3%)	0.07 (1.3%)	0.07 (1.5%)	0.07 (1.5%)	0.07 (1.6%)
	5%	4.67 (100%)	1.48 (34.1%)	1.37 (32.4%)	0.16 (2.8%)	0.14 (2.7%)	0.16 (3%)	0.17 (3.3%)	0.18 (3.6%)
	10%	4.56 (100%)	1.51 (34%)	1.38 (31.8%)	0.16 (3.1%)	0.15 (3%)	0.17 (3.3%)	0.19 (4.1%)	0.2 (4.1%)
	20%	4.55 (100%)	1.93 (43.2%)	1.92 (42.9%)	1.69 (34.3%)	1.62 (34.2%)	1.67 (34.5%)	1.67 (35.1%)	1.73 (35.3%)
	30%	5.4 (100%)	4.45 (78.4%)	4.45 (78.4%)	4.55 (78.4%)	4.4 (78.4%)	4.47 (78.4%)	4.44 (78.4%)	4.51 (78.4%)
	40%	4.14 (100%)	3.21 (72.1%)	3.2 (72.1%)	3.27 (72.1%)	3.28 (72.1%)	3.27 (72.1%)	3.2 (72.1%)	3.21 (72.1%)