A Hybrid Algorithm for the Longest Common Subsequence Problem

Hsiang-Yi Weng[a], Shyue-Horng Shiau[b], Kuo-Si Huang[c] and Chang-Biau Yang[a] *
[a]Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan 80424

[b]Department of Computer Aided Media Design
Chang Jung Christian University, Tainan, Taiwan

[c]Department of Information Management
Shu Zen College of Medicine and Management, Kaohsiung, Taiwan

Abstract

The k-LCS (longest common subsequence) problem is to find the LCS of k sequences. The k-LCS is difficult while the length and the number of sequences are large. For solving this problem, in this paper, we design a hybrid method, which is a combination of a heuristic method, GA (genetic algorithm) and ACO (ant colony optimization) algorithm. In our experiments, we compare our method with expansion algorithm, best next for maximal available symbol algorithm, GA and ACO algorithm. The experimental results on several sets of DNA sequences show that our method outperforms other algorithms in the length of solution.

1 Introduction

The k-LCS problem is to find the *longest common subsequence* (LCS) of k input sequences. Suppose we are given a set S of input sequences, where $S = \{s_1, s_2, ..., s_k\}$. If c is a subsequence of all s_i for $1 \leq i \leq k$ simultaneously, then c is said to be a *common subsequence* (CS) of S. For example, consider the set $S = \{s_1 = \texttt{ATCGTAC}, s_2 = \texttt{CTGTAGC}, s_3 = \texttt{GTTCATC}\}$. The sequences AC, CTC and TTAC are common subsequences of s_1, s_2 and s_3. If c is the longest among all of the common subsequences of S, then c is called the LCS of S. The long history of the LCS problem is well-known in computer science and bioinformatics [4, 5, 8, 13]. The common subsequences show a lot of relationships between input sequences, such as alignment, motif or conserved region of these sequences [6, 9]. If the size of S is an arbitrary number, k, where $k > 2$, the k-LCS problem is an NP-hard problem even over binary alphabet [10].

It is hard to find out all CSs of k sequences by exhaustive search and then to get the LCS from these CSs while k is large. For improving its efficiency, we select some good features of LCS to eliminate bad sequences in CSs. Some heuristic algorithms have been proposed for solving this problem, in which some effective evaluation schemes are used to determine the goodness of CS.

The *expansion algorithm* (EA) is proposed by Bonizzoni *et al.* [1]. The compressed content of input sequences with *run-length encoding* (RLE) is used for finding the basic structure (stream) of CS and all symbols of the stream are expanded to obtain longer CS. However, the performance is not so good. Huang *et al.* propose a *best next for maximal available symbols* (BNMAS) algorithm [7] for determining highly possible symbols as elements of LCS based on the occurrence frequency of each common symbol in the input sequences. It is an intuitive method to select common characters from sequences. However, a bad common symbol may be chosen when the occurrence frequencies of common symbols are the same and it may cause the found common subsequence shorter.

Some heuristic methods with the divide-and-conquer strategy may be used to determine the possible common character positions in the sequences. The distribution of characters can be viewed as important information for choosing

*Corresponding author: cbyang@cse.nsysu.edu.tw which characters form the CS. It will derive a CS from the possible solutions for the k-LCS problem. Heuristic algorithms can easily apply their own constrained conditions but they may only find the local optimal results for this problem.

There are several evolutionary algorithms for breaking through the searching restrictions. The complete model of *ant colony optimization* (ACO) algorithm for the k-LCS problem was proposed by Shyu and Tsai [11]. This algorithm tries to find the CS in S by the characteristics of ants looking for food [3]. It relies on the concentration of pheromone to achieve adaptation and to get better results. The ACO algorithms can remember better solutions that have been found, decide which character is a best common character and append it to the CS.

Another evolutionary strategy, called *genetic algorithm* (GA), imitates the evolution of genome. Chiang *et al.* proposed the genetic algorithm for solving the k-LCS problem [2]. In the algorithm, some CSs of S are regarded as chromosomes and they are evolved by mutation and crossover operators for finding better CSs. GA has the ability to retain good parts of ancestors to find better descendants and to reserve good members to form a new generation for next evolution based on the fitness function [12].

The heuristic methods for this problem usually use the same tactic to select possible candidates of LCS from the possible CSs. One may design a greedy algorithm to obtain the LCS for the specific input sequences. But this greedy algorithm may not be applicable to find the LCS in other input sequences. In ACO, the concentration of pheromone affects the judgment of character selection. It may lead some specific characters are not included in the optimal solution. For GA, it is able to get better solution with a large generations of evolution and it requires a lot of time. In order to compensate for the finding ability of single method, we adopt a mixed strategy for this problem. We want to find a longer CS by the evolution capacity of GA and the intelligence of ACO algorithm. We can disrupt original searching limitation and overstride the local optimal solution.

In this paper, a new hybrid method is proposed for solving the k-LCS problem. Section 2 introduces several related algorithms for the k-LCS problem. In Section 3, our hybrid method is presented to find a longer CS of k sequences by using heuristic algorithms, GA and ACO algorithm. The experimental results are shown in Section 4. Finally, we give a conclusion in Section 5.

2 Previous Works

It is hard to find the LCS of k sequences while k is large. Several heuristic and evolutionary algorithms are proposed for finding longer CS of input sequences. The expansion algorithm [1], the BNMAS algorithm [7], the algorithms with ant colony optimization [11] and genetic algorithm approaches [2] are introduced in this section.

2.1 Expansion Algorithm and BNMAS Algorithm

Heuristic algorithms are proposed for finding acceptable solutions for the k-LCS problem efficiently. The complete model of the *expansion algorithm* (EA) [1] was proposed by Bonizzoni *et al.* This algorithm compresses each input sequence into its shortest possible stream, where a stream is a sequence that does not contain successively identical symbols. After finding the longest common stream R of all sequences, we can expand all substrings of R to obtain a longer CS of S.

For reducing the executing time of EA, an approximation algorithm that is named as BNMAS was proposed [7]. This algorithm counts the occurrence number of each individual common symbol, *available symbols*, before a specific location of each symbol (including itself) for each sequence in S. The total mixed number of available symbols is used to decide which symbol can be appended to a CS. After processing by the forward direction, we can find a CS of S. The algorithm can extend easily to find the better results by the two opposite directions, and it requires a little execution time. By taking the advantage of time-efficiency, EA and the BNMAS algorithm can be applied to generate CSs as initial solutions for evolutionary algorithms.

2.2 Ant Colony Optimization Algorithm for k-LCS

Shyu and Tsai [11] propose a method to find longer CS for the k-LCS problem by using the *ant colony optimization* (ACO) algorithm. Suppose there are m ants, $m < k$, each ant chooses one sequence s_i from S to extract a subsequence from s_i as a candidate of CS of S. Let $s_i[j]$ represent the jth character in sequence s_i where $1 \leq i \leq k$ and $1 \leq j \leq |s_i|$. For example, s_1 = `ATCGTAC` implies $s_1[3]$ = `C` and $s_1[5]$=`T`. Let $a_i^r[j]$ denote the extracting state of the jth character from sequence s_i extracted by ant a_r. For example, if the

$$p\ (v_r) = \begin{cases} 1 & \text{if } \ q \leq q_0 \text{ and } v_r = \arg\ \max_{z_r \in W} \left\{\tau(z_r)\eta(z_r)^{\beta}\right\} \text{ (exploitation)}, \\ \frac{\tau(v_r)\eta(v_r)^{\beta}}{\Sigma_{z_r \in W}\tau(z_r)\eta(z_r)^{\beta}} & \text{if } \ q_0 < q \leq q_1 \text{ (biased exploration)}, \\ 0 & \text{otherwise } \ q = q_1 \text{ (exclusion)}. \end{cases} \tag{1}$$

$$\eta(v_r) = \frac{1}{\Sigma_{i=1}^{k}(v_i - u_i)}. \tag{2}$$

$$\tau(t_r) = (1-\rho)\tau(t_r) + \rho\Delta\tau(t_r). \tag{3}$$

$$\Delta\tau(t_r) = \begin{cases} \frac{|c^+|}{M} & \text{if } t_r \text{ is an element of } c^+ \text{ by } a_r, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

array a_1^2 is equal to 0101001, then it means that ant a_2 extracts `TGC` from s_1. One can construct a common subsequence candidate (CSC) based on a_i^r. Ant a_r might find a better CSC of S according to residue of pheromone on each position of s_i for $1 \leq i \leq k$. Let $c[i,j]$ denote a substring of c from indexes i to j. We can use a position vector, $V_r^l = \langle v_1^l, v_2^l, \cdots, \check{v}_i^l, \cdots, v_j^l, \cdots, v_k^l\rangle$, for $0 \leq l \leq |c|$, and $1 \leq r \leq m$, to keep track of the states, where $|c|$, $\check{v}_i^l$ and v_j^l denote the length of CS c found by a_r, the location index found by a_r in s_i and the minimal location index based on $\check{v}_i^l$ such that $c[0,l]$ is a subsequence of $s_j[0, v_j^l]$ for $j \neq i$, respectively. The initial position of all ants is 0 and V_r^0=$\langle 0, 0, \cdots, v_i^0, 0, \cdots, 0\rangle$ in the beginning for $1 \leq r \leq m$. For example, consider three sequences s_1 = `ATCGTAC`, s_2 = `CTGTAGC` and s_3 = `GTTCATC`. Suppose ant a_1 chooses characters 2, 5, 6 and 7 in s_1 as a CSC. The CS c = `TTAC` implies position vector as follows.

$$\begin{aligned} V_1^0 &= \langle 0,0,0\rangle, \\ V_1^1 &= \langle 2,2,2\rangle, \\ V_1^2 &= \langle 5,4,3\rangle, \\ V_1^3 &= \langle 6,5,5\rangle, \\ V_1^4 &= \langle 7,7,7\rangle. \end{aligned}$$

Based on Equation (1), an ant can decide which character would be selected as a common character or skip all elements in a window W of size d in sequence s_r chosen by a_r.

In Equation (1), $\tau(v_r)$ is the concentration of pheromone at $v_r \in W = [u_r + 1, min(u_r + d, |s_r|)]$, where $0 \leq u_r \leq |s_r| - 1$; $\eta(v_r)$ is a cost of v_r defined by function (2); β is a relational parameter between $\tau(v_r)$ and $\eta(v_r)$; q is a random number, $0 \leq q \leq 1$; q_0 and q_1 are used to decide which searching strategy (exploitation, biased exploration, exclusion) is favored, where $1 \leq q_0 < q_1 \leq 1$. If $p(v_r) = 1$, a_r chooses $v_r = \arg\ \max_{z_r \in W}\left\{\tau(z_r)\eta(z_r)^{\beta}\right\}$ as a common character. If $p(v_r) = 0$, a_r will discard all elements of W; otherwise, a_r chooses a character with its probability according to Equation (1). When all ants complete the tasks of finding CSs of S, the pheromone will be updated on all positions of the input sequences by Equation (3). ρ is evaporation coefficient for ant pheromone in nature. When the selected times of the character is reduced, its pheromone evaporated according to Equation (3). In Equation (4), M is the length of LCS of a random selected pair sequences (s_i, s_j) in S; c^+ is the best CS of S found by m ants in a round. After several rounds of implementation, we could get the best results upon these states.

2.3 Genetic Algorithm for k-LCS

Chiang *et al.* [2] develop a genetic algorithm (GA) for solving the k-LCS problem. It chooses the shortest sequence in S, or the last input sequence if all of them have the same length, which is regarded as the *template sequence*. A *template pattern* is defined as a binary sequence of 0 or 1 corresponding to the template sequence. For example, suppose a template sequence T_1=$\underline{\mathtt{G}}\mathtt{TA}\underline{\mathtt{CT}}\mathtt{G}\underline{\mathtt{A}}\mathtt{TAC}\underline{\mathtt{TGT}}$ and a template pattern p_1=`1001101000111`, it implies the template subsequence s_{p_1}=`GCTATCT`. We can change the content of the template pattern by GA to construct possible CSs. Let $s_{p_j}^m$ and $s_{p_j}^v$ are the number of the input sequences which fully contain subsequence s_{p_j} in S and the sum of symbols that s_{p_j} is greedily matched to all input sequence, respectively. The fitness function of GA, Equation (5), is used to identify whether the template pattern is good.

$$f(s_{p_j}) = \begin{cases} s_{p_j}^m \times s_{p_j}^v & \text{if } s_{p_j}^m = |S|, \\ -1 \times (|S| - s_{p_j}^m) \times s_{p_j}^v & \text{otherwise.} \end{cases} \tag{5}$$

If the template pattern p_j causes a lot of matching bits in the input sequences, we will presume that it is a good pattern for the CS. The score $f(s_{p_j})$ of a good pattern is higher than others. By repeatedly performing procedures of GA, we can get a better result for this problem.

3 Our Hybrid Algorithm

By taking the advantages of GA and ACO approaches, our algorithm for solving the k-LCS problem is a circulating feedback method and it combines two evolutionary methods (GA and ACO algorithm) and hursitic algorithms. Our algorithm is given as follows.

The hybrid algorithm for the k-LCS problem

Input: A set S contains k sequences over a fixed alphabet Σ.

Output: A common subsequence (CS) of S.

Step 1: Find a CS c of S by the heuristic method.

Step 2: Take c as the initial CS of ACO and continue to look for a new CS S_{ACO} by ACO.

Step 3: Take S_{ACO} as the initial CS of GA and keep on evolving a new CS S_{GA} by GA.

Step 4: Let $c = S_{GA}$. Repeat Steps 2 and 3 until the predefined number of iterations is reached.

Step 5: Return c.

Our algorithm discovers a preliminary result quickly and treats this result as the initial solution of the other methods. Let a possible common subsequence c=$c[1]c[2]\cdots c[j]$, the jth symbol $c[j]$ is the last symbol in c. Consider the last symbol $c[j]$ in c, let u_i be the minimal index of s_i such that $c[0,j]$ is a subsequence of $s_i[0,u_i]$. Let $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_{|\Sigma|}\}$ be the alphabet of this problem. For any candidate symbol σ_x, $1 \leq x \leq |\Sigma|$, let $v_i(\sigma_x)$ be the minimal index of s_i such that $c[0,j]\sigma_x$ is a subsequence of $s_i[0, v_i(\sigma_x)]$. If $c[0,j]\sigma_x$ is not a subsequence of s_i, we set $v_i(\sigma_x)$ $= \infty$ and σ_x will be never selected into the possible CS. For determining which character σ_x is a good common symbol of S, we calculate $\delta(\sigma_x)$ and $\Omega(\sigma_x)$ by Equations (6)and (7).

$$\delta(\sigma_x) = \begin{cases} \Sigma_{i=1}^{k}(v_i(\sigma_x) - 1) & \text{if } |c| = 0, \\ \Sigma_{i=1}^{k}(v_i(\sigma_x) - u_i - 1) & \text{if } |c| > 0. \end{cases} \quad (6)$$

$$\Omega(\sigma_x) = \begin{cases} \max_{i=1}^{k}\{v_i(\sigma_x) - 1\} & \text{if } |c| = 0, \\ \max_{i=1}^{k}\{v_i(\sigma_x) - u_i - 1\} & \text{if } |c| > 0. \end{cases} \quad (7)$$

For a possible CS c, it is easy to check that $s_1[v_1(\sigma_x)] = s_2[v_2(\sigma_x)] = \cdots = s_k[v_k(\sigma_x)] = \sigma_x$, $s_1[u_1] = s_2[u_2] = \cdots = s_k[u_k] = c[j]$, and $s_i[j] \neq \sigma_x$ where $u_i < j < v_i(\sigma_x)$ for $1 \leq i \leq k$. Consider that symbol σ_x is a candidate character for joining c. If $\delta(\sigma_x)$ is minimal, σ_x is selected as a common character as $c[j+1]$. If $\delta(\sigma_x)$ and $\delta(\sigma_y)$ are minimal and $\delta(\sigma_x) = \delta(\sigma_y)$ for $1 \leq x \neq y \leq |\Sigma|$, σ_x is selected as a common character $c[j+1]$, while $\Omega(\sigma_x) \leq \Omega(\sigma_y)$. After a new common character σ_x is selected, we ignore those characters in front of the location of σ_x (including itself) and then perform the next round of selecting a new common character. For example, consider three sequences s_1 = `ATCGTAC`, s_2 = `CTGTAGC` and s_3 = `GTTCATC`. The character relationships among sequences in S are given as follows.

$$\begin{aligned} s_1[1] = s_2[5] = s_3[5] = \texttt{A}. \\ s_1[2] = s_2[2] = s_3[2] = \texttt{T}. \\ s_1[3] = s_2[1] = s_3[4] = \texttt{C}. \\ s_1[4] = s_2[3] = s_3[1] = \texttt{G}. \end{aligned}$$

Based on the character relationships, it summaries that there are 8, 3, 5, and 5 characters, accumulated from each sequence, in front of symbols `A`, `T`, `C`, and `G`, respectively. Because symbol `T` has the minimal summation of characters in front of it, we choose symbol `T` as the common symbol of S and let c_1 = `T`. Then update starting positions $Start(s_1) = 3$, $Start(s_2) = 3$ and $Start(s_3) = 3$.

Now, the character relationships among sequences based on starting positions are updated as follows.

$$\begin{aligned} s_1[6] = s_2[5] = s_3[5] = \texttt{A}. \\ s_1[5] = s_2[4] = s_3[3] = \texttt{T}. \\ s_1[3] = s_2[7] = s_3[4] = \texttt{C}. \\ s_1[4] = s_2[3] = s_3[\infty] = \texttt{G}. \end{aligned}$$

It summaries that there are 7, 3, 5, and ∞ characters, accumulated from each sequence, in front of symbols `A`, `T`, `C`, and `G`, respectively. Because symbol `G` is no longer available in s_3, we define the index of symbol `G` in s_3 as ∞. Symbol `T` has the minimal summation of characters in front of it, hence we choose c_2 = `T` and update starting positions $Start(s_1) = 6$, $Start(s_2) = 5$ and $Start(s_3) = 4$. With the above procedure, we can find a CS `TTAC`.

In order to find longer CS, we take the CS found by the heuristic algorithm as the initial solution of the ACO algorithm to improve the solution for this problem. The ACO algorithm searches possible characters in the sequences of S by the way of ants searching for their food. Suppose there are three ants and each ant independently take charge of one sequence in $S = \{s_1 = \texttt{ATCGTAC}, s_2 = \texttt{CTGTAGC}, s_3 = \texttt{GTTCATC}\}$. These ants want to find possible common symbols of S by considering the pheromone of each symbol in $s_i[j_1, j_2]$, where $1 \leq j_1 \leq j_2 \leq |s_i|$ and $1 \leq i \leq k$. The pheromone of each symbol in each sequence is set as 1 initially and it is updated when all of these ants finish their searching in one round. Let w_i and $ph(w_i)$ be the possible common symbol in $s_i[j_1, j_2]$ and the pheromone of w_i, respectively. The selecting probability $P(w_i)$ is defined in Equation (8). We will choose the symbol with $\max\{P(w_i)|j_1 \leq i \leq j_2\}$ as the common character of S.

$$P(w_i) = (\frac{1}{\delta(w_i) + k})^2 * ph(w_i) \tag{8}$$

Suppose that ant a_1 searches common characters in s_1 and $(j_1,\ j_2) = (1,\ 4)$. Consider symbol A, $s_1[1] = s_2[5] = s_3[5] = \texttt{A}$. According to Equation (8), the selecting probability of the symbol A is $P(A_{1,5,5}) = (\frac{1}{(0+4+4)+3})^2 * ph(A_{1,5,5}) = \frac{1}{121}$. Based on the same formula, we can get $P(T_{2,2,2}) = \frac{1}{36}$, $P(C_{3,1,4}) = \frac{1}{64}$ and $P(G_{4,3,1}) = \frac{1}{64}$. We will choose the symbol T as a common character of S and set $(j_1,\ j_2) = (3,\ 6)$. If $\delta(w_i) = \infty$, it represents that w_i is not a common symbol in $s_i[j_1, j_2]$ and the selecting probability of symbol w_i will not be calculated. With this way based on the ACO algorithm, we can find a common subsequence TTAC of S.

Using the same evolutionary algorithm to construct the common subsequence, it is possible to get the same local optimal solution. In order to break the dilemma, we should rely on other algorithms to change the local optimal solution and to get a longer CS. Hence, the result of the ACO algorithm is taken as the initial solution of GA in our hybrid method so that GA can find longer CS. With the same principle, we can use the solution of GA as the input solution of ACO to increase the length of the CS.

In our hybrid method, GA can choose possible common characters to construct a common subsequence by its mutation and crossover operators. We will select a sequence T, the shortest sequence or the last sequence if the lengths of all input sequences are the same in S, at the beginning of GA as the template sequence. We randomly select some characters in this template sequence at different positions and check whether these charac-

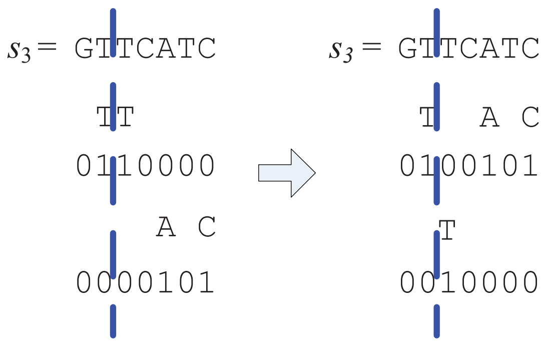

Figure 1: An example of crossover operator of GA.

ters form a common subsequence. The template subsequences will be sorted according to the fitness scores defined by Equation (5). We take out better template subsequences with the fixed proportion to change their composition through mutation and crossover operators.

The mutation operator of GA is defined as follows. Suppose a set $S = \{s_1 = \texttt{ATCGTAC}, s_2 = \texttt{CTGTAGC}, s_3 = \texttt{GTTCATC}\}$, and s_3 is taken as the template sequence. We can find a subsequence AC in s_3. We select a character G from s_3 and join the AC in G according to the order of sequence. We can examine whether c=GAC is a CS of S easily. A legitimate common character is an character in CS. Because c=GAC is a CS of S, the sum of the number of all legitimate characters in S is $\Sigma_{i=1}^{3}|c| = 3 + 3 + 3 = 9$, where $|c|$ is the length of c. By Equation (5), we obtain the fitness score of c is $f(c) = 3 * 9 = 27$.

Figure 1 shows an example of crossover operator of GA. There are two subsequences TT and AC in s_3 =GTTCATC and they can convert into two template patterns P_1 = 0110000 and P_2 = 0000101. We randomly select a crossover point, a position in s_3, to swap template patterns. For example, we choose the third bit as the crossover point to swap P_1 = $01\underline{10000}$ and P_2 = $00\underline{00101}$. After P_1 and P_2 are swapped at the crossover point, we can get two new template patterns P_3 = $01\underline{00101}$ and P_4 = $00\underline{10000}$, whose corresponding subsequences are TAC and T, respectively.

With our experimental experience, to find a longer CS with only one method may not be efficient, because the only one algorithm may find out local optimal solutions easily. It is difficult to break through the limitations of a specific evolutionary algorithm. Hence we propose a hybrid method that combines our heuristic method, GA

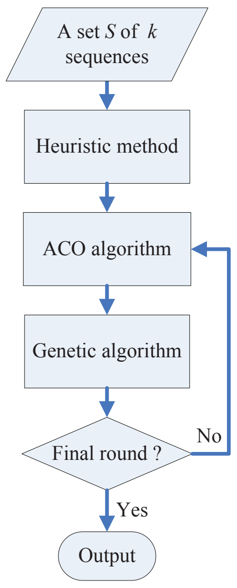

Figure 2: The flowchart of our hybrid algorithm.

and ACO algorithm to find a longer CS. The probability of obtaining longer CS will increase by combining various evolutional characteristics of GA and ACO algorithm. GA and ACO algorithm have different contributions in the habit of looking for CSs.

In fact, the CS found by our algorithm consists of several parts that are selected from GA and ACO interlacedly. The hybrid algorithm helps that the CS becomes longer gradually and this searching mode overcomes the limitation of a single algorithm. In our experiments, we alternately use ACO and GA to find better CS. We shows the flowchart of our hybrid algorithm in Figure 2.

4 Experimental Results

In general, artificial input sequences can be generated by a computer program with some generating rules. If one can guess generating rules from the input data, he may design an elegant algorithm to deal with these input data and to output excellent results. For preventing such exceptional guess, we use biological sequences as the input data in our experiments. The biological sequences in our experiment are provided by Shyu and Tsai [11]. We take the first 600 characters of each input sequence as the test data. Our program is implemented with C++ on a personal computer which runs on operating system platform Linux 2.6.27-gentoo-r7 with an AMD Athlon(tm) 64 X2 Dual Core Processor 3800+ (CPU) and 2GB RAM.

We find that resetting the pheromone can effectively help ACO algorithm get a breakthrough for finding longer solution and giving GA an initial CS can make it get better results in the experiment, so we reset the pheromone value at the beginning stage of ACO every time. We regard the length of CS found by various methods as the comparison standard. If the length of CS is longer obtained by method A, then method A is measured to be a better method for this problem. We use the same parameter setting for ACO and GA in our hybrid algorithm by Shyu and Tsai [11] and Chiang *et al.* [2], respectively.

Compared with algorithms EA (Expansion Algorithm), B (BNMAS), GA, ACO1, and ACO2, our method (HA) is better than others. In our experiments, ACO1 and ACO2 terminate if the solution has not been improved for 2000 and 100 generations, respectively. Tables 1, 2 and 3 show output lengths of CSs obtained by various algorithms for randomly selected DNA sequences, rat DNA sequences, and virus DNA sequences, respectively. Tables 4, 5 and 6 show the computer execution time for randomly selected DNA sequences, rat DNA sequences, and virus DNA sequences, respectively. Note that the alphabet size of DNA sequences is four. The length of CS obtained by each algorithm is the average of ten independent simulations. In these tables, our hybrid method (HA) finds out the longest CS than others. Especially, our method gets large improvement in rat DAN sequences and virus DNA sequences.

5 Conclusion

The longest common subsequence problem has been studied for several decades. It is a well-known and classical problem in computer science and computational biology. If the number of the

Table 1: The CS lengths for random DNA sequences.

k	HA	ACO1	ACO2	GA	B	EA
10	201.0	200.2	197.2	182.8	169	173
15	188.8	187.7	185.2	173.4	157	164
20	178.6	178.0	176.2	165.7	157	156
25	175.0	173.9	172.2	163.8	154	155
40	164.2	162.8	161.4	154.5	150	148
60	157.7	156.7	155.4	149.0	146	145
80	154.3	153.5	151.6	145.5	146	141
100	151.5	150.6	148.8	144.0	141	141
150	145.6	145.2	143.4	140.4	139	134
200	143.7	143.7	141.0	139.1	137	133

Table 2: The CS lengths for rat DNA sequences.

k	HA	ACO1	ACO2	GA	B	EA
10	185.0	182.3	182.0	170.4	163	160
15	170.3	168.3	166.6	155.7	149	148
20	160.4	160.0	160.0	149.2	149	146
25	160.0	157.4	155.8	141.1	145	144
40	146.2	145.6	143.4	134.0	135	136
60	144.5	143.6	142.4	134.1	131	129
80	132.2	128.8	128.8	123.5	121	116
100	127.8	124.7	124.6	121.4	118	113
150	121.1	116.2	115.6	114.4	109	100
200	119.2	114.6	114.6	111.8	108	101

Table 3: The CS lengths for virus DNA sequences.

k	HA	ACO1	ACO2	GA	B	EA
10	201.2	199.5	197.6	186.2	177	170
15	186.1	184.9	183.6	171.1	168	162
20	177.3	174.0	173.8	159.8	157	151
25	181.6	179.1	179.0	163.0	172	158
40	158.3	155.5	155.0	143.0	147	146
60	154.5	151.4	150.6	145.7	146	141
80	149.9	146.3	145.8	138.7	143	136
100	146.5	143.4	143.4	137.1	141	136
150	145.8	142.8	141.6	138.8	136	132
200	144.7	141.1	140.6	136.8	135	133

Table 4: Execution time (sec) for random DNA sequences.

k	HA	ACO1	ACO2	GA	B	EA
10	118.03	152.18	12.55	337.59	0.03	132.85
15	179.10	247.65	25.82	389.06	0.06	99.06
20	239.74	339.42	28.56	474.30	0.09	67.03
25	310.57	495.99	39.46	638.06	0.12	70.72
40	878.35	910.69	63.84	962.83	0.18	105.56
60	1567.94	1719.77	95.51	1799.92	0.26	274.19
80	2030.95	2039.71	110.40	2272.96	0.39	630.58
100	2578.12	2808.88	172.36	2521.96	0.51	1193.59
150	3761.38	4488.62	242.31	3858.98	0.71	3930.63
200	5090.58	6420.47	329.32	5065.11	1.00	9272.10

Table 5: Execution time (sec) for rat DNA sequences.

k	HA	ACO1	ACO2	GA	B	EA
10	108.51	118.75	10.88	272.50	1.01	71.47
15	157.93	237.76	22.39	341.77	1.03	48.63
20	213.13	373.96	28.41	444.12	1.06	29.15
25	289.82	494.22	39.53	551.18	1.11	36.39
40	854.53	620.46	50.02	875.21	1.15	85.56
60	1551.27	1714.13	102.16	1460.81	1.23	252.88
80	1895.49	2054.37	122.38	2120.88	1.34	564.89
100	2400.85	2635.22	134.81	2581.05	1.46	1082.21
150	3490.43	3713.02	213.27	3526.89	1.67	3579.92
200	4255.85	4638.01	303.12	4857.34	1.92	8499.41

Table 6: Execution time (sec) for virus DNA sequences.

k	HA	ACO1	ACO2	GA	B	EA
10	126.22	158.90	17.28	322.56	1.95	136.95
15	177.59	276.80	26.69	349.31	1.96	81.92
20	237.90	394.29	29.96	406.17	2.00	58.42
25	343.48	544.82	41.84	455.81	2.04	65.02
40	979.38	1033.64	77.36	836.79	2.11	141.98
60	1826.11	1873.22	94.53	1490.48	2.18	279.87
80	2169.38	2339.66	120.19	2104.48	2.29	608.88
100	2639.82	2776.33	159.51	2523.03	2.43	1185.56
150	4009.22	4459.56	256.90	3617.01	2.64	4254.76
200	5229.24	6170.26	345.60	4711.05	2.90	10233.00

input sequences is not a fixed constant, it will become an NP-hard problem. Several methods and approximate algorithms have been proposed to find longer CSs of k sequences, but some of these algorithms are not efficient. In this paper, we propose a hybrid algorithm, combining a heuristic method, GA and ACO algorithm, for solving the k-LCS problem. We get the longer CS than other algorithms in the experiments. In the future, we may try to shorten the execution time of our method and improve the effectiveness of our program. We can consider our method in combination with more algorithms to form better algorithms for the k-LCS problem.

References

[1] P. Bonizzoni, G. D. Vedova, and G. Mauri, "Experimenting an approximation algorithm for the LCS," *Discrete Applied Mathematics*, Vol. 110, No. 1, pp. 13–24, 2001.

[2] C.-H. Chiang, C.-B. Yang, and S.-H. Shiau, "A genetic algorithm for the longest common subsequence of multiple sequences," *Master Thesis, Department of Computer Science and Engineering National Sun Yat-sen University*, 2009.

[3] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, Vol. 26, No. 1, pp. 29–41, 1996.

[4] D. S. Hirschberg, "A linear space algorithm for computing maximal common subsequence," *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 ACM*, Vol. 24, pp. 664–675, 1977.

[6] K.-F. Huang, C.-B. Yang, and K.-T. Tseng, "An efficient algorithm for multiple sequence alignment," *Proc. of the 19th Workshop on Combinatorial Mathematics and Computation Theory*, Kaohsiung, Taiwan, pp. 50–59, 2002.

[7] K.-S. Huang, C.-B. Yang, and K.-T. Tseng, "Fast algorithms for finding the common subsequence of multiple sequences," *Proceeings of International Computer Symposium*, Taipei, Taiwan, pp. 90–95, 2004.

[8] J. W. Hunt and T. G. Szymanski, "A fast algorithm for computing longest common subsequences," *Communications of the ACM*, Vol. 20, No. 5, pp. 350–353, 1977.

[9] R. C. T. Lee, R. C. Chang, S. S. Tseng, and Y. T. Tsai, *Introduction to the Design and Analysis of Algorithm - a strategic approach.* ISBN 007-124346-1, McGraw Hill, 2005.

[10] D. Maier, "The complexity of some problems on subsequences and supersequences," *Journal of the ACM*, Vol. 25, No. 2, pp. 322–336, 1978.

[11] S.-J. Shyu and C.-Y. Tsai, "Finding the longest common subsequence for multiple biological sequences by ant colony optimization," *Computers & Operations Research*, Vol. 36, pp. 73–91, 2009.

[12] R. E. Smith, B. A. Dike, and S. A. Stegmann, "Fitness inheritance in genetic algorithms," *Proceedings of the 1995 ACM symposium on Applied computing*, Nashville, TN, USA, pp. 345–350, 1995.

[13] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *Journal of the ACM*, Vol. 21, No. 1, pp. 168–173, 1974.