Motif Finding in Biological Sequences *

Ying-Jer Liao Chang-Biau Yang† Shyue-Horng Shiau
Department of Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung, Taiwan 804
{liaoyj, cbyang, shiaush}@cse.nsysu.edu.tw

Abstract

A huge number of genomic information, including protein and DNA sequences, is generated by the human genome project. Deciphering these sequences and detecting local residue patterns of multiple sequences are very difficult. One of the ways to decipher these biological sequences is to detect local residue patterns from them. However, detecting unknown patterns from multiple sequences is still very difficult. In this paper, we propose an algorithm, based on the Gibbs sampler method, for identifying local consensus patterns (motifs) in monomolecular sequences. We first designed an ACO (ant colony optimization) algorithm to find a good initial solution and a set of better candidate positions for revising the motif. Then the Gibbs sampler method is applied with these better candidate positions as the input. The required time for finding motifs using our algorithm is reduced drastically. It takes only 20% of time of the Gibbs sampler method and it maintains the comparable quality.

Key words: Computational biology, motif finding, ACO algorithm, local sequence alignment.

1 Introduction

In 1920, Dr. Hans Winkler brought up this noun, *genome project*. From then on, more and more organizations started to involve in genome projects. Current studies show that the human genome consists of approximately 3 billions (3×10^9) base pairs. Researchers in America analyzed the human genome, and they found that there are around 35 thousands fragments.

Consequently, each of genome projects generate a huge number of data containing the information of the genomic sequences. How to search or classify the genomic sequences efficiently in such huge databases is a very difficult task. Therefore, two essential issues about genes in biology should be studied [7]. The first is what the function of each gene is. The second is how the genes are expressed.

To explore the function of each gene is a hard mission and has been studied intensively. Basically, it depends on the characteristics of the gene family. The most successful ways for characterizing a gene family is based on probabilistic models, such as *Hidden Markov Models* (HMMS) [12]. HMMS provides a global model to solve this problem, and HMMS allows three operations, insertions, deletions and transpositions. These capabilities represent that similar genes have had a common evolutionary history and the evolution process.

The second mission is even more difficult to accomplish than the first one. In the beginning, biologists have to determine the regulation of gene expression in animals. They did the determination according to relatively short sequences in the surrounding region of a gene. These sequences are various in length (about $5 \sim 12$) and there are no insertions, and deletions in the same position, but sometimes they may occur multiple times.

The motif finding method can be applied to comparative genomes on multiple species. The sequences are associated with several related genes from a single species. All the multi-gene approaches have an important primary capability. They will find only common elements of genes of a given organism.

Furthermore, the set of genes was obtained from experimental data. Several methods have been proposed for finding the patterns shared in monomolecular in a training set [2, 9, 13]. In this paper, we propose a more efficient method to find a motif in sequences. Our method is a combination of *ant colony optimization algorithm* (ACO) [6] and *the Gibbs sampler method* [9]. The goal of our method is to reduce the required computing time and to get better solution including sensitivity and specificity.

In Section 2, we will introduce what the motif finding task is, and some methods used to find motifs. In Section 3, we present the ACO algorithm, proposed by

*This research work was partially supported by the National Science Council of the Republic of China under NSC-90-2213-E-110-015.

†To whom all correspondence should be sent.

Dorigo [6]. By using this algorithm, we therefore can find better starting positions of sequences in a training set for our algorithm. These positions of sequences in a training set provide a set of candidates for the Gibbs sampler method to get a better solution for motif identification. We illustrate our method in Section 4. Some experimental results will be shown in Section 5. Finally, conclusions will be given in Section 6.

2 The Gibbs Sampler Method for Motif Finding

The *multiple sequences alignment* (MSA) [8, 15] has been studied intensively for many years. There are two versions of the MSA problem. The first is the *global MSA*, whose goal is to align entire sequences in the training set. The second is the local MSA. Its goal is to locate shorter patterns which are shared by all sequences in the training set. Global MSA has been well studied recently. Biologists apply these methods to classify monomolecular sequences. On the contrary, the local MSA has not been developed well yet.

Several methods have been proposed to identify the motifs shared among monomolecular sequences [2, 9, 13]. Most of these methods are based on local MSA. The local MSA is to locate a pattern of fixed length from each input sequence such that the score determined from the set of patterns is optimized. Stormo and Hartzell developed a heuristic iterative algorithm with the relative entropy score to find the optimal score [14]. Though the local MSA with the relative entropy scoring scheme is NP-hard, several practical algorithms have been developed. For example, the Gibbs sampler method [9] and EM (*expectation maximization*) [1, 10] algorithm are widely used.

2.1 The Motif Finding Problem

Local MSA can be regarded as a *motif finding* problem. We illustrate the problem with n protein sequences in Figure 1.

The motifs have several characteristics as follows [7]:

- The patterns are relatively short.
- The patterns are not aligned completely in all sequences. Therefore the variation is represented by a probability matrix. No single sequence can define these patterns exactly.
- The exact locations of patterns may not be important, because the goal of patterns is bound to other molecules.
- The operations of insertions and deletions do not occur, because these operations would have drastic effects on the conformation of patterns.

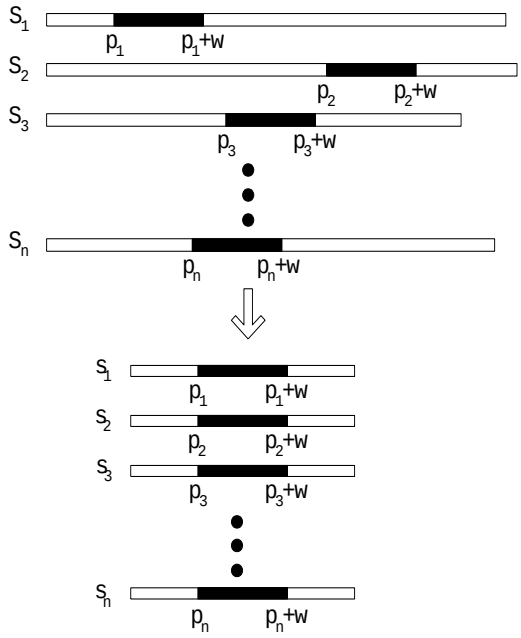

Figure 1: The motif finding in a set of n protein sequences.

- The patterns or motifs should be common in most, but not in all sequences. So it is not necessary that the motif appears in each of the given family of sequences.

We give an example to illustrate these characteristics. Assume T is a set of four DNA sequences, S_1, S_2, S_3 and S_4.

S_1:ATCGATGCGA
S_2:TGCCATGTA
S_3:GTAGCGCTGT
S_4:CCGATCGATA

Assume $w = 5$ is the length of the motif. After T is aligned, we can find the pattern in each of sequences. The patterns are underlined, denoted as x_1, x_2, x_3 and x_4, which is a motif in the training set Tas follows.

x_1:CGATG
x_2:CCATG
x_3:CGCTG
x_4:CGATC

Although this motif in T is not aligned completely but most nucleic acids are the same in positions. There are only three positions containing different nucleic acids in this motif, position 2, position 3 and position 5, which are underlined.

2.2 The Gibbs Sampler Method

Lawrence *et al.* proposed a local MSA algorithm to align n protein sequences to find motifs (specific patterns) in each sequence with a fixed length w [9]. They

assumed that prior information of the patterns or their locations within the sequences are unknown. Their algorithm has three fundamental characteristics. First, these substrings found in the given n protein sequences are fixed and have no gaps in the window of size w. Second, the pattern at each position is described by a probabilistic model of amino acid frequencies. Third, the locations of each pattern within the sequences can be calculated by a probabilistic model of amino acid frequencies.

This algorithm has two main data structures, which are a probabilistic model and a set of positions. The probabilistic model is a two-component mixture model. One component is applied to calculate the variables of amino acid frequencies at each position i of a set of subsequences, $1 \leq i \leq w$. The other one describes all other positions in the sequences (the "background"). The set of positions a_k, $1 \leq k \leq n$, constitutes the alignment, where a_k denotes the starting position of pattern x_k in sequence k.

The concept of their algorithm is as follows [9, 13]:

Input: A set T of n sequences $S_1, S_2, \cdots, S_n$, and motif length w.

Output: A motif $x_1, x_2, \cdots, x_n$, each of length w, where each x_i is a substring of S_i.

Step 1: Randomly choose a beginning position a_i in each sequence S_i, $1 \leq i \leq N$. Let x_i be a substring of S_i with length w and starting from position a_i.

Step 2: Randomly select one sequence, S^*, in T, whose substring is x_{S^*}. Let $U = \{x_1, x_2, \cdots, x_n\} - \{x_{S^*}\}$.

Step 3: With U, create a $20 \times w$ probability matrix Q. In the matrix, each row represents an amino acid. Note the number of amino acids is 20. $Q_{r,j}$ represents an occurrence frequency, which is the number of sequences that amino acid r appears at position j.

Step 4: Let P_l be a substring of S^* with length w and starting at position l. For each P_l, calculate a score q_l, which involves the frequency matrix Q and the background distribution B as follows:

$$q_l = \sum_{r=1}^{20} \sum_{j=1}^{w} Q_{r,j} log \frac{Q_{r,j}}{B_j} \qquad (1)$$

Step 5: Randomly choose a starting position l in S^* with probability proportional to q_l. The new starting position l will construct P_l which is a new substring of S^* with length w. Then x_{S^*} is replaced by P_l.

Step 6: If the best solution has not been changed after some predefined iterations, terminate the algorithm; otherwise, go to Step 2.

3 Ant Colony Optimization Algorithm (ACO)

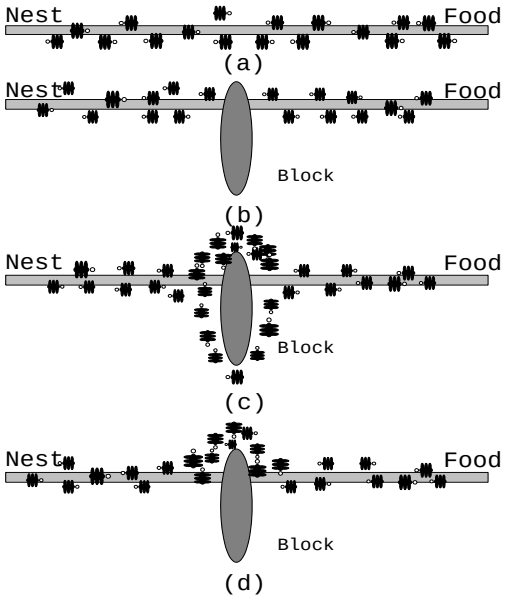

Figure 2: (a) Real ants travel a path between nest and food source. (b) A block hampers the movements of the ants on the path. Then each of the ants chooses one way, left or right, with equal probability. (c) Since there are more ants traveling on the shorter path, pheromone are deposited more quickly on the shorter path. (d) All ants choose the shorter path bit by bit.

Dorigo proposed an ant colony optimization algorithm (ACO) [6, 5], which has been successfully applied to several NP-hard problems, such as *the traveling salesman problem (TSP)* and *quality of service problem (QoS)*. [3, 5, 4, 11]. Just as its name implies, the ACO algorithm is enlightened by the behavior of real ant colony. One of the main ideas of the ACO algorithm is to exchange the information within the colony of ants. Here the ants can be regarded as artificial ants. As shown in Figure 2, real ants communicate with each other via pheromone. In the ACO algorithm, the artificial pheromone trails are presented by a number, and the numeric information of pheromone trails is modified by the artificial ants. The ACO algorithm is as follows:

Step 1: Set parameters and initialize pheromone trails.

Step 2: Each of ants constructs a solution.

Step 3: Calculate the scores of the m solutions.

Step 4: Update the pheromone trails.

Step 5: If the best solution has not been changed after some predefined iterations, terminate the algorithm; otherwise, go to Step 2.

The ACO algorithm simulates the behavior of real ants. Each of the real ants smells pheromone. Its movement depends on the smell. When an ant travels on a path, some pheromone will be left on the path. Each of the following ants selects a better path based on the the amount of pheromone on the trail. Each ant attempts to select a path with more pheromone. The pheromone trail can be presented as numeric information. Therefore we should know how to calculate the pheromone and set the parameters. Since the ACO algorithm is first proposed for solving the traveling salesman problem (TSP), we explain the formulas with a view of graphs. The formulas and parameters of the ACO algorithm can be written as follows.

Traveling probability: In Step 2 of the ACO algorithm, each of the ants constructs a solution by traveling all nodes in the graph. How does ant k choose a path between two nodes? By applying a pheromone action selection rule, ant k will make a choice. In other words, the travel of ant k from node s to node u depends on the pheromone probability action selection rule. The formula of the pheromone probability is given as follows:

$$p_k(s,u) = \frac{[\tau_{s,u}(t)]^\alpha [\eta(s,u)]^\beta}{\sum_{u \in J_k(s)} [\tau_{s,u}(t)]^\alpha [\eta(s,u)]^\beta} \qquad (2)$$

The left side $p_k(s,u)$ represents a probability that the ant k travels from node s to node u. α and β are two parameters which are applied to determine the influence of the pheromone trails. $J_k(s)$ denotes the set of neighborhood which ant k has not visited yet. $\eta(s,u)$ denotes the weight of the path from node s to node u. The two parameters, α and β, affect the solution probabilistic action choice rule by different variables. If α = 0, ant k chooses the nearest node. It corresponds to a classical greedy algorithm. If β = 0, ant k only considers the consistency of pheromone. This case will lead to the rapid emergence of a stagnation situation with the corresponding generation of tours which are strongly suboptimal. $\tau_{s,u}(t)$ will be introduced as follows.

Pheromone update: After all ants have constructed their solutions, the pheromone will be updated. At first, the consistency of pheromone on each edge is lowered by a constant parameter. And each of the ants will add pheromone on the edges it has visited:

$$\tau_{s,u}(t+1) = (1-\rho)\cdot\tau_{s,u}(t) + \sum_{k=1}^{m_{s,u}} \triangle\tau_{s,u}^k(t) \qquad (3)$$

The left side $\tau_{s,u}(t+1)$ means the consistency of pheromone on the edge from node s to node u. The parameter ρ, $0 < \rho < 1$, is the rate of the pheromone evaporation, and it is applied to avoid accumulating the pheromone unlimitedly. If the ants do not choose an edge, the consistency of pheromone on the edge will decrease exponentially. Thus the parameter enables the ACO algorithm to ignore a bad path that was chosen by an ant making a bad decision. $m_{s,u}$ denotes the total number of ants which have traveled from node s to node u. $\triangle\tau_{s,u}^k(t)$ is the variable of pheromone on the edge which it has visited.

4 Our Method

There are two flaws in the Gibbs sampler method. First, in Step 1 of the Gibbs sampler method, it randomly chooses a beginning position a_i of each sequence S_i in T. By using these random beginning positions, it makes slow progress and is very uncertain to converge to a better solution. Second, Step 4 of the Gibbs sampler method calculates the score q_l at each position l in S^*. In fact, the scores at most positions of S^* are not good enough to be paid more attention. Most positions do not provide any help to find a better convergent solution. So, calculation of the scores at most positions is futile.

We propose a concept to improve the performance. Our concept is to apply ACO algorithm to find a set of better candidate positions in each sequence. Then we apply the Gibbs sampler method with the set of better candidate positions as the inputs. And we only calculate the scores of those candidate positions in sequences instead of all positions.

By employing the ACO algorithm, we improve the Gibbs sampler method on both efficiency and quality. The total required computing time is reduced, because the beginning positions are randomly chosen in the Gibbs sampler method, which takes very long time to converge to a better solution. In addition, the final solution of our method becomes better, because it is very easy to fall into the locally optimal trap in the Gibbs sampler method with random starting positions.

In our strategy, we shall apply ACO algorithm to find a sample of the motif in input sequences. Since our ACO algorithm is not used to solve graph problems, we have to rewrite Formula 2 and Formula 3 in ACO algorithm in Section 3 as follows:

Probability:

$$p_k(l_u) = \frac{[\tau_{l_u}(t)]^\alpha}{\sum_{u \in C} [\tau_{l_u}(t)]^\alpha} \qquad (4)$$

The left side $p_k(l_u)$ represents the probability that ant k chooses the character u at position l. α is a parameter which is used to determine the influence of the

pheromone trails. C is the character set of input sequences. $\tau_{l_u}(t)$ is explained as follows.

Pheromone update:

$$\tau_{l_u}(t+1) = (1-\rho)\cdot\tau_{l_u}(t) + \sum_{k=1}^{m_{l_u}} \triangle\tau_{l_u}^k(t) \quad (5)$$

The left side $\tau_{l_u}(t+1)$ means the consistency of pheromone on the character u at position l. The parameter ρ, $0 < \rho < 1$, is the rate of the pheromone trails evaporation. m_{l_u} denotes the total number of ants which carry the character u at position l. $\triangle\tau_{l_u}^k(t)$ denotes the variable of pheromone on the character u at position l that ant k has chosen.

The ACO algorithm is to simulate the characteristic of real ants. The travel of ants between nodes depends on the pheromone. We presume that each ant travels from the starting point to the terminal point and passes through w middle nodes. On each node, there are $|C|$ kinds of foods, where $|C|$ denotes the number of characters in set C. In our ACO algorithm, pheromone τ_{l_u} is not left on edges, it is left on nodes (see Figure 3.), where $1 \le l \le w$ and $u \in C$. Each ant travels from the starting point *S*. When each ant passes through middle nodes, it chooses to pick up one of the $|C|$ foods depending on the consistency of pheromone. After each ant arrives at the terminal point *E*, it carries w foods which are collected from middle nodes. Those w foods are constructed into a string to form a *sample*.

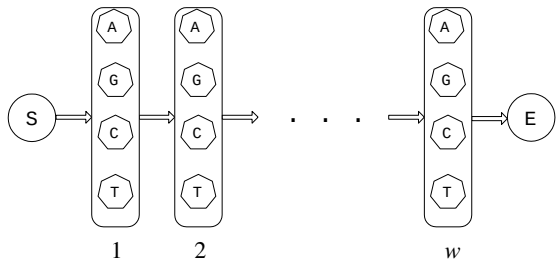

Figure 3: The ant travelling model of our ACO algorithm with DNA sequences. Each ant k travels from the starting point *S* to the terminal point *E*. When each ant passes through middle nodes, it picks up one of the four foods A, G, C, and T.

Ant k constructs a sample x^k with length w by Formula 4. The initial probability of each character r at each position l is $\frac{1}{|C|}$. Then sample x^k compares with each sequence S_i to find the best matched substring x_i^k of S_i. Let $V^k = \{x_1^k, x_2^k, \cdots, x_n^k\}$. Each ant k calculates the score q^k for V^k by Formula 1 and get the rank of q^k by sorting the scores. After all ants finish this procedure, we update the pheromone trails accordingly by Formula 4; and we will update the best sample x^b if the best score q^b has been changed, where x^b denotes the best sample of all samples and q^b denotes the score for x^b. The m ants repeat this procedure until the best sample x^b is not changed for some predefined iterations. By using the best sample x^b to compare with each sequence S_i, we can find a set A_i of better candidate positions for each sequence S_i. Then in A_i, we select the best position to be the initial position for S_i in the Gibbs sampler method.

When applying the Gibbs sampler method in our algorithm, we do not calculate the score q_l of each position l in S^*. We calculate only the score q_l of each candidate position l in the set A_{S^*}. Then by using these scores and candidate positions, we can find the new substring x_{S^*} of S^*.

The number of candidate positions contained in each set A_i is depending on a parameter θ. If the length of sequence S_i is n_i, then we find $\frac{n_i}{\theta}$ candidate positions in S_i to construct A_i, where $1 \le \theta \le n$. For example, if the length of the sequence S_i is 1000 and the parameter θ is 5, then we find 200 candidate positions in the sequence to construct A_i. If $\theta = 1$, A_i contains all positions in sequence S_i.

Algorithm MFS (Motif Finding for Sequences)

Input: A set T of n sequences $S_1, \cdots, S_n$, and motif length w.

Output: $x_1, \cdots, x_n$, each of length w, where x_i is a substring S_i.

Step 1: Set parameters and initialize pheromone trails.

Step 2: Each ant k randomly constructs a sample x^k with length w. The probability of choosing the character is calculated by Formula 4.

Step 3: Each ant k compares sample x^k with each sequence S_i to find the best matched substring x_i^k in S_i. Then each ant k gets a set $V^k = \{x_1^k, x_2^k, \cdots, x_n^k\}$.

Step 4: Apply the score function, Formula 1, to calculate the score q^k of each substring set V^k.

Step 5: Update the pheromone with Formula 5.

Step 6: Find the best score q^b among all q^ks. And update the best sample x^b accordingly. If the best sample x^b is not changed for some predefined iterations, go to Step 7; otherwise, go to Step 2.

Step 7: Compare x^b with each sequences S_i to find a set A_i of $\frac{n_i}{\theta}$ better candidate positions, where n_i is the length of sequence S_i and θ is a predefined parameter. And let x_i be the best position in A_i.

Step 8: Randomly select one sequence, S^*, in T, whose best position is x_{S^*}. Let $U = \{x_1, x_2, \cdots, x_n\} - \{x_{S^*}\}$.

Step 9: With U, create a $|C| \times w$ probability matrix Q, where each row represents a character in the set C. $Q_{r,j}$ represents an occurrence frequency, which is the number of sequences that amino acid r appears at position j.

Step 10: For each candidate position l of A_{S^*}, calculate the score of q_l by using Formula 1.

Step 11: Randomly choose a starting position l in A_{S^*} with probability proportional to q_l. Update the best substring set V^b if the best score q^b is changed, where V^b denotes the substring set of the best motif.

Step 12: If the best substring set V^b is not changed after some predefined iterations, terminate our algorithm; otherwise, go to Step 8.

5 Experimental Results

To completely illustrate our algorithm, we do two experiments to test on different experimental domains. One is on real domains, and the other is on artificial domains.

5.1 Experiments on real domains

Recall that our goal is to find the motifs. The motifs can express what the characteristic of the gene family is. Some common regulatory sites or motifs have been found in some genes families. The helix-turn-helix (HTH) motif, our test data on real domain, is referred to Lawrence *et al.* [9]. The HTH motifs generally occur in different sequence contexts.

The HTH structure consists of 20 contiguous amino acids. The test data contains 30 proteins. It is shown in Table 1. We consider that the length of motif, w, is 18. The results of this experiment are shown in Table 2. Our hardware environment is PC with AMD CPU 800 GHz and software environment is based on Windows 98 with Borland C++Builder. The results include the Gibbs sampler method (*Lawrence et. al.*) and our method with parameter θ from 1 to 12. When $\theta = 1$, our average computing time is half of that of the Gibbs sampler method. When $\theta = 12$, our average computing time is only 10% of that of the Gibbs sampler method. We also make a better progress on the average score. Our variances of scores are also smaller than those of the Gibbs sampler method. Since our scores are better than their scores, our method has much more chances to find a better solution. Note that it is very easy for their method to fall into a local optimal trap.

The score histogram in Table 1 is shown in Figure 4. In the figure, their method is easy stuck over a local optimum. However, the results of our method congregate in the high score area. Figure 5 shows the time and score performance curves in Table 2. It shows that our method has much improvements on both time and scores.

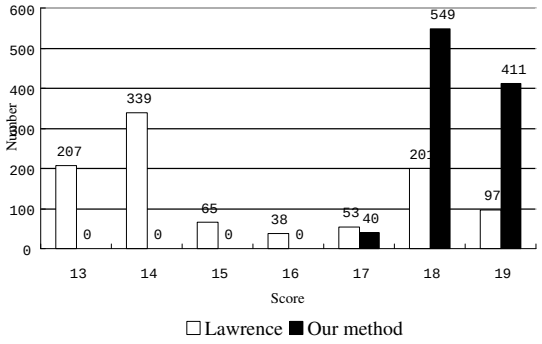

Figure 4: The performance histogram of case $\theta = 1$ in Table 2.

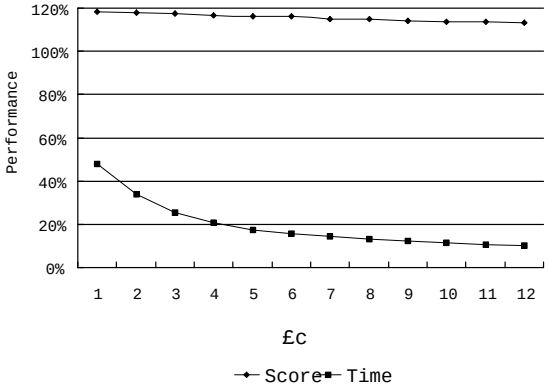

Figure 5: The performance curve in Table 2.

In addition, the best solutions of our method and the Gibbs sampler method are the same score. We show another experiment result for this test data in Table 3. Now, we are interested in the solutions with the scores which are greater than 19. In Figure 4, our method has 411 rounds that the scores are greater than 19 and the Gibbs sampler method has only 97 rounds. Table 3 shows the average computing time of our method and the Gibbs sampler method with acceptable solutions. In the best solution case (score $= 19.41$), our average computing time is 20% of that of the Gibbs sampler method. Our average computing time is about 10% of that of the Gibbs sampler method in other acceptable cases (score > 19).

Table 1: The experiment of real data, which is the HTH testing data that contains 30 proteins. The HTH is aligned with common pattern as follows. In this alignment, columns from left to right are sequence name, locations a_i of the common pattern in each sequence and aligned sequences.

sigma-37	225	···LTYIQNKSQK	ETGDILGISQMHVSRLQR	KAVKKLREAL···
spoIIIC	253	···LDLKKEKTQR	EIAKELGISRSYVSRIEK	RALMKMFHEF···
NAHR	25	···NQLLVDRRVS	ITAENLGLTQPAVSNALK	RLRTSLQDPL···
ANTP	329	···NRYLTRRRRI	EIAHALCLTERQIKIWFQ	NRRMKWKKEN···
NTRC	452	···ALAATRGNQI	RAADLLGLNRNTLRKKIR	DLDIQVYRSG···
DICA	25	···RRKNLKHTQR	SLAKALKISHVSVSQWER	GDSEPTGKNL···
MERD	54	···DAALQRLCFV	RAAFEAGIGLGALARLCR	ALDAANCDET···
FIS	76	···VMQYTRGNQT	RAALMMGINRGTLRKKLK	KYGMN
MATA1	102	···KQSLNSKEKE	EVAKKCGITPLQVRVWFI	NKRMRSK
CII	28	···LNKIAMLGTE	KTAEAVGVDKSQISRWKR	DWIPKFSMLL···
CRP	172	···DGMQIKITRQ	EIGQIVGCSRETVGRILK	MLEDQNLISA···
LambdaCRO	18	···KDYAMRFGQT	KTAKDLGVYQSAINKAIH	AGRKIFLTIN···
P22CRO	15	···DVIDHFGTQR	AVAKALGISDAAVSQWKE	VIPEKDAYRL···
ARAC	199	···HLADSNFDIA	SVAQHVCLSPSRLSHLFR	QQLGISVLSW···
FNR	199	···REFRLTMTRG	DIGNYLGLTVETISRLLG	RFQKSGMLAV···
HTPR	255	···LDEDNKSTLQ	ELADRYGVSAERVRQLEK	NAMKKLRAAI···
NTRC	447	···ALRHTQGHKQ	EAARLLGWGRNTLTRKLK	ELGME
CYTR	14	···KKQETAATMK	DVALKAKVSTATVSRALM	NPDKVSQATR···
DEOR	26	···LKRSDKLHLK	DAAALLGVSEMTIRRDLN	NHSAPVVLLG···
GALR	6	MATIK	DVARLAGVSVATVSRVIN	NSPKASEASR···
LACI	8	MKPVTLY	DVAEYAGVSYQTVSRVVN	QASHVSAKTR···
TETR	29	···EVGIEGLTTR	KLAQKLGVEQPTLYWHVK	NKRALLDALA···
TRPR	69	···ELLRGEMSQR	ELKNELGAGIATITRGSN	SLKAAPVELR···
NIFA	498	···ALEKAGWVQA	KAARLLGMTPRQVAYRIQ	IMDITMPRL
SPOIIGB	208	···LVGEEEKTQK	DVADMMGISQSYISRLEK	RIIKRLRKEF···
PIN	163	···RLIAAGTPRQ	KVAIIYDVGVSTLYKRFP	AGDK
PURR	5	ATIK	DVAKRANVSTTTVSHVIN	KTRFVAEETR···
EBGR	6	MATLK	DIAIEAGVSLATVSRVLN	DDPTLNVKEE···
LEXA	93	···LIGKVAAGES	ILAQEHIESHYQVDPALF	HPRADFLLRV···
P22C1	28	···LNRIAIRGQR	KVADALGINESQISRWKG	DFIPKMGMLL···

5.2 Experiments on artificial domains

In addition to the experiments on real domains, we do testing of our algorithm on some artificial data. We randomly generate sets of artificial amino acid sequences. The numbers of the sequences are 50, 70 and 90, respectively. The average lengths of the sets of sequences are 500 bp, 1000 bp and 1500 bp, respectively.

In Table 4, we test our method with the artificial data, which contain 50 sequences with average length 500. In the table, our method is examined with parameter θ from 1 to 10. When $\theta = 1$, our average computing time and score are almost the same as those of the Gibbs samplers method. The results of our method are slightly better than that of Gibbs sampler method. Because the testing data are artificial and unnatural, there are no characteristics and shared patterns in the sequences. Though the patterns of the artificial sequences are not similar, our method still has much improvement to the Gibbs sampler method in the computing time. When $\theta = 5$, our average computing time is 21% of that of the Gibbs sampler method with comparable quality.

We also test some other sets of artificial data and the results are shown in Table 5, where we set the parameter $\theta \doteq 5$. In the table, we can see that our method takes about 20% time of the Gibbs sampler method with comparable quality.

6 Conclusions

First, we apply ACO algorithm to find a set of better initial positions for the Gibbs sampler method. It makes the solution converge to a better solution quickly, and reduces the probability that the solution is stuck on a local optimal solution. We improve the uncertain of random choosing a set of beginning positions, and converge to a better solution. Second, we find that the Gibbs sampler method calculates the scores at each position on each sequence, and most of those calculations are useless. By reducing useless calculation at most positions, we improve the efficiency of the Gibbs sampler

method. The time required for finding motifs with our algorithm is reduced drastically.

Comparing in both in time and score, our method is faster and better than the Gibbs sampler method. In the experiments on real domains, our average computing time is 10% that of the Gibbs sampler method. Moreover, we make a better progress in the average score. An approximate progress is 20%. In the experiments on artificial domains, our average computing time is about 20% of that of the Gibbs sampler method with comparable quality.

References

[1] T. L. Bailey and C. Elkan, "Unsupervised learning of multiple motifs in biopolymers using expectation maximization," *Machine Learning*, Vol. 21, No. 1-2, pp. 51–80, 1995.

[2] J. Buhler and M. Tompa, "Finding motifs using random projections," *Journal of Computational Biology*, Vol. 9, No. 2, pp. 225–242, 2002.

[3] G. D. Caro and M. Dorigo, "Antnet: Distributed stigmergetic control for communications networks," *Journal of Artificial Intelligence Research (JAIR)*, Vol. 9, pp. 317–365, Dec. 1998.

[4] C.-H. Chu, J. Gu, X. D. Hou, and Q. Gu, "A heuristic ant algorithm for solving QoS multicast routing problem," *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pp. 1630–1635, 2002.

[5] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, pp. 53–66, 1997.

[6] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, Vol. 26, No. 1, pp. 29–42, 1996.

[7] Y.-J. Hu, S. B. Sandmeyer, and D. F. Kibler, "Detecting motifs from sequences," *In Proceedings of the 16th International Conference on Machine Learning (ICML)*, pp. 181–190, 1999.

[8] K. Karadimitriou and D. H. Kraft, "Genetic algorithms and the multiple sequence alignment problem in biology," *Proceedings of the Second Annual Molecular Biology and Biotechnology Conference*, pp. 1–7, 1996.

[9] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton, "Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment," *Science*, Vol. 262, pp. 208–214, Oct. 1993.

[10] C. E. Lawrence and A. A. Reilly, "An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences," *Proteins*, Vol. 7, No. 1, pp. 41–51, 1990.

[11] G. Leguizamon and Z. Michalewicz, "A new version of ant system for subset problems," *Proceedings of the Congress on Evolutionary Computation*, pp. 1459–1464, 1999.

[12] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, Vol. 77, pp. 257–282, 1989.

[13] E. Rocke and M. Tompa, "An algorithm for finding novel gapped motifs in DNA sequences," *Proceedings of the Second Annual International Conference on Computational Molecular Biology*, pp. 228–233, Mar. 1998.

[14] G. Stormo and G. Hartzell, "Identifying proteinbindign sites from unaligned DNA fragments," *Proc. Natl. Acad. Sci. USA*, Vol. 86, pp. 1183–1187, 1989.

[15] J. Stoye, "Multiple sequence alignment with the divide-and-conquer method," *Gene*, Vol. 211, No. 2, pp. 45–56, 1998.

Table 2: Experimental results of the real data in Table 1. 'Time(sec.)' is the average time for running the data 1000 rounds. 'Score' is the average of 1000 rounds. 'Variance' is the variance of scores. Performance = (our method) / (Lawrence's method)

Lawrences et al.					
-	Time(sec.)	Score	Variance	Performance(Time)	Performance(Score)
-	19.20	15.90	4.57	-	-
Our method					
θ	Time(sec.)	Score	Variance	Performance(Time)	Performance(Score)
1	9.23	18.82	0.16	48%	118%
2	6.51	18.75	0.16	34%	118%
3	4.87	18.68	0.14	25%	117%
4	4.00	18.55	0.16	21%	117%
5	3.36	18.48	0.17	17%	116%
6	3.04	18.48	0.20	16%	116%
7	2.74	18.30	0.21	14%	115%
8	2.50	18.27	0.21	13%	115%
9	2.38	18.16	0.25	12%	114%
10	2.17	18.09	0.21	11%	114%
11	2.02	18.06	0.21	10%	114%
12	1.93	18.00	0.24	10%	113%

Table 3: Experimental results of the real data in Table 1. The average computing time is the time when the method finds an acceptable solution. Performance = (our method) / (Lawrence's method)

	Our method	Lawrences et al.	
Score	Time(sec.)	Time(sec.)	Performance(Time)
$= 19.41$	88.40	421.50	21%
> 19.11	31.53	329.03	10%
> 19.09	28.17	282.82	10%
> 19.08	26.86	202.39	13%

Table 4: Experimental results of a set of artificial data, which contains 50 sequences with average length 500. Performance = (our method) / (Lawrence's method)

Lawrences et al.				
-	Time(sec.)	Score	Variance	Performance(Time)
-	102.75	12.05	0.047	-
Our method				
θ	Time(sec.)	Score	Variance	Performance(Time)
1	101.38	12.17	0.029	99%
2	57.69	12.16	0.041	56%
3	37.77	12.11	0.051	37%
4	27.46	12.07	0.039	27%
5	21.88	12.06	0.050	21%
6	17.59	11.96	0.046	17%
7	15.24	11.92	0.05	15%
8	13.07	11.85	0.032	13%
9	11.26	11.82	0.043	11%
10	10.66	11.85	0.056	10%

Table 5: Experimental results of some other sets of artificial data. Performance = (our method) / (Lawrence's method)

Sequences	Lawrence et al.		Our method		Performance
Num/Length	Time(sec.)	Score	Time(sec.)	Score	Time(%)
50/500	102.8	12.05	21.9	12.06	21%
50/1000	303.2	12.90	43.1	12.91	14%
50/1500	410.4	13.17	65.0	13.21	16%
70/500	245.1	10.55	48.2	10.57	20%
70/1000	551.6	11.26	108.4	11.30	20%
70/1500	740.3	11.62	130.4	11.62	18%
90/500	426.3	9.61	110.6	9.66	26%
90/1000	1000.3	10.33	181.5	10.34	18%
90/1500	1494.7	10.79	240.4	10.79	16%