

Partitioned optimization algorithms for multiple sequence alignment

Yixin Chen¹ Yi Pan² Ling Chen³ Juan Chen³

¹ Department of Computer Science, Washington University in St. Louis, St. Louis, MO 63130, USA

² Department of Computer Science, Georgia State University, Atlanta, GA 30303, U.S.A

³ Department of Computer Science, Yangzhou University, Yangzhou 225009, China)

Abstract Multiple sequence alignment is an important and difficult problem in molecular biology and bioinformatics. In this paper, we propose a partitioning approach that significantly improves the solution time and quality by utilizing the locality structure of the problem. The algorithm solves the multiple sequence alignment in three stages. First, an automated and suboptimal partitioning strategy is used to divide the set of sequences into several subsections. Then a multiple sequence alignment algorithm based on ant colony optimization is used to align the sequences of each subsection. Finally, the alignment of original sequences can be obtained by assembling the result of each subsection. The ant colony algorithm is highly optimized in order to avoid local optimal traps and converge to global optima efficiently. Experimental results show that the algorithm can significantly reduce the running time and improve the solution quality on large-scale multiple sequence alignment benchmarks.

Keywords: bioinformatics, multiple sequence alignment, partitioning, ant colony optimization, genetic algorithm

1. Introduction

Multiple sequence alignment (MSA) is an important task in bioinformatics. Simultaneous alignment of many nucleotide or amino acid sequences is an essential tool in molecular biology and bioinformatics, and it plays a key role in detecting regions of significant sequence similarity from collections of primary sequences of nucleic acid or proteins. MSA can also be used to support the reconstruction of phylogenetic trees, find patterns to characterize protein families, detect homology between new sequences and existing ones, and predict the secondary and tertiary structure of protein sequences.

The basis of sequence alignment and MSA is evolutionary theory. According to this theory, during the course of evolution, mutations occur, and differences are created among families of species. Most of these changes are due to local mutations which consist of three operations including insertion (inserting certain letters to the sequences), deletion (deleting certain letters from the sequences), and substitution (replacing a letter by another).

Given two sequences X and Y , a pair-wise alignment indicates positions of each sequence that are considered to be functionally or evolutionarily related. Given a family $S = (S_1, \dots, S_N)$ of N sequences, we would like to find out the common patterns of this family. Since aligning each pair of sequences from S separately often does not reveal the common information, it is necessary to perform multiple sequence alignment (MSA). Within an MSA, preserved motifs often appear as

columns with a much lower degree of variation than their surroundings.

MSA is naturally formulated as an optimization problem where an objective function from biological principles is used. The problem is a nonlinear optimization problem defined in a discrete space which is computationally very expensive to solve. We have made a key observation that, in most applications, the aligned sequences are usually not randomly distributed but demonstrate strong locality. This motivates us to utilize the problem structure by partitioning the overall alignment problem into multiple subproblems and solving them individually before composing the overall solution.

In this paper, based on the key observation of structural locality of aligned sequences, we propose an efficient partitioned optimization algorithm for MSA. The algorithm solves the multiple sequence alignment in three stages. First, an automated partitioning strategy is used to divide the set of sequences into several subsections while approximately preserving the optimality. Then, a multiple sequence alignment algorithm based on ant colony optimization is used to align the sequences of each subsection. Finally, an alignment of the original sequences is obtained by assembling the results from multiple partitions. Experimental results show that the algorithm can significantly reduce the running time and improve the solution quality.

2. Multiple sequence alignment: definition and previous work

Given a family $S = (S_1, \dots, S_N)$ of N sequences, an MSA of S is a new set of sequences

$S' = (S'_1, \dots, S'_N)$ such that all the strings in S' are of equal length and each S'_i is generated from S_i by inserting gaps. An example of multiple sequence alignment is shown in Figure 1, where “-“ stands for an inserted gap.

```

-GRRRSVQWCAVSNPEATKCFQWQRNMRKVR---GPPVSCLKRDSPIQCIQAI
----KTVRWCAVNDHEASKCANFRDSMKKVLPEDGPRIICVKKASYLDCIKAI
-----VKWCVKSEQELRKCHDLAAKVAE-----FSCVRKDGSEFECIQAI
--KEKQVRWCVKSNSELKKCKDLVDTCKNK----EIKLSCVEKSNTDECSTAI
-----EVRWCATSDPEQHKCGNMSEAFREAGI--QPSLLCVRGTSADHCVQLI
APPKTTVRWCTISSAEKKCNLSLKDHMQER----VTLSCVQKATYLDICIKAI

```

Figure1. An example of multiple sequence alignment

When performing MSA, we wish to evaluate the quality of the alignment by giving it a numerical score. The SP (sum-of-pairs) function is the most popular scoring method for MSA in bioinformatics. The SP score of an alignment S is the sum of the scores of pair wise global alignments induced by S . If the objective function is additive, the SP score of S is just the sum of the scores of all aligned columns. The score of a column is computed using the formula below:

$$SP - Score(c_1, c_2, \dots, c_N) = \sum_{i=1}^{N-1} \sum_{j=i+1}^N match(c_i, c_j), \quad (1)$$

where c_i represents the i -th character of this column and $match(c_i, c_j)$ denotes the comparing score between c_i and c_j .

The goal of general multiple sequence alignment algorithms is to find out the alignment with the highest *SP* score. Since the computational cost for exact global optimization will typically be exceedingly expensive when there are more than three sequences, most existing MSA algorithms are heuristic and cannot guarantee global optimality.

MSA can be viewed as an extension of the two-sequence alignment problem. The Needleman and Wunsch algorithm [1] is a classical dynamic programming algorithm for pair-wise sequence alignment. Both time and space cost of the algorithm are $O(mn)$, where m and n are the length of two given sequences. Unfortunately, to align several sequences simultaneously, one will need to generalize the Needleman and Wunsch algorithm to a multidimensional space, where the

space and time cost will grow exponentially as $O(2^N - 1)(\prod_{i=1}^N |S_i|)$, where N is the number of

sequences and $|S_i|$ is the length of the i -th sequence. In fact, MSA with the *SP* (sum-of-pairs) score is an NP-hard problem [27], and the dynamical programming method is only practical for a maximum of three sequences.

The MSA program [2], an implementation of the Carrillo and Lipman algorithm [5] which identifies in advance the portion of the hyperspace that does not contribute to the solution and excludes it from computation, can align up to ten closely related sequences. Stoye described a new divide and conquer algorithm DCA [3,4] that sits on the top of MSA and extends its capabilities.

Progressive alignment is a widely used heuristic MSA method that does not guarantee any level of optimality [7]. ClustalW [6] is a popular program that improved the algorithm presented by Feng and Doolittle [7]. The main shortcoming of ClustalW is that once a sequence has been aligned, that alignment can never be modified even if it conflicts with sequences added later. Dialign [10,11] is an other implementation of the progressive alignment method which assembles the alignment in a sequence-independent manner by combining segment pairs in an order dictated by their scores until the residue of every sequence has been incorporated in the alignment.

Iterative method is another classical technique for MSA. It refines the alignment through a series of cycles until no more improvements can be made. Example implementation of the iterative method includes Prpp [8], AMPS [25], Berger and Munsen's algorithm [26], SAGA [9], Gibbs's sampling algorithm [24], MSA algorithm based on simulated annealing [22], T-Coffee [17], MUSCLE [18], and PROBCONS [20].

Other MSA algorithms include Hidden Markov Model (HMM) [12], the partial order graph method [13], MAFFT, a MSA algorithm based on fast Fourier transform [14], and block-based algorithms [15]. A survey of recent progress in MSA can be found in [21].

3. Automated partitioning strategy for MSA

The general idea of our approach is to partition the overall problem into smaller subproblems that are exponentially easier to solve. Suppose we are given a family $S = (S_1, \dots, S_N)$ of N sequences, we partition the sequences into several subsets of segments vertically. If we cut each sequence S_i at a suitable position c_i near to the midpoint, we obtain two new families of shorter sequences, one

family consisting of the prefixes $S^p = (S_1^p(c_1), \dots, S_N^p(c_N))$ and one of suffixes $S^s = (S_1^s(c_1), \dots, S_N^s(c_N))$. If we can align these two new families of sequences optimally, we can concatenate our resulting alignments to obtain an alignment of the original sequences. Such a partitioning method can be applied recursively to reduce the original problem to multiple smaller MSA problems until the lengths of the subsequences are all less than an acceptable threshold. Figure 2 gives a schematic representation of the partitioning approach for three sequences.

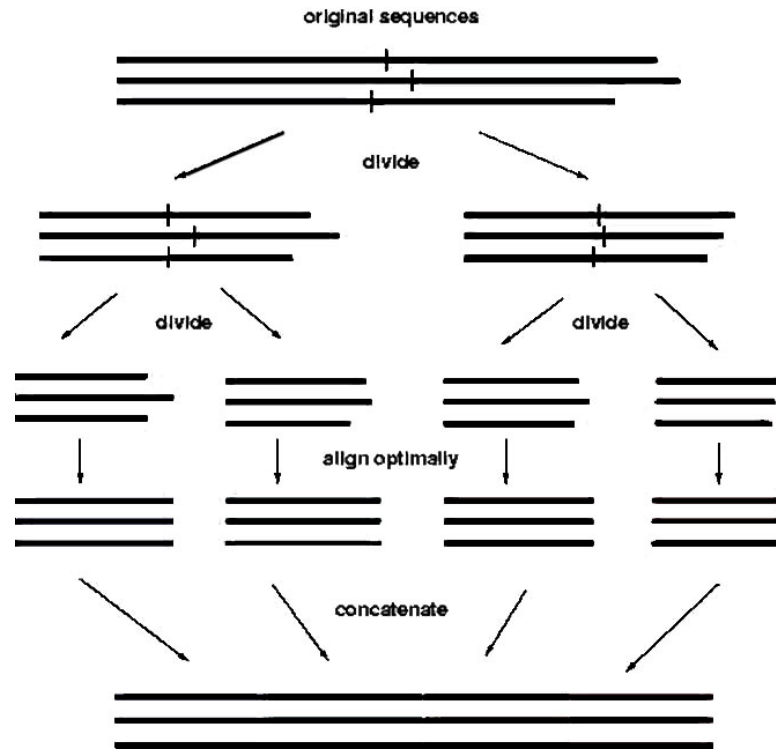


Figure2. The scheme of the algorithm

However, since an arbitrary partitioning of the original problem usually leads to degradation of the quality of the composed solution, a key issue for the partitioning strategy is to find optimal cut-off positions to minimize the loss of solution quality due to partitioning.

3.1 Estimation of the matching score

In each iteration of the partitioning method, we partition the sequences into several subsets of segments vertically. For the partitioning strategy, we wish to find suitable cut-off positions such that the resulting total alignment is optimal, or at least very close to the optimal alignment of the original sequences. We first consider partitioning each sequence into two parts. A family of the cut-off positions (c_1, \dots, c_N) is called optimal if there is an alignment P of the family of prefixes $S^p = (S_1^p(c_1), \dots, S_N^p(c_N))$ and Q of the family of the suffixes $S^s = (S_1^s(c_1), \dots, S_N^s(c_N))$ such that the concatenation PQ is an optimal alignment of the original sequences $S = (S_1, \dots, S_N)$.

Without a pre-given optimal alignment, the computation of an optimal family of cut-off positions

is not simple. In fact, due to the NP-completeness of MSA, we cannot expect that the computation of optimal cut-off positions requires less time than the computation of an alignment itself. The main difficulty is to evaluate the alignments of the families of the subsequences by their SP scores since it takes large amount of time to calculate the scores of all the possible partitions.

Let X and Y be two sequences with lengths of n and m , respectively. The score of their alignment, namely the length of their longest common subsequences, can be calculated using dynamic programming algorithm in $O(mn)$ time. To reduce the computation time, we designed a score estimating (SE) algorithm to approximately estimate the score of a two-sequence alignment. The algorithm can estimating the score of the alignment of two sequences in $O(m+n)$ time.

The ES algorithm consists of 4 steps each of which scans the two sequences from a different direction. Denote X and Y as the upper and lower sequence, respectively. The four steps are denoted as Left-Upper, Right-Upper, Left-Lower and Right-Lower. The step of Left-Upper starts from the first character in X , say $X[0]$ and searches for the first matching character in Y from left to right. If there is no character in Y matching $X[0]$, it restarts the scan to search for the character matching $X[1]$ in Y . After such character, say $Y[j]$, being found, the algorithm searches for the first character matching with $Y[j]$ in the rest part of X from left to right. If there is no character in X matching $Y[j]$, it restarts the scan to search for the character matching $Y[j+1]$ in Y . After such character, say $X[i]$, being found, the algorithm searches for the first character matching with $X[i]$ in the rest of Y from left to right. We alternately repeat such scans until reaching the end of the sequences. As a result, the number of the matching characters can be obtained in the scan. The procedure of Left-Upper is described as follows, where *count* is the number of the matching characters in X and Y .

Algorithm1 Left-Upper($X, Y, n, m, count$)

```

begin
   $i=0; j=-1; count=0;$ 
  While ( $i < n$ ) and ( $j < m$ ) do
     $k=j+1; found=false;$ 
    Repeat
      While  $X[i] \neq Y[k]$  do  $k=k+1;$ 
      If  $k=m$  then
         $j=j+1; k=j+1$ 
      else
         $count=count + 1; j=k; found=true$ 
      end if
    Until  $found;$ 
     $k=i+1; found=false;$ 
    Repeat
      While  $Y[j] \neq X[k]$  do  $k=k+1;$ 
      If  $k=n$  then
         $i=i+1; k=i+1$ 
      else
         $count=count + 1; i=k; found=true$ 
      end if
  
```

```

    Until found;
  End while
End

```

The procedures of Right-Upper, Left-Lower and Right-Lower are similar to Left-Upper except for the start points and the scan directions. The score estimating (SE) algorithm uses these four procedures and their largest count value as the approximate score of the alignment for X and Y :

Algorithm2 SE(X, Y)

```

Begin
  Left-Upper( $X, Y, n, m, count1$ );
  Left-Lower( $X, Y, n, m, count2$ );
  Right-Upper( $X, Y, n, m, count3$ );
  Right-Lower( $X, Y, n, m, count4$ );
  Return[ max( $count1, count2, count3, count4$ ) ]
End

```

Both the time and space cost of Score-Estimating algorithm are $O(m+n)$.

Let $X = \text{"ACTGCTA"}$, $Y = \text{"TCGATACT"}$, the process of the 4 procedures in Score-Estimating algorithm is depicted in Figure 3. The result of the SE algorithm is 4, which is equal to the exact length of the longest common substring of X and Y .

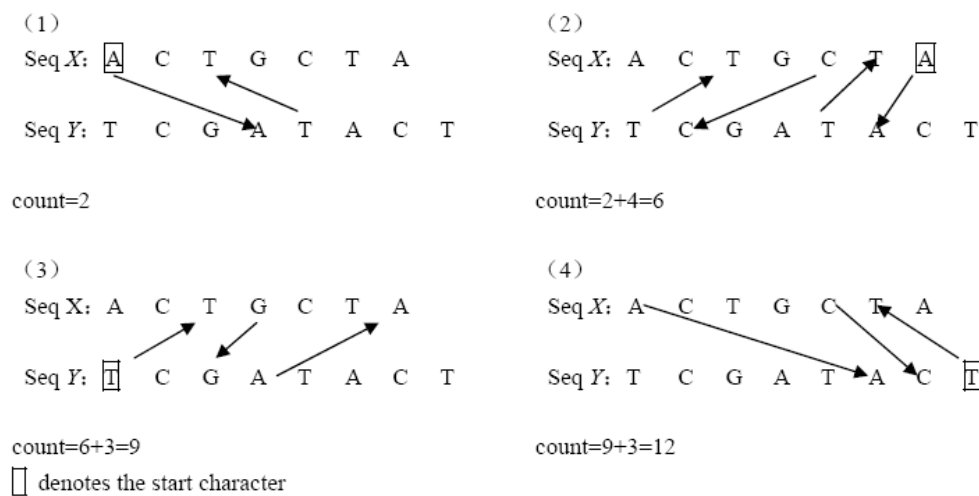


Figure3. An example of Score-Estimating algorithm

3.2 Searching for optimal cut-off points

The first step of our algorithm is to divide the set of sequences into several subsections vertically using an automated partitioning strategy. Each iteration of the partitioning strategy searches for the cut-off points of the sequences using a genetic algorithm. The genetic algorithm starts with a randomly generated population where each individual represents a possible solution of the sequence partitioning, i.e. sets of cut-off points for all sequences. The population is expressed by a 2-dimensional array d where d_{ij} denotes the cut-off position of the j -th sequence in the i -th solution.

In each iteration, the individuals of the $(k+1)$ -th generation in the genetic algorithm are derived from the k -th by the following steps:

1. The fitness of the r -th individual of the k -th generation is evaluated based on the score estimating algorithm.

$$fitness(r) = \sum_{i=1}^{N-1} \sum_{j=1}^{i-1} SE(S_i^p(d_{ri}), S_j^p(d_{rj})) + \sum_{i=1}^{N-1} \sum_{j=1}^{i-1} SE(S_i^p(d_{ri}), S_j^p(d_{rj})) \quad (2)$$

2. The r -th individual of the k -th generation is selected to the next generation according to the following probability:

$$P(r) = \frac{fitness(r)}{\sum_{i=1}^K fitness(i)} \quad , \quad (3)$$

where K is the population size of the number of the k -th generation.

3. Perform crossover operations on the K individuals selected.

Under a certain probability of crossover, the i -th and j -th individuals are selected to be crossed over. In the operation of crossover, two integers l and r , $0 \leq l \leq r \leq N-1$, are randomly selected to indicate the crossover position of the two individuals. In the crossover operation, the values of the elements d_{ik} and d_{jk} are exchanged for all k satisfying $l \leq k \leq r$, and consequently the i -th and j -th individuals are modified.

4. Perform mutation operations on the K individuals selected.

Under a certain probability of mutation, the j -th individual is selected to be mutated. In the operation of mutation, an integer c is randomly generated such that $0 \leq c \leq N-1$, and the value of the element d_{jc} is replaced by an integer ran , which is randomly selected and satisfies $0 \leq ran \leq |S_j|-1$.

After applying the genetic operations including selection, crossover and mutation, a population of a new generation is formed. In the next iteration, the algorithm performs these genetic operations on the new population. This iterative process is repeated for $CYCLE$ times, where $CYCLE$ is a constant number. Finally, the individual with the highest fitness is the result. The algorithm for obtaining a bisecting plan is described as follows.

Algorithm3 Bisect(S, S^p, S^s)

Input: N sequences: $S=(S_0, S_1, \dots, S_{N-1})$

Output: Two families of subsequences S^p and S^s derived from S under a near-optimal partitioning

Begin

- 1 Randomly generate K cut-off plans as the initial population;
- 2 For $generation=1$ to $CYCLE$ do
 - 2.1 Calculate the fitness of the K individuals according to (2);
 - 2.2 Select K individuals as candidates for the next generation according to the probability function (3);
 - 2.3 Perform crossover and mutation operations on the candidates to generate the individuals of the next generation;

End-for *generation*

- 3 Calculate the fitness of the K individuals according to (2). Select the solution with the highest fitness to partition S into two families of subsequences S^p and S^s ;

End

Denote the length of the longest sequence in S as LEN . Both step 2.1 and 3 of the algorithm can be finished in $O(N^2 \cdot LEN)$ time because K is a constant. Since steps 2.2 and 2.3 can be finished in $O(N)$ time, the time cost of each iteration is $O(N^2 \cdot LEN)$. Therefore, the algorithm can be finished in $O(CYCLE \cdot N^2 \cdot LEN)$ time.

To partition the problem of aligning N sequences of length at most L to the problem of aligning about N/L families of short sequences of maximal length l , we can call Bisect() recursively to derive the partitioning strategy, which is described below.

Algorithm4 Partition(S, l, R, num)

Input: $S=(S_0, S_1, \dots, S_{N-1})$: sequences to be partitioned;

l : length limit of the subsequences in the result sets

Output: R : Sets of subsequences derived from S by the optimal cut-off,

Num : number of sets in R

Begin

if the maximal length of S_0, S_1, \dots, S_{N-1} is larger than l then

Bisect(S, S^p, S^s);

partition($S^p, l, R^p, num1$); partition($S^s, l, R^s, num2$);

$R = R^p \cup R^s, num = num1 + num2$;

end if

end

Since the cut-off points in the optimal solutions are usually near the midpoint of the sequences, the algorithm Partition() can be completed in about $\log LEN$ recurrences. In the i -th recurrence, the number of sets of subsequences is about 2^i , the maximal length of the longest subsequence in the sets is near $LEN/2^i$. Since each execution of Bisect() takes $O(CYCLE \cdot N^2 \cdot LEN/2^i)$ time, the i -th recurrence requires $O(CYCLE \cdot N^2 \cdot LEN)$ time. Therefore the time complexity of the algorithm Partition() is $O(CYCLE \cdot N^2 \cdot LEN \cdot \log LEN)$.

After the set S of N sequences of a maximum length L being partitioned into sets of short sequences of a maximum length l , we can align these sets of subsequences separately. While in principle any MSA algorithm can be used to solve the subproblems, we develop an ant colony algorithm as the basic solver for subproblems. The algorithm of multiple sequences alignment based on our partitioning strategy and ant colony optimization is described below.

Algorithm5 Divide_Ant_MSA(S, l, S')

Input: $S=(S_0, S_1, \dots, S_{N-1})$: set of sequences to be aligned;

l : length limit of the subsequences after portioning;

Output: S' : aligned sequences

Begin

1.partition(S, l, R, num); /* Suppose $R = \{R_i | i=1, 2, \dots, num\}$ */

2. for $i=1, 2, \dots, num$

3. Ant-Align(R_i, R'_i)
 4. end-for
 5. assemble R_i ($i=1,2,\dots,num$) to form S'
- end.

In line 3, Ant-Align(R_i, R'_i) is the procedure of aligning the subsequences using ant colony optimization, which is presented in the next section.

4. Ant colony optimization for MSA subproblems

The Ant Colony Algorithm (ACA) is a discrete optimization algorithm proposed by M. Dorigo, et al. [16]. Ant algorithm has been applied to sequential ordering [6] and pair-wise sequence alignment problems [19], and shown high effectiveness and efficiency in solving complex combinational optimizations.

One problem of ACA is the occurrence of super individuals, which can limit ACA's ability of global optimization and lead to premature and local optimal solution. Another problem of ACA is the problem of close race. "Close race" refers to the phenomenon that the individuals' fitness or construction being similar, which means the solutions such individuals represent are very close at the solution space [31]. The two problems above are the results of the ACA's lack of diversity in the strategies of choosing routes, which causes contradictory between the convergence speed of the algorithm and the quality of the solutions. To guarantee successful search of optimal solutions, it is necessary to maintain the diversity of solutions. In our ant colony algorithm for MSA, we develop two methods which adaptively adjust the parameters in the probability function and update the pheromones in order to maintain the diversity of the solutions and avoid local traps.

4.1 Ant colony algorithm for MSA subproblems

The ant system consists of N ants each of which represents a solution of alignment. Each ant search for a alignment by moving on the sequences to choose the matching characters. Let the N sequences be S_0, \dots, S_{N-1} , an artificial ant starts from $S_0[0]$, the first character of S_0 , and selects one character from each of the sequences of S_1, \dots, S_{N-1} matching with $S_0[0]$. From the sequence S_i , $i=1,2,\dots,n-1$, the ant selects a character $S_i[j]$ by a probability determined by the matching score with $S_0[0]$, deviation of its location from $S_0[0]$ and pheromones trail on the logical edge between $S_i[j]$ and $S_0[0]$. In addition, an ant may choose to insert an empty space according to a predetermined probability. Next, the ant starts from $S_0[1]$, selects the characters of S_1, \dots, S_{N-1} matching with $S_0[1]$ to form the second path. Similarly, starting from $S_0[2], \dots, S_0[|S_0|-1]$, the ant can form other paths. All these $|S_0|$ paths form an alignment solution as shown in Figure 4.

When an ant selects a character, the path selected should not cross its existing paths since crossover of paths denotes an impossible alignment. It is also possible for the ant to select an empty character from a sequence, which means a gap is inserted into the sequence.

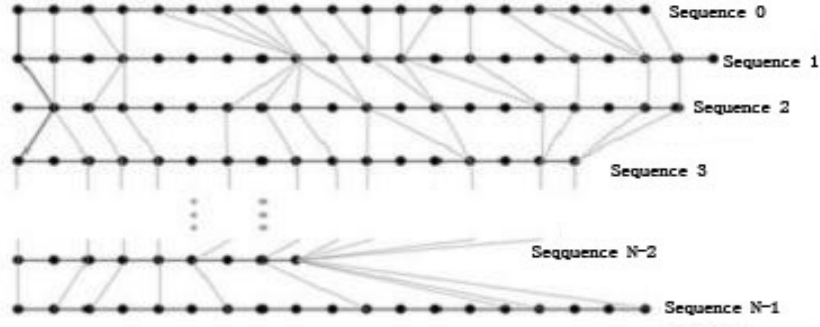


Figure4. An alignment represented by the paths by an ant

The other ants select their path in the same manner, but start from different sequences. The i -th ant starts from S_i and successively goes through S_{i+1} , S_{i+2} , ..., S_n . When it reaches S_n , it continues going through S_0, S_1, \dots, S_{i-1} , to complete the path.

To evaluate an alignment represented by a set of paths, we calculate the positions of characters that are not selected by ants by aligning them to the left and adding gaps to the right and then calculate its SP score. At the end of each iteration, the pheromone on the logical edges is updated according to the alignment score.

4.2 Implementation details

We describe the implementation details of some key components of the ant colony algorithm.

(1) Probability function for selecting the character

Let $P(k, l, n, m)$ be the probability for the ant starting from $S_k[l]$ to select the character $S_n[m]$

in S_n . We define $P(k, l, n, m)$ as follows:

$$P(k, l, n, m) = \frac{phe(k, l, n, m) \times a + mat(k, l, n, m) \times b + dev(k, l, n, m) \times c}{\sum_{r=loc(k, l, n)}^{loc(k, l, n)+h} phe(k, l, n, r) \times a + mat(k, l, n, r) \times b + dev(k, l, n, r) \times c} \quad (4)$$

Here,

$phe(k, l, n, m)$: the pheromone on the logical edge between $S_k[l]$ and $S_n[m]$.

$mat(k, l, n, m)$: the SP score of the characters the ant starting from $S_k[l]$ selects before reaching $S_n[m]$. The SP score is calculated using formula (2).

$loc(k, l, n)$: the start location in S_n when the ant searches the character matching with $S_k[l]$ within S_n . Assuming the ant selects $S_n[q]$ in S_n to match with $S_k[l-1]$, $loc(k, l, n)$ is equal to q .

$dev(k, l, n, m)$: the location deviation between m and q .

a, b, c : the weights of the pheromone, matching score, and location deviation.

h : the range in S_n for the ant to select the character matching with $S_k[l]$.

By this probability function, the characters in S_n which has higher pheromone on the logical edge connecting with $S_k[l]$, higher matching score with $S_k[l]$ and less deviation from $loc(k, l, n)$ has higher probability to be selected.

(2) Selecting a space according to a predetermined probability

It is also possible for the ant to select an empty character from the sequence S_n , which means that a gap is selected in the sequence of the alignment. In our algorithm, all ants select a space according to a fixed probability.

(3) Selecting a character in S_n matching with $S_k[l]$.

When the ant selects a character in S_n to match with $S_k(l)$, it first calculates the selecting probabilities for all characters in predetermined range of S_n . If $S_n(m)$ has the largest probability and $S_n(m)$ is equal to $S_k(l)$, then the ant selects $S_n(m)$. Otherwise, the ant select the characters (include the gap) according to their selecting probabilities by the ‘‘roulette wheel’’ method.

(4) Fitness function

We define the fitness of an alignment A as its SP score which can be calculated by formula (5).

$$SP - Score(A) = \sum_{j=0}^{L-1} \sum_{i=0}^{N-2} \sum_{k=i+1}^{N-1} match(S_{ij}, S_{kj}) \quad , \quad (5)$$

where L is the length of alignment A , S_{ij} is the j -th character of the i -th aligned sequence, and $match(S_{ij}, S_{kj})$ is the matching score between S_{ij} and S_{kj} . Let Σ denote the alphabet, $match(S_{ij}, S_{kj})$ is defined as:

$$match(S_{ij}, S_{kj}) = \begin{cases} +2 & \text{if} \\ -1 & \\ -2 & \\ 0 & \end{cases} \quad (6)$$

$$p(S_{ij}, S_{kj}) = \begin{cases} +2 & \text{if } (S_{ij} = S_{kj} \text{ and } S_{ij}, S_{kj} \in \Sigma) \\ -1 & \text{if } (S_{ij} \neq S_{kj} \text{ and } S_{ij}, S_{kj} \in \Sigma) \\ -2 & \text{if } ((S_{ij} \in \Sigma, S_{kj} \in \text{'-'}) \text{ or } (S_{kj} \in \Sigma, S_{ij} \in \text{'-'})) \\ 0 & \text{if } (S_{ij} = \text{'-'}, S_{kj} = \text{'-'}) \end{cases}$$

(5) Global updating of pheromone

After each ant completes a solution, the pheromone on its logical edges should be updated according the score of the solution. Let $phe(k, l, n, m)$ be the pheromone on the logical edge between $S_k[l]$ and $S_n[m]$, and $evap1$ be the evaporation coefficient, $0 \leq evap1 \leq 1$, the value of $phe(k, l, n, m)$ is updated as:

$$phe(k, l, n, m) = phe(k, l, n, m) \times (1 - evap1) + (alignsum - average) \times evap1, \quad (7)$$

where $alignsum$ is the SP score of the alignment, $average$ is the average score of all the previous alignments.

(6) Adjusting the pheromone

After a number of iterations, a few logical edges might accumulate more pheromone than others, and the ant algorithm could probably converge into a local optimal solution. To avoid such local traps, we adjust the pheromone on the logical edges dynamically.

After a certain number of iterations, if all the scores of the alignments in an iteration are less than the average score of the alignments in the last d iterations, the local convergence will

possibly occur. Here d is a parameter which is adjustable according to the number of the iterations:

$$d = k_1 + \left\lceil \frac{k_2}{\text{generation}} \right\rceil, \quad (8)$$

where k_1, k_2 are constants, and generation is the number of current iteration. It is obvious that in the early iterations d is assigned a greater value, since the pheromone is evenly distributed. In the later iterations, the d value becomes smaller since the pheromone might accumulate on a few logical edges. The rule of pheromone adjusting is: if the pheromone is more than a threshold, it should be evaporated according to a coefficient evap2 ($0.5 \leq \text{evap2} \leq 1$):

$$\text{phe}(k, l, n, m) = \text{evap2} \times \text{phe}(k, l, n, m) \quad (9)$$

By reducing the pheromone on these logical edges, they have less probability to be chosen by the ants in the later iterations.

(7) Adaptive parameters a, b and c

The ants tend to select a path with more pheromone than other paths even if the difference is very slight. As a consequent, the paths the ants select would be concentrated on several paths where the edges have higher pheromone intention. This could result in super individuals and the solution is a local optimal rather than the global optimal.

To avoid the super individuals, we adjust the parameters a, b , and c in probability function (4) adaptively, where a, b , and c are the weights of pheromone, matching score, and location deviation, respectively. In the early stages, we set the values of b and c be larger than a so that the heuristic information of matching score and location deviation is more important. This helps the ants to search in a wide range to avoid the convergence to a local optimum in the early stages.

In later iterations, we should make full use of the feedback information represented by the pheromone. Therefore, we increase a , and decrease b and c adaptively. Combined with the techniques of global and local pheromone updating, the algorithm can successfully speedup the convergence while maintaining the diversity of the solutions to avoid being trapped into local optimal solutions. The initial values of a, b , and c are s_a, s_b , and s_c , respectively. At the end of each iteration, the algorithm adjusts a, b , and c as follows:

$$a = a \times (1 + v_a) \quad \text{if } (a > t_a) \quad a = s_a \quad (10)$$

$$b = b \times (1 - v_b) \quad \text{if } (b < t_b) \quad b = s_b \quad (11)$$

$$c = c \times (1 - v_c) \quad \text{if } (c < t_c) \quad c = s_c \quad (12)$$

where v_a, v_b, v_c are rates of update in the range of (0,1), and t_a, t_b, t_c are upper/lower bounds for a, b , and c , respectively.

Putting all the pieces together, the ant colony algorithm for MSA is outlined below.

Algorithm6 Ant-Align(S, S')

Input: $S = (S_0, S_1, \dots, S_{N-1})$: set of N sequences;

Output: $S' = (S'_0, S'_1, \dots, S'_{N-1})$: the optimal alignment of S ;

alignsum : score of the alignment:

```

Begin
3 Initialization
4 for cycle=1 to Cyclenum do // Cyclenum is the number of iterations
3     for k=1 to N do // loop for the N ants, the k-th ant starts from Sk
4         for l=1 to |Sk| do // loop for the characters Sk[l] in Sk,
5             for n=1 to N do // loop for the other sequences Sn
6                 if (n<>k) then // Select a gap or a character in Sn
7                     for m=loc(k,l,n) to loc(k,l,n)+h-1 do
8                         calculate P(k,l,n,m) using formula (4)
9                     end for m
10                    m= arg maxg P(k,l,n,g)
11                    if (Sn(m)=Sk(l)) then select Sn(m); Sn(loc(k,l+1,n))=m;
                        else probabilistically select a character or a gap using a
                            “roulette wheel” method;
                            Sn(loc(k,l+1,n))= location of the character or gap selected;
12                    end if
13                end if
14            end for n
15        end for l
16    Form an alignment
17    Compute the fitness of the alignment using formula (5)
18    Update the pheromone using formula (9)
19    end for k
20    Select the best alignment from the N solutions.
21    if cycle>start_period then adjust the pheromone
22    Adjust the parameters a ,b, c using formulae (10), (11), (12)
23    end for cycle
End

```

Obviously, the first line and lines from 10 to 13 can be finished in $O(1)$ time. Let L be the maximum length of the sequences in S , and LEN be the length of alignment, lines from 7 to 9 can be calculated in $O(L)$ time because h is a constant. As a result, loop of n can be finished in $O(NL)$ time. Since $|S_k|$ is less than L , the loop of l can be finished in $O(NL)$ time. Because the time cost of lines from 17 to 19 is $O(LEN \cdot N)$ and L is less than LEN , loop of k can be finished in $O(LEN \cdot N^2)$ time. Since the time cost of lines from 21 to 23 is $O(LEN \cdot N)$, the loop of $cycle$ can be finished in $O(Cyclenum \cdot N^2 \cdot LEN)$ time. Therefore, time cost of the algorithm Ant-Align is $O(Cyclenum \cdot N^2 \cdot LEN)$.

5. Experimental results

We test the Divide-Ant-MSA algorithm using some sequences randomly selected from the

benchmark database BALiBASE 2.0. We also test the algorithm of Ant-Align as an independent sequence alignment algorithm, and the SAGA algorithm [9], a leading MSA program using an evolutionary algorithm, to compare their performance with Divide-Ant-MSA.

Comparing to SAGA, the proposed partitioning method is one to two orders of magnitude faster. The running time of Divide-Ant-MSA is only about 1% to 5% of that of SAGA. For example, for 4 sequences with length of 200, the running time of SAGA is 200 to 400 seconds, while the running time of Divide-Ant-MSA is 10 to 20 seconds.

Experimental results also show that the proposed Divide-Ant-MSA algorithm improves the alignment accuracy for long sequences and requires less computational time than Ant-MSA without partitioning. A comparison of the running time of Divide-Ant-MSA and Ant-MSA on sequences with 1000 characters and with an increasing number of sequences is shown in Figure 6. We can see that the performance gain of using the partitioning strategy increases as the number of sequences increases.

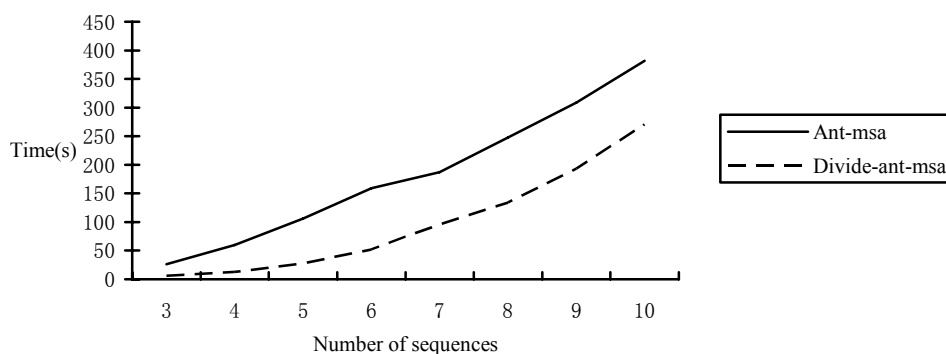


Figure6. Comparison of Ant-MSA and Divide-ant-MSA on sequences with 1000 characters.

Table 1 shows the experimental results of Divide-Ant-MSA on the benchmark problem sets randomly selected from the BALiBASE 2.0 database. The last column in the table is the score of the reference alignment (the larger the better) offered by BALiBASE which is very close to the theoretically optimal alignment. Comparing the scores obtained by Divide-Ant-MSA with the reference alignment scores, we can see that the solution quality of Divide-Ant-MSA is very close to that of the near-optimal reference alignment, and is even better in six cases.

In bioinformatics, if the proportion of the similarity in a set of sequences (shown in the 2nd column of Table 1) is below 30%, most of the sequence alignment methods cannot find the correct alignment. Therefore, it is called in the “twilight zone”. The experiment results in Table1 show that Divide-Ant-MSA can efficiently align not only similar sequences, but also those in the “twilight zone” (with “Similarity” less than 30 in Table 1). From the figures and the table we can also see that Divide-Ant-MSA is efficient for both short and long sequences, and it can find high-quality solutions in reasonable time.

Name of sequences	Similarity	Number of sequences	Length of the longest sequence	Running time(s)	Alignment score of Divide-ant-msa	Score of reference alignment
1fmb	49	4	106	4	811	828
1r69	12	4	78	3	-272	-289
1rvx.A	16	4	69	2	-250	-268
1ubi	16	4	94	3	-290	-342
2trx	15	4	102	3	-303	-293
1idy	13	5	67	4	-278	-285
2fxb	51	5	63	4	752	784
1krn	45	5	82	5	742	814
1uky	17	4	218	12	-745	-731
3grs	17	4	237	13	-786	-768
1hav.A	15	5	199	19	-1520	-1508
1bbr3	14	5	192	19	-1414	-1395
1sbp	19	5	263	25	-1209	-1193
1adj	35	4	418	36	299	306
1ajs.A	14	4	387	33	-1469	-1458
1lvi	19	4	449	40	-1212	-1214
1pam.A	18	5	572	65	-3498	-3443
gal4	14	5	395	46	-2245	-2281
1abo.A	30	15	80	679	384	790
1idy	19	27	60	3513	-11059	-4701

Table 1. Experimental results of Divide-Ant-MSA

6. Conclusions

A novel MSA algorithm based on partitioned discrete optimization is proposed. We have designed an automated partitioning algorithm that can suboptimally divide a MSA set into multiple subproblems using an efficient approximation algorithm to estimate the solution quality after partitioning. To solve each subproblem efficiently and optimally, we have carefully designed an efficient and robust ant colony algorithm for MSA. Adaptive rules to help the ant colony algorithm escape from local traps and converge to high-quality global optimal solutions are proposed. Experimental results have demonstrated superior efficiency and solution quality of the proposed method on some large-scale benchmark sets.

Reference

1. Needleman, S. and Wunsch, C. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, 48:443--453, 1970.
2. Lipman DJ, Altschul SF, Kececioglu JD: A tool for multiple sequence alignment. *Proc.Natl. Acad. Sci. USA* 86, 4412-4415 (1989).
3. Stoye J, Moulton V, Dress AW: DCA: an efficient implementation of the divide-andconquer approach to simultaneous multiple sequence alignment. *Comput. Appl. Biosci.*13(6), 625-6 (1997).
4. Reinert K, Stoye J, Will T: An iterative method for faster sum-of-pair multiple sequence alignment. *Bioinformatics* 16(9),808-814 (2000).
5. Carrillo H, Lipman DJ: The multiple sequence alignment problem in biology.*SIAM J. Appl. Math.* 48, 1073-1082 (1988).

6. Thompson, JD, Higgins, DG and Gibson, TJ (1994) CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice. *Nucleic Acids Research*,1994, vol.22,No.22,4673-4680.
7. Feng D-F, Doolittle RF: Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J. Mol. Evol.* 25,351-360 (1987)
8. Gotoh O: Significant Improvement in Accuracy of Multiple Protein Sequence Alignments by Iterative Refinements as Assessed by Reference to Structural Alignments. *J. Mol. Biol.* 264(4), 823-838 (1996).
9. Notredame C, Higgins DG: SAGA:sequence alignment by genetic algorithm. *Nucleic Acids Res.* 24, 1515-1524 (1996).
10. B. Morgenstern and T. Werner (1997) DIALIGN: Finding local similarities by multiple sequence alignment. *Bioinformatics* 14, 290-294.
11. Morgenstern B DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, 1 March 1999, vol.15, no. 3, pp. 211-218(8).
12. A.Krogh. An introduction to hidden markov models for biological sequences. In: S. L. Salzberg, D. B. Searls and S. Kasif. *Computational Methods in Molecular Biology*. Elsevier, 1998: 45-63
13. Christopher Lee, Catherine Grasso and Mark F. Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, Vol 18 no. 3 2002, pp452-464
14. Kazutaka Katoh , Kazuharu Misawa1 , Kei-ichi Kuma and Takashi Miyata MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Res.* 2002 Jul 15;30(14):3059-66
15. A heuristic algorithm for multiple sequence alignment based on blocks. P.Zhao and Tao Jiang *J. Combinatorial Optimization* 5(1):95–115, Mar 2001
16. Dorigo M., Maniezzo V., Colomi A. Ant system: Optimization by a colony of cooperating agents [J]. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 1996, 26(1): 29-41.
17. Notredame C, Higgins DG, Heringa J T-Coffee: A novel method for fast and accurate multiple sequence alignment. *J Mol Biol.* 2000 Sep 8;302(1):205-217
18. Edgar RC. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics.* 2004 Aug 19;5(1):113.
19. LIANG Dong, HUO Hong—wei. An Adaptive Ant Colony Optimization Algorithm and Its Application to Sequence Alignment. *Computer Simulation*, 2005, 22(1):100-106.
20. Do, CB, Brudno, M., and Batzoglou, S. 2004. ProbCons: Probabilistic consistency-based multiple alignment of amino acid sequences. *Proc. the Thirteenth National Conference on Artificial. Intelligence*, pp. 703–708.
21. C. Notredame, Recent progresses in multiple sequence. alignment: a survey, *Pharmacogenomics* 3(1) (2002). 131–144
22. Kim J, Pramanik S, Chung MJ: Multiple Sequence Alignment using Simulated Annealing. *Comp. Applic. Biosci.* 10(4), 419-426 (1994).
23. Arthur L Delcher,Simon Dasif. Alignment of whole genomes[J]. *Nucleic Acids Research*,1999(27):2369-2376
24. Lawrence CE, Altschul SF, Boguski MS, Liu JS, Neuwald AF, Wootton JC: Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science* 262,208-214 (1993).
25. Barton GJ, Sternberg MJE: A strategy for the rapid multiple alignment of protein sequences: confidence levels from tertiary structure comparisons. *J. Mol. Biol.* 198,327-337 (1987).
26. Berger MP, Munson PJ: A novel randomized iterative strategy for aligning multiple protein sequences. *Comput. Appl. Biosci.* 7,479-484 (1991).