An Enhanced ACO Algorithm with Pair Matching Strategy for the Longest Common Subsequence Problem

Hsiang-Yi Weng[a], Shyue-Horng Shiau[b], Chang-Biau Yang[a]*,
Yung-Hsing Peng[a] and Kuo-Si Huang[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

Abstract

The longest common subsequence (LCS) can indicate globally identical relationship among input sequences. The k-LCS problem tries to find the LCS of k sequences, and it becomes difficult if the length and the number of sequences are large. This paper adopts the pair matching strategy with ant colony optimization (ACO) algorithm for improving the performance of finding LCS. It requires less computational time than the hybrid algorithm of genetic algorithm (GA) and ACO algorithm. In the experiments, our method has better performance than other algorithms, including expansion algorithm, best next for maximal available symbol algorithm, GA, ACO algorithm and the hybrid algorithm of GA and ACO algorithm.
Keywords: longest common subsequence, multiple sequences, ant colony optimization, pair matching, bioinformatics

1 Introduction

The *longest common subsequence* (LCS) problem is well-known in computer science and bioinformatics [4, 5, 8, 13]. In bioinformatics, the common subsequences show specific relationships among sequences, such as alignment, motif or conserved region [6, 10]. The k-LCS problem tries to find the LCS of k input sequences. Given a set $S = \{s_1, s_2, ..., s_k\}$ of k input sequences, s_c is regarded as a *common subsequence* (CS) of S if s_c is a subsequence of each $s_i \in S$. Sequence s_c is called the LCS of S, if s_c is the longest one among all of the common subsequences of S. If the size $k = |S|$ is not fixed, the k-LCS problem is an NP-hard problem [11].

For finding the LCS, it is infeasible to find out the longest sequence over all possible CSs of large set of input sequences. One may apply the dynamic programming strategy for the k-LCS problem, but it requires $O(n^k)$ time and space where n and k are the length and number of input sequences, respectively. For improving the time and space complexities of dynamic programming strategy, some algorithms [3, 9] were developed but they are not so practical. Therefore, some heuristics have been proposed for the k-LCS problem, such as *expansion algorithm* (EA) [1], *best next for maximal available symbols* (BNMAS) [7], *ant colony optimization* (ACO) algorithm [12], *genetic algorithm* (GA) [2], and the hybrid algorithm combined with GA and ACO (HGACO) [14].

Algorithm HGACO is a hybrid algorithm, combining a heuristic method, GA and ACO algorithm, for solving the k-LCS problem. HGAGO is a circulating feedback method by taking the advantages of GA and ACO approach. In ACO, the concentration of pheromone affects the judgment of character selection. It may cause some specific characters not included in the final solution. GA can get better solution with tedious generations of evolution, but it requires a lot of time. For reducing the required time of HGAGO, this paper adopts the matching pair algorithm to substitute the GA of HGAGO. It gets better performance and requires less time than HGAGO.

This paper is organized as follows. Section 2 introduces some related algorithms for the k-LCS problem. Section 3 presents the matching pair algorithm and the enhanced ACO algorithm to find a longer common subsequence of k sequences. The

*Corresponding author: cbyang@cse.nsysu.edu.tw

experimental results are shown in Section 4. Finally, the conclusion is given in Section 5.

2 Previous Works

It is hard to find the LCS of a set of sequences while the number and the length of sequences are large. Several heuristic and evolutionary algorithms have been proposed for finding longer CS of input sequences, such as the expansion algorithm [1], the BNMAS algorithm [7], the algorithm with genetic algorithm (GA) [2], the algorithm with ant colony optimization (ACO) approach [12], and the hybrid algorithm combined with GA and ACO [14].

Heuristic algorithms are proposed for finding acceptable solutions for the k-LCS problem efficiently. A *stream* is a sequence that does not contain contiguously identical symbols. The *expansion algorithm* (EA) [1], proposed by Bonizzoni *et al.*, compresses each input sequence into its shortest possible stream. Based on 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, the BNMAS algorithm [7] counts the numbers of *available symbol*, the number of occurrence of each individual common symbol in front of a specific location of symbol for each sequence in S. The concurrent number of available symbol is used to decide which symbol can be appended to a CS. By taking the advantage of time-efficiency, EA and the BNMAS algorithm can be applied to generate CSs as initial solutions for evolutionary algorithms.

The genetic algorithm (GA) for solving the k-LCS problem was proposed by Chiang *et al.* [2]. It chooses the shortest sequence in S 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=GTACTGATACTGT 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 $s^m_{p_j}$ and $s^v_{p_j}$ be 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. Equation (1) gives the fitness function of GA.

$$f(s_{p_j}) = \begin{cases} s^m_{p_j} \times s^v_{p_j} & \text{if } s^m_{p_j} = |S|, \\ -1 \times (|S| - s^m_{p_j}) \times s^v_{p_j} & \text{otherwise.} \end{cases} \quad (1)$$

If the template pattern p_j holds a lot of matching bits in the input sequences, we regard it as a good pattern. The fitness value $f(s_{p_j})$ of a good pattern is higher than others. We may get better result by repeatedly performing procedures of GA.

2.1 ACO Algorithm for k-LCS

Shyu and Tsai [12] proposed the method using the *ant colony optimization* (ACO) approach to find longer CS for the k-LCS problem. Suppose there are m ants, $m < k$, each ant chooses a sequence s_i from S to extract a subsequence from s_i as a candidate of CS of S. Let $s_i[j]$ and $a^r_i[j]$ denote the jth character in sequence s_i and the extracting state of the jth character from sequence s_i extracted by ant a_r, respectively. For example, consider s_1 = ATCGTAC, $s_1[2]$ = T and $s_1[6]$=A; a^2_1 = 0101101 indicates that ant a_2 extracts TGTC from s_1. A *common subsequence candidate* (CSC) can be constructed based on a^r_i. Ant a_r might find a better CSC of S according to the residue of pheromone on each position of $s_i \in S$. Based on Equation (2), 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.

The meanings of variables in the formulas of ACO approach are described as follows. In Equation (2), $\tau(v_r)$ is the concentration of pheromone at $v_r \in W = [u_r + 1, min(u_r + d, |s_r|)]$, where $0 \le u_r \le |s_r| - 1$; $\eta(v_r)$ is a cost of v_r defined by function (3); β is a relational parameter between $\tau(v_r)$ and $\eta(v_r)$; q is a random number, $0 \le q \le 1$; q_0 and q_1 are used to decide which searching strategy (exploitation, biased exploration, exclusion) is favored, where $1 \le q_0 < q_1 \le 1$. If $p(v_r) = 1$, a_r chooses $v_r = \arg\ \max_{z_r \in W} \{\tau(z_r)\eta(z_r)^\beta\}$ 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 (2). When all ants finish the tasks of finding CSs of S, the pheromone will be updated on all positions of the input sequences by Equation (4). ρ denotes evaporation coefficient of ant pheromone in nature. When the selected time of the character is decreased, its pheromone evaporates according to Equation (4). In Equation (5), 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 one round. After several iterations, the best result can be obtained based upon these states.

$$p\ (v_r) = \begin{cases} 1 & \text{if } q \le q_0 \text{ and } v_r = \arg\ \max_{z_r \in W} \left\{\tau(z_r)\eta(z_r)^\beta\right\} \text{ (exploitation)}, \\ \frac{\tau(v_r)\eta(v_r)^\beta}{\Sigma_{z_r \in W}\tau(z_r)\eta(z_r)^\beta} & \text{if } q_0 < q \le q_1 \text{ (biased exploration)}, \\ 0 & \text{otherwise } q = q_1 \text{ (exclusion)}. \end{cases} \quad (2)$$

$$\eta(v_r) = \frac{1}{\Sigma_{i=1}^{k}(v_i - u_i)}. \quad (3)$$

$$\tau(t_r) = (1-\rho)\tau(t_r) + \rho\Delta\tau(t_r). \quad (4)$$

$$\Delta\tau(t_r) = \begin{cases} \frac{|c^+|}{M} & \text{if } t_r \text{ is an element of } c^+ \text{ by } a_r, \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

2.2 Hybrid Algorithm with GA and ACO

By taking the advantages of GA and ACO approache, the hybrid algorithm with GA and ACO (HGACO) [14] was proposed. It is a circulating feedback method and it combines two evolutionary methods (GA and ACO) for solving the k-LCS problem.

Algorithm: HGACO
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 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.

In order to find longer CS, HGACO takes the CS found by the heuristic algorithm as the initial input and uses the ACO algorithm to improve the solution. The ACO algorithm searches possible characters in the sequences of S by imitated the behavior of ants searching for their food. It is possible to get locally optimal solution if only one evolutionary algorithm is applied. One can apply other algorithms to disturb the locally optimal solution and to get a longer CS. Hence, the result of the ACO algorithm is taken as the initial solution of GA in HGACO method to find longer CS. With the same principle, we can use the solution of GA as the input of ACO to improve the length of the CS. GA in HGACO method can choose possible common characters to construct a common subsequence by its mutation and crossover operators.

3 Our Algorithm

For improving the solution and reducing the execution time of the HGACO algorithm in Section 2.2, we propose the concept of *matching pair algorithm* (MPA), which can be invoked in our enhanced ACO algorithm. The MPA concept provides better candidates of CSs in an efficient way. By taking the advantages of ACO and MPA, our enhanced ACO (EACO) algorithm for solving the k-LCS problem is also a circulating feedback method that combines ACO algorithm, MPA and heuristic algorithms. Algorithm EACO is given as follows.

Algorithm: EACO
Input: A set S of 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 algorithm.
Step 2: Take c as the initial CS of ACO and look for a new CS s_{ACO} by ACO.
Step 3: Apply MPA to the first t longest solutions to get a new CS s_{MPA}.
Step 4: Let $c = s_{MPA}$.
Step 5: Repeat Steps 2 and 5 until the predefined number of iterations is reached.
Step 6: Return c.

We capture the advantages of ACO's search ability and the benefits of MPA. ACO can produce a large number of solutions in a short search generation and MPA can combine all of possible pairs in ACO's solutions by a greedy method efficiently. MPA tries to find the best matching positions of two CSs, and then to concatenate the left substring (prefix) of one CS and the right substring (suffix) of another CS to produce a new CS. The detail of MPA is given as follows.

Algorithm: MPA

Input: A set S of k input sequences over alphabet Σ and m CSs of S.

Output: A common subsequence (CS) of S.

Step 1: Mark leftmost mapping positions of $s_i \in S$ for each CS from left to right.

Step 2: Mark rightmost mapping positions of $s_i \in S$ for each CS from right to left.

Step 3: Find the best matching partner which can form longer CS by joining the suffix of a CS found from right to left and the prefix of one of m CSs found from left to right.

Step 4: Return the longest CS.

In Section 2.2, GA requires a lot of time to check whether these patterns are valid patterns of CS in S and hardly decides which position in each pattern is a good crossover point. In the matching pair algorithm, we only consider CSs produced by ACO to find position mapping of each CS and regard every position as a possible crossover point to construct a longer CS. For example, consider S={ s_1=ATCGTAC, s_2=CTGTAGC, s_3=GTTCATC}. Let S'={ c_1=TTC, c_2=GTC, c_3=AC} be a set of CSs which are found by ACO. Let $P_D(c_m[i])=[p_1, p_2, \ldots, p_k]$ denote the position distribution of the ith symbol of c_m by the direction D in S, where p_j is a position in jth sequence for $1 \le j \le k$, D=L (or D=R) means the direction for searching position mapping of symbol $c_m[i]$ from left to right (or right to left). By the order of c_1's context, we can find

$$P_L(c_1[1]) = [2,2,2], \quad P_R(c_1[3]) = [7,7,7],$$
$$P_L(c_1[2]) = [5,4,3], \quad P_R(c_1[2]) = [5,4,6],$$
$$P_L(c_1[3]) = [7,7,4], \quad P_R(c_1[1]) = [2,2,3].$$

With the same way, we get

$$P_L(c_2[1]) = [4,3,1], \quad P_R(c_2[3]) = [7,7,7],$$
$$P_L(c_2[2]) = [5,4,2], \quad P_R(c_2[2]) = [5,4,6],$$
$$P_L(c_2[3]) = [7,7,4], \quad P_R(c_2[1]) = [4,3,1],$$

$$P_L(c_3[1]) = [1,5,5], \quad P_R(c_3[2]) = [7,7,7],$$
$$P_L(c_3[2]) = [3,7,7], \quad P_R(c_3[1]) = [6,5,5].$$

Based on the information of $P_D(c_m[i])$, we can find the best pair of CSs by trying to assembly all possible pairs. For example, the leftmost prefix substring TT ($P_L(c_1[1]) = [2,2,2]$ and $P_L(C_1[2]) = [5,4,3]$) can be combined with the rightmost suffix substring AC ($P_R(c_3[1])$=$[6,5,5]$ and $P_R(c_3[2])$=$[7,7,7]$). Note that we only consider that the pairs can produce the longest CS with leftmost prefix substring of c_{j_1} ($P_L(c_{j_1}[i_1])$) and rightmost suffix substring of c_{j_2} ($P_R(c_{j_2}[i_2])$), where $1 \le i_1 \le |c_{j_1}|$, $1 \le i_2 \le |c_{j_2}|$ and $1 \le j_1 \ne j_2 \le k$. We can find that TTAC and GTAC are the longest CSs in combinations of all pairs of CSs.

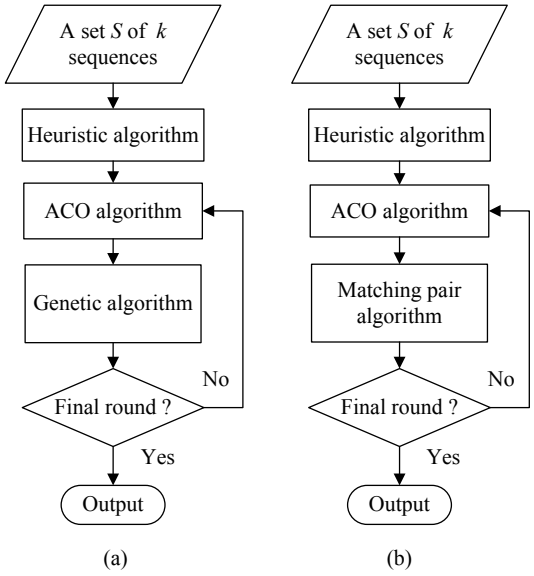

Figure 1: The flowcharts of two algorithms. (a) HGACO [14]. (b) Enhanced ACO (EACO) algorithm with pair matching strategy.

In our experiments, we alternately use ACO and MPA to find better CS. The flowcharts of HGACO and the enhanced ACO (EACO) algorithm with MPA are shown in Figure 1. Compared with the flowchart of HGACO algorithm, one may easily find the difference between HGACO and EACO algorithms. By setting the shorter convergence generation of ACO, we get a lot of better CSs in a short time. Some substrings of one CS may be a part of other longer CSs. In our experiments, the default parameter values of our enhanced ACO algorithm are shown in Table 1.

We add a process to perform three rounds of local search for ACO while all ants finish the tasks of finding CSs of S. For selecting h non-candidate symbols with higher pheromone, we try to take these symbols as candidate symbols. The positions of these symbols do not affect the existence of candidate symbols in the sequence s_i extracted by ant a_r. If we choose some symbols among h non-candidate symbols which are not common symbols, we only choose one of these conflicted symbols finally.

After the completion of local search, we can select new h non-candidate symbols once again and reconsider these symbols whether they are candi-

Table 1: Default parameter values of our enhanced ACO algorithm.

Parameter	q_0	q_1	β	ρ	window size	m	ρ_0	g	h
Value	0.9	0.95	2	0.1	$10(\Sigma_{DNA})$ or $24(\Sigma_{protein})$	16	1	10	5

date symbols or not. It can make up for the shortcomings of ACO search and find better solutions. We set the number of convergence generations of ACO as 10. While all ants cannot improve the best solution in consecutive 10 generations, the ACO algorithm will report the final best solution. ACO with 10 generations can make many solutions in the process and spends little time.

We record the top 80 distinct longer CSs in ACO algorithm and use MPA to find the best solution. The 80 distinct CSs may come from many different groups of solutions and we combine all CSs to get new solutions by MPA. We repeatedly perform the ACO and MPA for 20 iterations in order to reset the pheromone of ACO and recombine all CSs to find better solutions.

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 of the input data, it is possible to design elegant algorithms to deal with these input data and to output excellent results. For preventing such exceptional guess, we use real biological sequences as the input data in our experiments. The biological sequences in our experiment were provided by Shyu and Tsai [12]. The first 600 characters of each input sequence are extracted to form one test sequence. 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) 64×2 Dual Core Processor 3800+ (CPU) and 2GB RAM.

We find that resetting the pheromone can effectively help ACO algorithm get a breakthrough for finding a longer solution and MPA can get better solutions by given some CSs. So we reset the pheromone values at the beginning stage of ACO every time. We test various values of parameters for ACO and MPA. Let *iteration*, *generation* and *top* denote the number of iterations, number of ACO's generations, and number of CSs for MPA, respectively. In the simulations, *iteration*=20, *generation*=10 and *top*=80 are the settings of parameters for EACO algorithm based on the simulation of pre-test cases.

Let DNA_k denote the test set of k DNA sequences. For finding out suitable parameter settings of our EACO algorithm, we take three test cases, Random DNA_{10}, Rat DNA_{20} and Rat DNA_{200}, as the pre-test cases. Note that in the candidate sets of sequences, the solutions of the Random DNA_{10} is better than other Random cases for EACO; the solutions of the Rat DNA_{20} are close; and the solutions of the Rat DNA_{200} are shorter than other types of 200 input sequences.

We use the lengths of CS found by various methods to measure the performance of these methods. If the length of CS is longer obtained by method A, then method A is regarded as a better method for the k-LCS problem. Table 2 shows output lengths of CSs obtained by various algorithms for randomly selected, rat, and virus DNA sequences. Note that the solutions of the ACO* were given by Shyu and Tsai [12] and the solutions of ACO come from our implementation on ACO algorithm with the same parameter setting of ACO*.

In addition, the alphabet size of DNA sequences $|\Sigma_{DNA}| = 4$. The length of CS obtained by each algorithm is the average of 10 independent simulations. In Table 2, we set the numbers of generations of GA and ACO as 500 and 100, respectively. We use the same parameter settings for ACO and GA in HGACO, proposed by Shyu and Tsai [12] and Chiang [2], respectively. Note that we only change the GA's generations from 2000 to 100 in HGACO. In summary, EACO finds out the longest CS than others.

Table 3 shows the execution time required for the testing sets of DNA sequences, including randomly selected, rat, and virus DNA sequences. Tables 4 and 5 show the comparisons of output CS length and execution time, respectively, obtained by various algorithms for various types of protein sequences.

ACO algorithm quickly finds many solutions in a few generations and they can be processed by MPA to form longer solutions. Due to MPA, our EACO algorithm effectively finds better solutions from many CSs found by ACO for every test case of DNA or protein sequences. MPA can be embedded into other algorithms which can find many CSs and combine these solutions. If the solutions have more diversity in position distribution, then

Table 2: Comparison of CS lengths for DNA sequences.

Data type	k	EACO	HGACO	ACO	ACO*	GA	BNMAS	ExpA
Random	10	203.2	201.0	197.3	197.2	185.0	169.0	173.0
	15	189.2	188.8	184.1	185.2	173.5	157.0	164.0
	20	180.2	178.4	176.0	176.2	166.3	157.0	156.0
	25	175.4	175.0	171.8	172.2	164.0	154.0	155.0
	40	164.7	164.2	162.1	161.4	155.0	150.0	148.0
	60	158.2	157.7	155.6	155.4	149.8	146.0	145.0
	80	154.6	153.8	152.4	151.6	145.0	146.0	141.0
	100	151.9	151.2	149.3	148.8	144.0	141.0	141.0
	150	146.2	145.6	144.0	143.4	140.3	139.0	134.0
	200	144.0	143.4	142.0	141.0	139.5	137.0	133.0
Rat	10	187.6	183.3	180.7	182.0	170.5	163.0	160.0
	15	171.7	170.3	166.7	166.6	156.3	149.0	148.0
	20	160.7	160.4	155.6	160.0	148.5	149.0	146.0
	25	159.4	159.0	156.4	155.8	141.8	145.0	144.0
	40	145.7	145.7	143.1	143.4	134.5	135.0	136.0
	60	144.3	144.3	141.9	142.4	133.8	131.0	129.0
	80	131.2	130.9	128.1	128.8	123.5	121.0	116.0
	100	127.8	126.7	125.8	124.6	121.8	118.0	113.0
	150	120.5	120.1	117.4	115.6	115.0	109.0	100.0
	200	117.2	117.2	113.7	114.6	111.0	108.0	101.0
Virus	10	206.4	201.2	199.1	197.6	186.0	177.0	170.0
	15	189.3	186.0	183.9	183.6	172.3	168.0	162.0
	20	175.9	175.1	172.3	173.8	160.8	157.0	151.0
	25	181.7	181.6	178.2	179.0	162.3	172.0	158.0
	40	157.7	157.3	155.4	155.0	144.0	147.0	146.0
	60	153.9	153.7	152.3	150.6	146.0	146.0	141.0
	80	150.1	149.5	148.7	145.8	138.8	143.0	136.0
	100	147.7	145.3	145.5	143.4	136.3	141.0	136.0
	150	144.7	144.7	142.4	141.6	139.0	136.0	132.0
	200	143.3	143.3	141.3	140.6	136.8	135.0	133.0

Table 3: Comparison of execution time (sec) for DNA sequences.

Data type	k	EACO	HGACO	ACO	GA	BNMAS	ExpA
Random	10	33.23	119.65	2.44	170.44	0.00	132.85
	15	43.77	176.43	3.35	193.43	0.01	99.06
	20	58.35	227.86	4.64	235.17	0.01	67.03
	25	69.15	323.05	5.80	307.17	0.01	70.72
	40	110.77	882.95	8.77	487.78	0.01	105.56
	60	163.89	1591.36	14.44	884.41	0.02	274.19
	80	228.61	2082.21	22.58	1134.01	0.03	630.58
	100	287.89	2611.10	29.10	1299.08	0.04	1193.59
	150	457.52	3745.31	50.15	1934.55	0.06	3930.63
	200	657.70	5082.51	74.18	2529.04	0.08	9272.10
Rat	10	31.27	108.55	2.31	138.60	0.00	71.47
	15	39.93	161.68	3.29	174.06	0.00	48.63
	20	50.25	213.70	4.38	224.91	0.01	29.15
	25	65.65	288.58	5.49	275.95	0.01	36.39
	40	96.32	870.15	8.29	451.98	0.01	85.56
	60	151.13	1594.34	13.86	741.84	0.02	252.88
	80	165.18	1930.31	15.07	1069.67	0.03	564.89
	100	212.44	2397.73	22.10	1308.18	0.04	1082.21
	150	322.99	3546.74	37.75	1779.06	0.06	3579.92
	200	506.98	4281.04	61.79	2603.18	0.08	8499.41
Virus	10	35.75	124.55	2.71	157.79	0.00	136.95
	15	44.71	179.38	3.79	165.47	0.00	81.92
	20	60.64	234.79	5.28	206.54	0.01	58.42
	25	78.66	347.58	6.25	224.17	0.01	65.02
	40	123.39	956.28	13.61	412.46	0.02	141.98
	60	174.50	1888.84	20.61	745.37	0.02	279.87
	80	248.98	2239.09	28.85	1031.12	0.03	608.88
	100	304.41	2732.27	32.24	1273.88	0.04	1185.56
	150	385.15	3977.47	41.49	1805.77	0.06	4254.76
	200	742.74	5245.42	85.07	2387.33	0.08	10233.00

Table 4: Comparison of CS lengths for protein sequences.

Data type	k	EACO	HGACO	ACO	ACO*	GA	BNMAS	ExpA
Random	10	58.4	56.5	55.4	54.0	44.5	47.0	34.0
	15	49.3	47.7	47.1	46.2	38.3	40.0	28.0
	20	44.2	43.2	42.5	42.4	36.2	37.0	27.0
	25	41.3	40.5	39.7	40.0	34.2	35.0	28.0
	40	36.0	35.9	34.6	34.2	30.5	31.0	25.0
	60	32.6	32.3	31.4	30.6	28.6	30.0	24.0
	80	31.0	30.5	29.9	29.0	27.2	28.0	24.0
	100	29.9	29.6	28.9	28.4	26.7	28.0	25.0
	150	27.4	27.2	26.7	26.0	25.1	25.0	24.0
	200	26.0	26.0	25.2	25.0	24.3	24.0	23.0
Rat	10	66.9	64.5	63.4	63.4	52.4	54.0	34.0
	15	59.3	57.7	56.4	56.6	46.1	53.0	28.0
	20	49.7	49.5	47.1	47.8	40.7	45.0	27.0
	25	47.3	47.3	45.7	46.2	39.1	39.0	28.0
	40	45.2	45.2	43.6	44.2	37.7	43.0	25.0
	60	44.3	44.3	42.9	43.0	36.4	42.0	24.0
	80	41.4	41.4	39.2	39.6	34.7	39.0	24.0
	100	37.1	36.6	35.6	37.0	30.0	33.0	25.0
	150	34.3	33.6	32.1	34.0	30.0	32.0	24.0
	200	32.4	31.7	30.4	32.4	27.8	29.0	23.0
Virus	10	70.3	67.7	67.5	65.6	54.5	60.0	34.0
	15	59.5	57.5	57.3	55.8	47.2	53.0	28.0
	20	55.7	55.2	53.1	53.6	44.5	48.0	27.0
	25	52.2	51.0	50.0	49.6	43.0	46.0	28.0
	40	47.4	47.4	46.2	46.4	38.2	42.0	25.0
	60	45.4	45.2	44.4	43.4	37.0	44.0	24.0
	80	43.3	43.2	42.0	43.0	36.1	40.0	24.0
	100	43.0	42.9	41.9	42.0	34.1	39.0	25.0
	150	43.2	43.1	41.5	42.6	34.4	38.0	24.0
	200	41.8	41.2	40.5	41.0	33.6	40.0	23.0

Table 5: Comparison of execution time (sec) for protein sequences.

Data type	k	EACO	HGACO	ACO	GA	BNMAS	ExpA
Random	10	14.84	13.67	2.08	43.11	0.02	1.08
	15	17.97	18.05	2.47	46.32	0.02	2.83
	20	22.51	19.91	2.53	53.34	0.02	6.58
	25	25.64	23.36	3.13	57.71	0.02	12.41
	40	36.12	35.16	4.18	74.31	0.03	49.81
	60	52.56	52.19	5.84	110.66	0.05	166.69
	80	69.38	80.23	8.12	138.73	0.06	391.91
	100	90.58	125.68	11.15	186.14	0.08	764.19
	150	138.22	267.79	18.01	338.66	0.13	2565.38
	200	197.50	391.10	23.60	499.83	0.16	6079.77
Rat	10	15.24	15.68	2.06	51.12	0.02	1.25
	15	19.44	21.22	2.34	56.02	0.02	2.81
	20	18.18	25.20	2.57	62.64	0.02	5.97
	25	21.76	27.37	2.95	67.20	0.02	11.94
	40	31.61	43.63	3.97	93.20	0.03	48.02
	60	47.00	75.48	6.32	133.10	0.05	160.30
	80	62.99	124.19	7.73	191.77	0.08	397.48
	100	73.47	185.49	10.27	207.78	0.09	758.39
	150	116.76	385.44	15.76	400.20	0.14	2565.08
	200	173.06	480.35	19.29	562.55	0.17	6236.59
Virus	10	16.38	16.94	2.01	58.49	0.02	1.55
	15	20.04	20.44	2.61	58.23	0.03	3.05
	20	23.03	24.43	3.04	70.84	0.02	7.64
	25	28.57	27.50	3.50	77.32	0.03	12.81
	40	40.20	44.54	4.79	109.18	0.03	52.75
	60	61.26	75.45	7.51	141.27	0.05	184.61
	80	86.62	136.45	9.42	204.95	0.06	453.97
	100	119.83	219.10	12.38	265.77	0.08	915.84
	150	193.24	466.43	22.59	511.44	0.16	3857.95
	200	266.32	633.76	31.13	783.87	0.20	9194.26

it has better chance to combine the solutions to get better ones. Setting fewer generations of ACO may produce more distinct solutions for MPA in a short period of time. Resetting the pheromone of ACO in each iteration may change the behavior of ACO and may escape from the trap of locally optimal solutions. With the advantages of ACO and MPA, our EACO algorithm outperforms others.

5 Conclusion

The longest common subsequence problem is a classical problem in computer science and computational biology. If the number of the input sequences is not fixed, it will become an NP-hard problem. Therefore, several methods and approximate algorithms have been proposed to find longer common subsequence (CS) of k sequences, but some of these algorithms are not so efficient. In order to improve the efficiency, we adopt the hybrid strategy to find better solutions for k-LCS. In this paper, we propose the enhanced ACO (EACO) algorithm with heuristic algorithm and matching pair algorithm (MPA). We get longer CS than other algorithms in the experiments.

In the future, we may try to reduce the execution time of the EACO algorithm and improve the effectiveness of our algorithm. One may modify MPA to combine the solutions by using various alignment techniques, or apply EACO algorithm to the multiple sequence alignment problem. In addition, we may consider our method in combination with more algorithms to get 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, "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] K. Hakata and H. Imai, "The longest common subsequence problem for small alphabet size between many strings," *Proceedings of the Third International Symposium on Algorithms and Computation, Lecture Notes in Computer Science 650, Springer Verlag*, pp. 469–478, 1992.

[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. W. Irving and C. B. Fraser, "Two algorithms for the longest common subsequence of three (or more) strings," *Proceedings of CPM'92, the Fourth Annual Symposium on Combinatorial Pattern Matching, Arizona, Lecture Notes in Computer Science 644, Springer Verlag*, pp. 214–229, 1992.

[10] 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.

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

[12] 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.

[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.

[14] H.-Y. Weng, S.-H. Shiau, K.-S. Huang, and C.-B. Yang, "Hybrid algorithm for the longest common subsequence problem," *Proceedings of the 26th Workshop on Combinatorial Mathematics and Computation Theory*, Chiayi, Taiwan, pp. 122–129, 2009.