IPSJ Transactions on Bioinformatics Vol. 2 63–73 (May 2009)

Original Paper

A Modified Algorithm for Sequence Alignment Using Ant Colony System

Ai Mikami[†1] and Jianming Shi[†1,*1]

In this study, we use the Ant Colony System (ACS) to develop a heuristic algorithm for sequence alignment. This algorithm is certainly an improvement on ACS-MultiAlignment, which was proposed in 2005 for predicting major histocompatibility complex (MHC) class II binders. The numerical experiments indicate that this algorithm is as much as 2,900 times faster than the original ACS-MultiAlignment algorithm. We also compare this algorithm to the other approaches such as Gibbs sampling algorithm using numerical experiments. The results show that our algorithm finds the best value prompter than Gibbs approach.

1. Introduction

Many algorithms in bioinformatics attempt to obtain an optimal alignment of a set of sequences. In most situations, such problems remain unsolved because they are $\mathcal{NP}$-hard in complexity. To obtain an optimal alignment, the algorithm should be designed so that the corresponding alignment has an optimal score. For this purpose, some evolutionary algorithms [1),4)] and the Gibbs sampling method [18)] have been proposed.

The Ant Colony System (ACS), introduced by Dorigo, et al. [8)], exploits a collective memory consisting of pheromone trails of ants try to obtain an optimal solution. The ACS can find a better solution to many combinatorial optimization problems, such as the traveling salesman problem (TSP) [6)–8)], the sequential ordering problem [10)], the vehicle routing problem [5)], and other [14),17),19)].

†1 Department of Computer Science and Systems Engineering, Muroran Institute of Technology
*1 Corresponding author. This author is partially supported by Grant-in-Aid for Scientific Research (No. 19510135) of the Ministry of Education, Science, Sports and Culture of Japan. E-mail: shi@mmm.muroran-it.ac.jp

Recently, an ACS-based algorithm named ACS-MultiAlignment was proposed [13)] to obtain a better alignment of a dataset of peptides of major histocompatibility complex (MHC) class II molecules. As pointed out by the authors in Ref. 13), their algorithm converges to better alignments and is one of the most advanced predictors currently in use. On the other hand, ACS-MultiAlignment is extremely time-consuming compared to the Gibbs sampling method. In this study, we attempt to improve the convergence speed and predictive accuracy of ACS-MultiAlignment.

The search method proposed in this paper shares a certain similarity with other approaches such as the Gibbs sampler in that all of these methods obtain a better alignment with the same objective function. The main difference between other evolutionary algorithms and our approach is that they use an explicit predicting evaluation in fitness function.

The purpose of this paper is twofold: (1) to modify ACS-MultiAlignment and (2) to verify the effectiveness of the proposed algorithm using numerical experiments.

The remainder of this paper is organized as follows. The conventional ACS algorithm is reviewed in Section 2. Section 3 is devoted to a modified algorithm for aligning sequences. In section 4, the results of numerical experiments are reported, and the new algorithm is evaluated by comparison with the ACS-MultiAlignment and the Gibbs method. Our conclusions are presented in the last section.

2. Conventional ACS Algorithm

ACS was originally proposed to solve the Traveling Salesman Problem (TSP). The idea behind ACS comes from the observation of the behavior of colonies of ants that almost always succeed in obtaining the shortest path from the nest to the food source by cooperating through pheromone trails.

Let us briefly review the ACS algorithm in the case of the TSP problem. Suppose that we are given a set of n cities $c_1, c_2, \ldots, c_n$ on an undirected graph. We look for a tour in which a traveling salesperson visits each of the cities exactly once so that he/she travels the minimum total distance. The key to the problem is how to choose the next city c_j from current city c_i at each step. The frame-

work of the ACS algorithm is as follows: Suppose that we are at the tth iteration. There exists a predetermined parameter $q_0 \in (0,1)$. Ant k at city c_i chooses the next city c_j according to the following rules: At the beginning, a real number q is randomly generated following a uniform distribution over $(0,1)$. If $q > q_0$, then ant k at city c_i chooses the next city c_j from the following set with the highest probability (see Eq. (2)):

$$\arg \max_{j \in N_k(t)} [\tau_{ij}(t)]^\alpha [\eta_{ij}]^\beta, \tag{1}$$

where $N_k(t)$ is the set of unvisited cities at iteration t, $\tau_{ij}(t)$ is the real-valued pheromone on the path between the cities c_i and c_j (or path (i,j) briefly), η_{ij} is defined as the reciprocal of the distance d_{ij} between cities c_i and c_j. The exponents α and β are predetermined parameters. If $q \le q_0$, ant k chooses the next city c_j with the following probability:

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_k(t)} [\tau_{il}(t)]^\alpha [\eta_{il}]^\beta} \qquad \text{for all} \quad j \in N_k(t). \tag{2}$$

After ant k moves from city c_i to city c_j, the pheromone on path (i,j) is updated according the following update rule:

$$\tau_{ij}(t) = (1-\rho)\tau_{ij}(t) + \rho \Delta\tau_0, \tag{3}$$

where $\Delta\tau_0$ and $\rho \in (0,1)$ are predetermined parameters. Once all the ants complete their tours, the current best tour among them can be selected. The pheromone on every path (i,j) of the current best tour is updated according the following global update rule:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho \Delta\tau_{ij}(t), \tag{4}$$

where $\Delta\tau_{ij}(t)$ is usually set to Q/D_t. The numerator Q is a parameter and the denominator D_t is the total distance of the current best tour. The ACS algorithm can be summarized as follows:

Algorithm ACSforTSP

`Step 1` set all parameters.

`Step 2` repeat until each ant k has completed a tour.

`for` each ant k $(k = 1, \ldots, m)$ do

select the next city j from current city i with probability $p_{ij}^k(t)$ of Eq. (1) and Eq. (2).

(local pheromone update)

update pheromone $\tau_{ij}(t)$ on path (i,j) according to Eq. (3).

`Step 3` (global pheromone updating)

update pheromone $\tau_{ij}(t+1)$ on each path of the current best tour according to Eq. (4).

`Step 4 go to Step 2` and repeat until the stop criteria is satisfied.

3. A Modified Algorithm for Sequence Alignment

In this section, we first review an ACS-based algorithm [13] designed for sequence alignment of MHC class II. Then, we consider a method to improve the algorithm.

Suppose that n sequences s_j $(j = 1, \ldots, n)$ are given. The length of s_j is denoted by $|s_j|$. Generally, molecules can bind peptides of very different lengths L. The different lengths of peptides further complicate the problem. There exist *class I* and *class II* in MHC molecules. In most situations, the peptide binding the motifs in MHC class II is assumed to have a fixed length of nine amino acids, that is $L = 9$. A detailed discussion on this assumption can be found in, for instance, Ref. 13).

We denote the i_jth nucleotide from the starting position in sequence s_j by s_{j,i_j}. The aim of this study is to determine a better starting point s_{j,i_j} for each sequence s_j.

For a starting point i_j in each sequence s_j, where $i_j + L - 1 \le |s_j|$, we define a binding core as a matrix M as follows:

$$M(s_{1,i_1}, \ldots, s_{n,i_n}) = \begin{pmatrix} s_{1,i_1} & s_{1,i_1+1} & \cdots & s_{1,i_1+L-1} \\ s_{2,i_2} & s_{2,i_2+1} & \cdots & s_{2,i_2+L-1} \\ & \cdots & & \\ s_{n,i_n} & s_{n,i_n+1} & \cdots & s_{n,i_n+L-1} \end{pmatrix}. \tag{5}$$

Note that M has L columns. If each starting point i_j for each sequence s_j is given, that is $(s_{1,i_1}, \ldots, s_{n,i_n})$ is given, then a unique matrix M can be determined. Therefore, solving the problem is equal to obtaining a locally optimal matrix M with an objective function. In bioinformatics, the objective function

associated with an alignment $(s_{1,i_1}, \ldots, s_{n,i_n})$ is defined by the fitness (energy) of the alignment as follows:

$$E(s_{1,i_1}, \ldots, s_{n,i_n}) = \sum_{j=1}^{L} \sum_{d=1}^{20} \{\text{FRQ}(d,j) \times \text{PSSM}(d,j)\}, \quad (6)$$

where $\text{FRQ}(d,j)$ is the occupancy number for amino acid d at position j in the alignment, i.e., column j in matrix M of Eq. (5). $\text{PSSM}(d,j)$ is defined as follows:

$$\text{PSSM}(d,j) = \ln \frac{Q(d,j)}{P(d)}, \quad (7)$$

where $P(d)$ is the background frequency of amino acid d, which is obtained from the SWISS-PROT database [3]. Altschul, et al. [2] calculated $Q(d,j)$ as follows:

$$Q(d,j) = \frac{aF(d,j) + bG(d,j)}{a+b}. \quad (8)$$

Here, $F(d,j)$ is the observed frequency, which can be calculated using the sequence weighting method [11], $G(d,j)$ is the pseudo-count frequency, a the mean number of different amino acids in the alignment, and b the weight on the pseudo-count correction. One can use Henikoff-Henikoff [12] sequence weighting scheme to calculate the weight. In this research, for the sake of comparison of efficiency of the algorithms we set $b = 50$ the same value as in Ref. 13). Detailed information about the calculation of Eq. (6)–Eq. (8) can be found in Ref. 1), 9), 11), 13), 16).

The reason for applying the ACS algorithm in this study is to exploit its excellent searching ability in order to maximize the objective $E(\cdot)$ of Eq. (6) over the set of all feasible alignments.

3.1 ACS-MultiAlignment Algorithm

The study by Karpenko, et al. [13] is the first to apply ACS to alignment of sequences of MHC class II molecules. The algorithm can be summarized as follows. As a pre-procedure, a dummy sequence with only one residue is placed at the top of the given sequence with index 0. All ants start from this dummy sequence and look for their suitable positions on sequence 1. After moving from sequence 0 to sequence 1, all ants look for suitable positions on the next sequence (sequence 2) from sequence 1, and continue doing so until they arrive at the last sequence, i.e., until they complete a tour. The position visited by an ant is considered the starting point of an alignment. Once a tour is completed, an alignment $(s_{1,i_1}, s_{2,i_2}, \ldots, s_{j,i_j}, \ldots, s_{n,i_n})$ is determined. Associated with this alignment, the pheromone on a path from position s_{j,i_j} on sequence j to position $u_{j+1,i_{j+1}}$ on sequence $j+1$ is defined by

$$\eta(s_{j,i_j}, u_{j+1,i_{j+1}}) = \frac{1}{n|E(s_{1,i_1}, s_{2,i_2}, \ldots, s_{j,i_j}, u_{j+1,i_{j+1}}, s_{j+2,i_{j+2}} \ldots, s_{n,i_n})|}. \quad (9)$$

Note that $E(s_{1,i_1}, s_{2,i_2}, \ldots, s_{j,i_j}, u_{j+1,i_{j+1}}, s_{j+2,i_{j+2}}, \ldots, s_{n,i_n})$ is considered the pseudo-distance between the two points. Thus, pheromone $\eta(s_{j,i_j}, u_{j+1,i_{j+1}})$ is given by the reciprocal of the pseudo-distance, which is analogous to η_{ij} used in Eq. (1), Eq. (2), and Step 2 in the ACSforTSP algorithm. Based on a similar discussion, an aligning algorithm named ACS-MultiAlignment was proposed in Ref. 13).

Initially, ACS-MultiAlignment randomly chooses an alignment for each ant k. A function value E_0^k associated with each alignment is calculated by Eq. (6). On each path, between two consecutive sequences s_j and s_{j+1}, the initial pheromone τ_0 is assigned as follows:

$$\tau_0 := \frac{1}{n|\max\{E_0^k | k \in \{1, \ldots, m\}\}|}. \quad (10)$$

The complete algorithm is listed below.

Algorithm ACS-MultiAlignment [13]

Step 1: form an initial feasible solution $(s_{1,i_1}^k, \ldots, s_{n,i_n}^k)$ for each ant k, $k = 1, \ldots, m$.
/* where i_j is the starting position in sequence s_j.*/
calculate the function value E_0^k. Set $\tau_0 := \frac{1}{n|E_0|}$,
where $E_0 := \max_k\{E_0^k\}$.
for each pair $(s_{j,i_j}^k, s_{j+1,i_{j+1}}^k)$, $j = 0, 1, \ldots, n-1$, $i_j \in PS_j$,
/* PS_j is a set of possible starting positions for each sequence j */
set $\tau(s_{j,i_j}, s_{j+1,i_{j+1}}) := \tau_0$.
set parameters ρ_l, ρ_g, p_0, and β.

Step 2: `for` each sequence j $(j = 0, 1, \ldots, n-1)$ `do`
`for` each ant k $(k = 1, \ldots, m)$ `do`
set $r := s_{j,i_j}^k$. /* r is the current position of ant k in

sequence j */
generate $p \in (0,1)$ randomly,
then, choose a starting position u in sequence j as follows.
`if` $p \le p_0$, compute $\max_{u \in PS_{j+1}}\{\tau(r,u) \times (\eta(r,u))^{\beta}\}$
and obtain u_0 such that
$\tau(r,u_0) \times (\eta(r,u_0))^{\beta} \in \arg\max_{u \in PS_{j+1}}\{\tau(r,u) \times (\eta(r,u))^{\beta}\}$;
`if` $p > p_0$, compute probabilities

$$p(r,u) = \frac{\tau(r,u) \times (\eta(r,u))^{\beta}}{\sum_{t \in PS_{j+1}} \tau(r,t) \times (\eta(r,t))^{\beta}} \text{ for each } u \in PS_{j+1},$$

and choose u_0 from PS_{j+1} with the above probabilities.
set $s^k_{j+1,i_{j+1}} := u_0$, and move ant k from sequence j to sequence $j+1$.

$$\tau(s^k_{j,i_j}, s^k_{j+1,i_{j+1}}) := (1-\rho_l)\tau(s^k_{j,i_j}, s^k_{j+1,i_{j+1}}) + \rho_l \frac{1}{n|E(s^k_{1,i_1}, \ldots, s^k_{n,i_n})|};$$

/* local pheromone update */
`end` of each ant k
`end` of each sequence j
move all ants back to sequence 0.

Step 3: compute $E_{\text{best}} := \max_{k=1,\ldots,m} E(s^k_{1,i_1}, \ldots, s^k_{n,i_n})$, and obtain an ant k_* such that $E(s^{k_*}_{1,i_1}, \ldots, s^{k_*}_{n,i_n}) = E_{\text{best}}$;
`for` each path from position $s^{k_*}_{j,i_j}$ to position $s^{k_*}_{j+1,i_{j+1}}$ $(j = 0, 1, \ldots, n-1)$ `do`

$$\tau(s^{k_*}_{j,i_j}, s^{k_*}_{j+1,i_{j+1}}) := (1-\rho_g)\tau(s^{k_*}_{j,i_j}, s^{k_*}_{j+1,i_{j+1}}) + \rho_g \frac{1}{n|E_{\text{best}}|}.$$

/*global pheromone update*/
`end` of each path

Step 4: `go to` **Step 2** and repeat several times.

Although ACS-MultiAlignment is a heuristic algorithm, it is easy to prove that for any given $\varepsilon > 0$ if the pheromone $\tau(\cdot,\cdot)$ in the algorithm is positive and bounded, say, $\tau(\cdot,\cdot) \in [b_l, b_u]$ with $1 > b_u \ge b_l > 0$, then the algorithm converges to ao optimal solution after $\left(\lceil \frac{\ln \varepsilon}{n \ln(1-b_l)} \rceil + 1\right)$ iterations with a probability $(1-\varepsilon)$ [13].

A computational performance and behavior of ACS-MultiAlignment has been reported in Ref. 13). According to that study, ACS-MultiAlignment is one of the most advanced predictors currently in use and the algorithm can converge to better alignments in practice. Despite these advantages, ACS-MultiAlignment still has some drawbacks. In fact, the algorithm is very time-consuming. This shortcoming might be a significant barrier to the practical use of the algorithm.

3.2 Modification of ACS-MultiAlignment Algorithm

ACS-MultiAlignment is considered to be a faithful imitation of ACS. The starting position in a sequence corresponds to a city; two starting positions in the consecutive sequences is analogous to a path between two cities in TSP. The pheromone in TSP is dropped on a path between two cities, because the aim of the algorithm is to determine the best route that consists of the paths. In this study, we are going to obtain a better alignment, which is determined by starting positions of the sequences.

In ACS-MultiAlignment, a path between two points in two consecutive sequences is constructed to establish a route and the pheromone is placed on this path. Suppose that the pheromones are to be placed in two sequences s_{j_1} and s_{j_2}. The number of feasible positions are $(|s_{j_1}| - L + 1)$ in sequence s_{j_1} and $(|s_{j_2}| - L + 1)$ in sequence s_{j_2}. Note that the total number of feasible paths between the two sequences s_{j_1} and s_{j_2} are $(|s_{j_1}| - L + 1) \times (|s_{j_2}| - L + 1)$ [13]. Thus, determining a better path between sequences s_{j_1} and s_{j_2} implies searching for the one in $(|s_{j_1}| - L + 1) \times (|s_{j_2}| - L + 1)$ feasible paths.

However, an alignment is decided from the starting positions in the sequences. That is, an alignment is not necessarily represented by a path between two consecutive sequences. The fashion in ACS-MultiAlignment might not be efficient to construct a path of two starting positions and assign the pheromone on the path. In our new method, the pheromone is directly delivered to the starting positions. This modification reduces the computational time of assigning pheromones and decreases the number of the candidate paths and routes in search space.

In the new algorithm in this study, the pheromone is directly assigned at the starting position. Thus, the number of feasible starting positions between the two sequences s_{j_1} and s_{j_2} is $(|s_{j_1}| - L + 1) + (|s_{j_2}| - L + 1)$. Note that $(|s_{j_1}| - L + 1) + (|s_{j_2}| - L + 1)$ is generally much less than $(|s_{j_1}| - L + 1) \times (|s_{j_2}| - L + 1)$ in the problem.

In ACS-MultiAlignment, the value of η_{ij} in Eq. (1) is taken as the reciprocal of the pseudo-distance suggested in ACS. To enhance the efficiency of calculation, the value η_{ij} in the modified algorithm is calculated by the ratio of likelihoods as follows:

$$\eta(u_{j+1,i_{j+1}}) = \frac{\prod_{l=1}^{L} F(u_{j+1,i_{j+1}+l-1}, l)}{\prod_{l=1}^{L} P(u_{j+1,i_{j+1}+l-1})}, \tag{11}$$

where $u_{j+1,i_{j+1}+l-1}$ stands for the lth peptide from position i_{j+1} in sequence s_{j+1}.

The parameters m and ρ in expression of Eq. (3) are important to obtaining a better solution. Those two parameters are also sensitive to convergence of the algorithm. A wrong setting of the parameters may put obstacles in convergence. To circumvent this problem, we change expression of Eq. (3) of a local pheromone update as follows:

$$\tau_{ij}(t) = (1 - \rho\gamma^{\omega})\tau_{ij}(t) + \rho\gamma^{\omega}\Delta\tau_0 \tag{12}$$

where γ is randomly chosen from $(0, 1)$ and ω is the number of iterations in which the algorithm has not found a current best solution. Based on the above discussion, we now propose a modified algorithm under the name ACS-SearchAlignment as follows:

Algorithm ACS-SearchAlignment

Step 1: form an initial feasible solution $(s^k_{1,i_1}, \ldots, s^k_{n,i_n})$ for each ant k $(k = 1, \ldots, m)$.
/* where i_j is the starting position in sequence s_j */
calculate the function value E^k_0. set $\tau^k_0 := \frac{E^k_0}{n}$, $E_{best} = -\infty$, $\omega = 0$.
`for` all s^k_{j,i_j}, $j = 1, \ldots, n$, $i_j \in PS_j$ `do` set $\tau(s^k_{j,i_j}) = \tau^k_0$.
/* PS_j is a set of possible starting positions for each sequence j */
set parameters α, ρ, p_0 and β.

Step 2: randomly `choose` one sequence j from $j \in \{1, \ldots, n-1, n\}$ `do`
`for` each ant k $(k = 1, \ldots, m)$ `do`
generate $p \in (0, 1)$ randomly, then, choose a starting position u for position j as follows:
`if` $p \le p_0$, compute $\max_{\in PS_j} \tau(u)^{\alpha} \times (\eta(u))^{\beta}$ and obtain u_0 such that $\tau(u_0)^{\alpha} \times (\eta(u_0))^{\beta} = \arg\max_{\mu \in PS_j} \tau(u)^{\alpha} \times (\eta(u))^{\beta}$, where $\eta(u)$ is calculated by (11);
`if` $p > p_0$, compute probabilities $\dfrac{\tau(u)^{\alpha} \times (\eta(u))^{\beta}}{\sum_{i_j \in PS_j} \tau(u)^{\alpha} \times (\eta(u))^{\beta}}$ and choose u_0 from the above probabilities.
set $s^k_{j,i_j} := u_0$.
$\tau(s^k_{j,i_j}) := (1 - \rho\gamma^{\omega})\tau(s^k_{j,i_j}) + \rho\gamma^{\omega}\tau_0$;
/* local pheromone update */
`if` $i < n$, move ant k from sequence j to sequence $j + 1$;
`if` $i = n$, move ant k from sequence j to sequence 1.
`end` of each ant k
`end` of choose sequence j

Step 3: compute $E_{current} := \max_{k=1,\ldots,m} E(s^k_{1,i_1}, \ldots, s^k_{n,i_n})$,
`if` $E_{current} > E_{best}$, $E_{best} := E_{current}$;
obtain an ant k_* such that $E(s^{k_*}_{1,i_1}, \ldots, s^{k_*}_{n,i_n}) = E_{current}$;
for all of $s^{k_*}_{j,i_j}$ `do`
$\tau(s^{k_*}_{j,i_j}) := (1 - \rho)\tau(s^{k_*}_{j,i_j}) + \rho E_{best}$.
`else` $\omega = \omega + 1$
/* global pheromone update */

Step 4: `go to` **Step 2** and repeat several times.

In this algorithm, at Step 2, a sequence is randomly chosen. We allow the same sequence to be chosen more than twice, because the alignment when the sequence is chosen the first time is usually different from the alignment when the same sequence is chosen the second time. Thus, two different alignments associated with the same sequence have different objective function values in most cases. In our method, both the local and global pheromone are updated in a different way to ACS-MultiAlignment in which the pheromone on a path is increased

by local update at Step 2. Conversely, in our algorithm the pheromone is reduced in the local updating procedure because, usually $\tau_0 < \tau(s_{j+1,i_{j+1}})$. The global pheromone is updated only if $E_{current}$ is equal to the maximum during the calculation.

4. Numerical Experiments

We report a computational experimental study of the proposed algorithm in this section. The experiments were conducted on a Windows PC with 2.19 GHz, and 248 MB RAM at Department of Computer Science and Systems Engineering, Muroran Institute of Technology.

We first present the experimental results with a small-scale dataset. Second, we discuss the issue of parameter setting. Then, we make a comparison between ACS-MultiAlignment and our method, and between the Gibbs sampling method and our method by using the datasets used in Ref. 13). Finally, we evaluate the performance of our algorithm using several benchmark datasets.

4.1 Results with a Small-Scale Dataset

ACS-SearchAlignment was tested with a dataset consisting of only five sequences. From the results, it was confirmed that our algorithm ACS-SearchAlignment worked very well. **Table 1** shows the input and output in detail.

The optimal value of problem is 32.6366 in units under the objective function $E(\cdot)$ of Eq. (6). The parameters we used here were $m = 10, \alpha = 1.8, \beta = 0.5, \rho = 0.8, p_0 = 0.1$. The algorithm was implemented five times. The optimal value of 32.6366 is obtained from the ACS-SearchAlignment.

4.2 Parameter Setting

Some parameters in ACS are fundamentally important for the performance of the algorithm [13]. The parameter ρ is considered to control the volatile quantity of pheromone at each step. According to this study, the convergence of the algorithm is mainly affected by the values of ρ and β. The dataset we employed was the same as that used in Ref. 13), which has 67 sequences. In this study, unless noted otherwise the parameters for ACS-SearchAlignment were set to the values $m = 10, \alpha = 1.8, \rho = 0.5, p_0 = 0.1$, while $\beta = 0.4$ to 0.6. The parameter b in the definition of $Q(i, j)$ in Eq. (8) was set to $b = 50$.

Parameters in ACS are usually sensitive. The following performance of the algorithms is observed: Increasing the values m and ρ may result in much more time to convergence, and a better value of the objective function. A too small or too big value of parameter β can lead a bad objective value .

Table 1 An optimal alignment obtained by our algorithm (small-scale case).

```
FVRFDSDAASQRMEP                    FVRFDSDAA SQRMEP
VDDTQFVRFDSDAASQRM                VDDTQ FVRFDSDAA SQRM
FVRFDSDAASQRM          ⇒           FVRFDSDAA SQRM
VDDTQFVRFDSDAASPRGEP              VDDTQ FVRFDSDAA SPRGEP
DGKDYIALNEDLSS                     DGKD YIALNEDLS S
```

4.3 Comparison between ACS-MultiAlignment and Our Method

In this section, we compare the behavior of ACS-MultiAlignment and our algorithm using numerical experiments. Both algorithms were implemented with a dataset which was extracted from a benchmark and used in Ref. 13). The algorithms were implemented 600 iterations in this experiment.

The parameter values for ACS-MultiAlignment were set to the same as suggested in Ref. 13), which are believed to be the best settings according to the study. Note that in both algorithms the number of ants in each iteration was 3, i.e., $m = 3$.

Table 2 makes a comparison between ACS-MultiAlignment and ACS-SearchAlignment in CPU time and objective function values. The experimental results in this table show that our algorithm is much faster than ACS-MultiAlignment. The approximation functions of CPU time for both algorithms were linear to the number of iterations, which were $T(n) = 0.044n$ (s) for ACS-SearchAlignment and $T(n) = 123n$ (s) for ACS-MultiAlignment.

Here n stands for the number of iteration. Thus, in that sense ACS-SearchAlignment might be estimated about 2,900 times faster than ACS-

Table 2 A comparison between ACS-MultiAlignment and ACS-SearchAlignment.

number of iterations		100	200	300	400	500	600
ACS-MultiAlignment	value	−1338.3	−1314.7	−1314.7	−1308.1	−1308.1	−1308.1
	time	12392	24701	37029	49349	66162	73990
ACS-SearchAlignment	value	−1337.1	−1312.5	−1308.2	−1308.2	−1308.2	−1308.2
	time	4.6410	8.9220	13.1880	17.4540	21.7350	25.9540

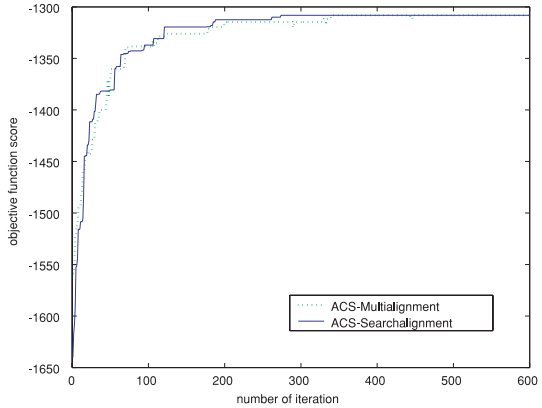

Fig. 1 Behavior of ACS-MultiAlignment and ACS-SearchAlignment.

MultiAlignment.

Figure 1 depicts that both the algorithms have a similar behavior with respect to the iterations.

4.4 Comparison between Gibbs Method and Our Method

The *Gibbs sampling method* was introduced into bioinformatics by Lawrence, et al. [15], and is one of the most successful tools used to identify motifs and alignments in sequences. Extensive comparisons have been made between the Gibbs sampling method and other methods. Nielsen, et al. [18] reported that the Gibbs method has a better ability to predict motifs than other methods according to their numerical experiments on benchmark datasets.

In this section, instead of directly comparing with other methods, we only compare our ACS-SearchAlignment and the Gibbs sampling method. A dataset used for this comparison was the same as that used in Ref. 13). The total number of iterations for each implementation was set up to 1,000.

There are two distinct Monte Carlo moves used in Gibbs algorithm. In the first, a new start point for the alignment of a sequence is randomly selected. The alignment of a sequence is shifted a random number of positions. This move is called the *single sequence move*. The second is *the phase shift move*: a window of the alignment is shifted a random number of residues to the left or right.

In this study, we used a combination of these two moves. In *the phase shift move*, a feasible range of the shift move is decided by the length of the shortest sequence. Therefore, when the length of a sequence is too short, then the feasible range of a shift move will be too small to move. To implement a phase shift move well, we randomly divide the sequences into some smaller partitions, and each partition was independently shifted.

Table 3 Variation of the maximum function value with different r values.

value of r	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Average	-1184.6	-1200.1	-1216.9	-1185.9	-1225.9	-1218.4	-1218.7	-1193.8	-1205.1
Max.	-1182.4	-1199.5	-1196.5	-1159.1	-1194.1	-1196.3	-1192.8	-1184.6	-1159.4

In Gibbs sampling, the probability p_g of accepting a path between the current and the immediately previous move is given as

$$p_g = \min\left\{1, e^{\frac{\Delta E}{T}}\right\}, \tag{13}$$

where ΔE is the energy difference between the current and the immediately previous move and T is a scalar, which indicates the current temperature of the system; T decreases during the implementation. In this study, T was lowered according to the equation

$$T_k = \max\{rT_{k-1}, 10^{-100}\}, \tag{14}$$

where $r < 1$, T_0 is a predetermined parameter.

To choose a suitable value of r, the Gibbs algorithm was implemented five times with each value of $r = 0.1, 0.2, \ldots, 0.9$.

Table 3 shows the maximum function value and an average of the maximum function values in the five implementations.

No significant difference between the maximal values was observed when r was varied from 0.1 to 0.9. For $r = 0.9, 0.99, 0.999, 0.9999$, the four average values for five implementations of the Gibbs sampling method are plotted in **Fig. 2**. These outcome indicate that when the value of r is set to 0.9999, the Gibbs sampling method finds the best solution.

Relatively, the values of r between 0.8 to 0.9 were believed to be better for obtaining the maximum function value. Thus, we set the value of r to $r = 0.8, 0.9, 0.9999$ in this comparison.

For each value of $\beta = 0.4, 0.5, 0.6$ the ACS-SearchAlignment was implemented five times. For each value of $r = 0.8, 0.9, 0.9999$, the Gibbs sampling

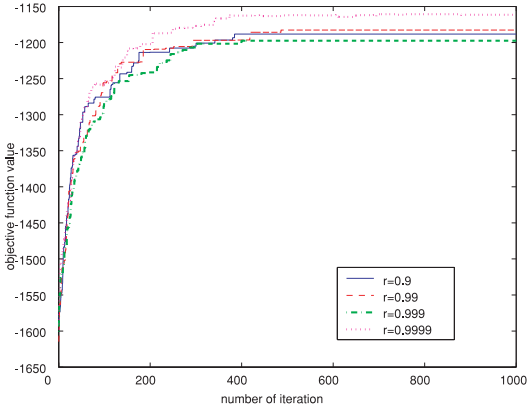

Fig. 2 Situation of different r values: from 0.9 to 0.9999.

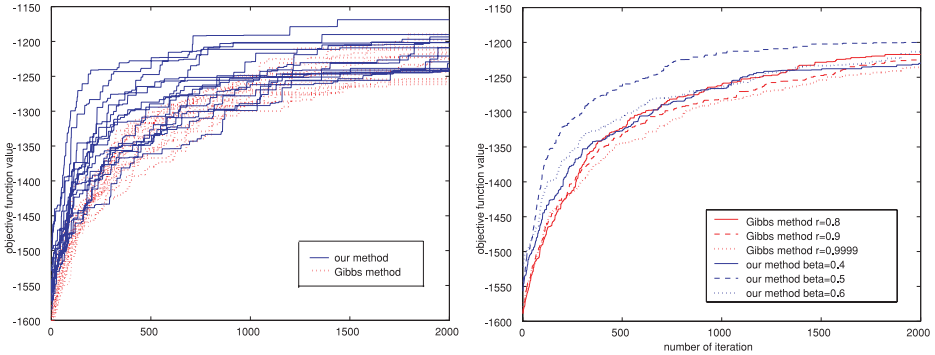

Fig. 3 Objective function values obtained at each iteration.

algorithm was also implemented five times.

In the left panel of **Fig. 3** the 30 curves are the results of the 30 implementations. The 15 dotted-dashed (blue) curves demonstrate the calculation results for ACS-SearchAlignment, while the 15 solid (red) curves present the results of Gibbs sampling. It can be seen that the 15 dotted-dashed curves almost dominate the 15 solid curves between iteration 0 to iteration 400. This indicates that ACS-SearchAlignment obtained a better solution faster than the Gibbs sampling method.

Table 4 An average CPU time (s) for five implementations, parameter r was set to 0.9 for Gibbs and β was set to 0.5 for our method.

number of iterations	200	400	600	800	1000
Gibbs sampling	19.3020	36.6880	52.1250	75.8130	92.4370
ACS-SearchAlignment	25.8280	50.5160	76.9370	103.3750	127.2970
number of iterations	1200	1400	1600	1800	2000
Gibbs sampling	118.6720	132.0360	144.8120	160.1090	194.7190
ACS-SearchAlignment	151.0310	180.5310	206.5020	229.4850	259.1560

Table 5 Five IEDB datasets. The parameter r was set to 0.9999 for Gibbs and β was set to 0.6 for ACS-SearchAlignment.

	total			Our method		Gibbs method	
	iterations	*length*	*avg range*	*max value*	*iteration*	*max value*	*iteration*
Human Immunodeficiency virus	2000	64	15.5	-1126.4	481	-1161.1	1316
Hepatitis B virus	2000	47	15.7	-772.8	803	-812.9	1525
Hepatitis C virus	2000	127	16.9	-2624.9	799	-2609.6	1592
Homo Sapiens	2000	188	15.6	-3561.5	1353	-3614.1	1651
Plasmodium Falciparum	2000	160	18.2	-4074.4	1694	-4072.7	1962

In the right panel of Fig. 3 each curve shows the average values of each five implementations. It can be seen that the ACS-SearchAlignment obtained a better average value faster than the Gibbs sampling.

Table 4 exhibits an average CPU time (seconds) in the five implementations. It was observed that our algorithm ACS-SearchAlignment algorithm spends slightly more time than the Gibbs sampling method for each single iteration in the experiments.

4.5 Evaluation of ACS-SearchAlignment on IEDB

To investigate the behavior of our ACS-SearchAlignment algorithm in detail, we extracted five independent datasets from IEDB [20] and performed further numerical experiments.

ACS-SearchAlignment was implemented once for each dataset with $\beta = 0.6$ and the Gibbs sampling algorithm was also implemented once for each dataset with $r = 0.9999$.

Table 5 shows the maximum value obtained by ACS-SearchAlignment and the Gibbs sampling method for each dataset. The numbers in the *total iterations* column are, as the name suggests, the total number of iterations in the implementation. The numbers in the *max value* column indicate the maximum values of objective function, obtained from the corresponding algorithm through

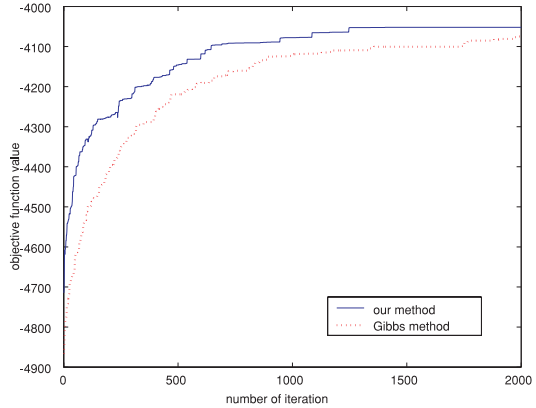

Fig. 4 Behavior of ACS-SearchAlignment and Gibbs sampling: Plasmodium Falciparum case.

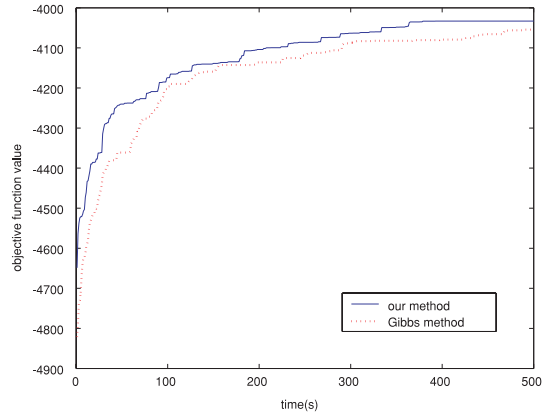

Fig. 5 Behavior of ACS-SearchAlignment and Gibbs sampling: Plasmodium Falciparum case.

three implementations. The numbers in the *iteration* column mark the iteration number when the algorithm obtained the maximum for the first time during the calculation.

Figure 4 shows the behavior of the algorithms in 2000 iterations in detail for dataset Plasmodium Falciparum, the one with largest size in the experiments.

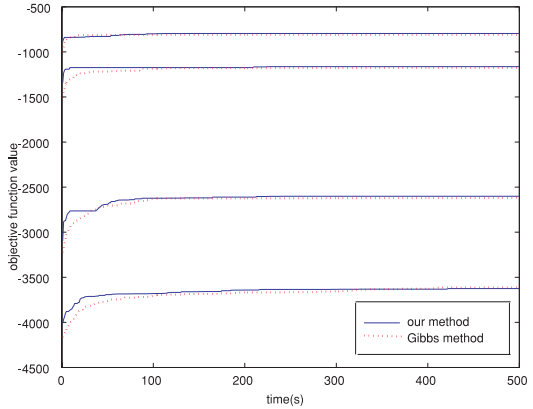

Fig. 6 Behavior of ACS-SearchAlignment and Gibbs sampling: From data1 to data4 case.

ACS-SearchAlignment obtained the maximum of -4074.4 at iteration 1694, while the Gibbs algorithm obtained the maximum of -4072.7 at iteration 1962.

Next we discuss the issue of execution time for finding a better solution. We run the algorithms for 500 seconds with each dataset, and tracked the change of the values of objectibe function.

Figure 5 shows the situation in detail for dataset Plasmodium Falciparum. The results for the other datasets are showen in **Fig. 6**. The horizontal axis is for CPU time and the vertical axis is for the incumbent best values of the objective function.

The setting of parameters β and r were same as in Table 5. For the other datasets in Table 5 we also conduced the experiments and obtained similar results on execution time. Consequently, these results show that our algorithm finds a better solution faster than Gibbs algorithm.

5. Conclusion

We have proposed an ACS-based algorithm, ACS-SearchAlignment, for identifying alignment in sequences. Numerical experiments indicate that:

1. ACS-SearchAlignment is as much as 2,900 times faster than the previous version, ACS-MultiAlignment [13],
2. ACS-SearchAlignment obtains a better alignment prompter than the

Gibbs sampling method in time.

Therefore, according to the experiments in this study, ACS-SearchAlignment is promising to be as good as the Gibbs sampling method, which is one of the most competent predictors currently in use.

References

1) Abbas, A.E. and Holmes, S.P.: Bioinformatics and management science: some common tools and techniques, *Oper. Res.*, Vol.52, pp.165–190 (2004).
2) Altschul, S., Madden, T. Schaffer, A., Zhang, J., Zhang, Z., Miller, W. and Lipman, D.: Gapped BLAST and PSI-BLAST: A new generation of protein database search programs, *Nucleic Acids Research*, Vol.25, pp.3389–3402 (1997).
3) Bairoch, A. and Apweiler, R.: The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000, *Nucleic Acids Research*, Vol.28, pp.45–48 (2000).
4) Brusic, V., Rudy, G., Honeyman, M.C., Hammer, J. and Harrison, L.C.: Prediction of MHC class-II binding peptides using an evolutionary algorithm and artificial neural network, *Bioinformatics*, Vol.14, pp.121–130 (1998).
5) Bullnheimer, B., Hartl, R.F. and Strauss, C.: An improved ant system algorithm for the vehicle routing problem, *Annals of Operations Research*, Vol.89, pp.319–328 (1999).
6) Dorigo, M. and Gambardella, L.M.: Ant colonies for the traveling salesman problem, *BioSystems*, Vol.43, pp.73–81 (1997).
7) Dorigo, M. and Gambardella, L.M.: Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem, *IEEE Transactions on Evolutionary Computation*, Vol.1, pp.53–66 (1997).
8) Dorigo, M., Maniezzo, V. and Cambini, A.C.: Ant system: Optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man, and Cybernetics—Part B*, Vol.26, pp.29–41 (1996).
9) Ewens, W.J. and Grant, G.R.: *Statistical Methods in Bioinformatics*, Springer (2001).
10) Gambardella, L.M. and Dorigo, M.: HAS-SOP: A hybrid ant system for the sequential ordering problem, *Technical Report*, No.IDSIA 97-11, IDSIA, Lugano, Switzerland (1997).
11) Henikoff, S. and Henikoff, J.G.: Amino acid substitution matrices from protein blocks, *Proc. National Academy of Sciences USA*, Vol.89, pp.10915–10919 (1992).
12) Henikoff, S. and Henikoff, J.G.: Position-based sequence weights, *Journal of Molecular Biology*, Vol.243, pp.574–578 (1994).
13) Karpenko, O., Shi, J. and Dai, Y.: Prediction of MHC class II binders using the ant colony search strategy, *Artificial Intelligence in Medicine*, Vol.35, pp.147–156 (2005).
14) Korošec, P., Šilc, J. and Robič, B.: A multilevel ant-colony optimization algorithm for mesh partitioning, *International Journal of Pure and Applied Mathematics*, Vol.5, pp.143–159 (2003).
15) Lawrence, C.E., Altschul, S.F., Bogouski, M.S., Liu, J.S., Neuwald, A.F. and Wooten, J.C.: Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment, *Science*, Vol.262, pp.208–214 (1993).
16) Lund, O., Nielsen, M., Lundegaard, C., Kesmir, C. and Brunak, S.: *Immunological Bioinformatics*, The MIT Press, 2005.
17) Maniezzo, V. and Colorni, A.: The ant system applied to the quadratic assignment problem, *IEEE Transactions on Knowledge and Data Engineering*, Vol.11, pp.769–778 (1999). Bench, http://www.imtech.res.in/raghava/mhcbench/
18) Nielsen, M., Lundegaard, C. and Worning, P., et al.: Improved prediction of MHC class I and class II epitopes using a novel Gibbs sampling approach, *Bioinformatics*, Vol.20, pp.1388–97 (2004).
19) Tkindt, V., Monmarche, N., Tercinet, F. and Laügt, D.: An ant colony optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem, *European Journal of Operational Research*, Vol.142, pp.250–257 (2002).
20) IEDB database, http://www.immuneepitope.org/

(Received February 8, 2009)
(Accepted March 6, 2009)
(Released May 25, 2009)

(Communicated by *Junichiro Yoshimoto*)

Ai Mikami earned her master's degree from Muroran Institute of Technology in 2009. She is interested in Computational methods and its applications in bioengineering. She is now with HBA Corporation in Sapporo.

Jianming Shi is an associate professor of Department of Computer Science and Systems Engineering, Muroran Institute of Technology. He earned his Ph.D. degree from University of Tsukuba in Systems and Applied Mathematics. His research interest includes Mathematical Programming, Global Optimization and their applications. His papers have appeared in such as Journal of Global Optimization, Annals of Operations Research, Artificial Intelligence in Medicine, Journal of Computational and Applied Mathematics and others.