ELSEVIER

Theoretical Computer Science 210 (1999) 341–374

Theoretical Computer Science

Enhanced A* algorithms for multiple alignments: optimal alignments for several sequences and k-opt approximate alignments for large cases [1]

Takahiro Ikeda [2], Hiroshi Imai *

Department of Information Science, University of Tokyo, Tokyo 113, Japan

Abstract

The multiple alignment of the sequences of DNA and proteins is applicable to various important fields in molecular biology. Although the approach based on Dynamic Programming is well-known for this problem, it requires enormous time and space to obtain the optimal alignment. On the other hand, this problem corresponds to the shortest path problem and the A* algorithm, which can efficiently find the shortest path with an estimator, is usable.

First, this paper directly applies the A* algorithm to multiple sequence alignment problem with more powerful estimator in more than two-dimensional case and discusses the extensions of this approach utilizing an upper bound of the shortest path length and of modification of network structure. The algorithm to provide the upper bound is also proposed in this paper. The basic part of these results was originally shown in Ikeda and Imai [11]. This part is similar to the branch-and-bound techniques implemented in MSA program in Gupta et al. [6]. Our framework is based on the edge length transformation to reduce the problem to the shortest path problem, which is more suitable to generalizations to enumerating suboptimal alignments and parametric analysis as done in Shibuya and Imai [15–17]. By this enhanced A* algorithm, optimal multiple alignments of several long sequences can be computed in practice, which is shown by computational results.

Second, this paper proposes a k-group alignment algorithm for multiple alignment as a practical method for much larger-size problem of, say multiple alignments of 50–100 sequences. A basic part of these results were originally presented in Imai and Ikeda [13]. In existing iterative improvement methods for multiple alignment, the so-called group-to-group two-dimensional dynamic programming has been used, and in this respect our proposal is to extend the ordinary two-group dynamic programming to a k-group alignment programming. This extension is conceptually straightforward, and here our contribution is to demonstrate that the k-group alignment can be implemented so as to run in a reasonable time and space under standard computing environments. This is established by generalizing the above A* search approach. The k-group alignment method can be directly incorporated in existing methods such as iterative improvement algorithms [2, 5] and tree-based (iterative) algorithms [9]. This paper performs computational experiments by applying the k-group method to iterative improvement algorithms, and shows that our approach can find better alignments in reasonable time. For example, through larger-scale

[1] This paper is a refined version combining results of two papers presented at Genome Informatics Workshop'94 and '95 [11, 13] and also those of the Master's thesis of the first author [10].

[2] Currently at C&C Media Research Laboratories, NEC Corporation.

* Corresponding author. E-mail: imai@is.s.u-tokyo.ac.jp.

0304-3975/99/$ – see front matter
PII S0304-3975(98)00093-0

computational experiments here, 34 protein sequences with very high homology can be optimally 10-group aligned, and 64 sequences with high homology can be optimally 5-group aligned.

Keywords: Multiple Alignment; A* Algorithm; A Algorithm; Group Alignment

1. Introduction

The multiple sequence alignment is the problem to find the alignment of multiple sequences with highest score due to a given scoring criterion between characters. The solution of this problem for multiple sequences of DNA and proteins represent the similarity of them and are applicable to various important fields such as the prediction of three-dimensional structures of proteins and the inference of phylogenetic tree in molecular biology; see, for example, [19].

This paper proposes enhanced A* algorithms for multiple alignments. The results consists of the following two parts:

1. the enhanced A* algorithm which can compute an optimal multiple alignment of several sequences (a basic idea was presented in Ikeda and Imai [11], and here we present more extended use of A* paradigm based on Ikeda [10])
2. k-group iterative improvement algorithm which can compute an good approximate multiple alignments of many sequences (a basic idea was given in Imai and Ikeda [13], and here new larger-size experimental results are shown)

In the following, we describe results for each part in this order.

1.1. Optimal multiple alignment of several sequences by A*

This problem can be solved by finding the shortest path on some directed acyclic graph. Suppose that S_k denotes the kth sequence whose length is $n_k = O(n)$ and d denotes the dimension, the number of sequences. Then in the directed acyclic graph $G = (V,E)$ such that $V = \{(x_1,\ldots,x_d) \mid x_i = 0,1,\ldots,n_i\}$ and $E = \bigcup_{e\in\{0,1\}^d}\{(v,v+e) \mid v,v+e \in V,\ e \neq 0\}$, a path from the vertex $s = (0,\ldots,0)$ to the vertex $t = (n_1,\ldots,n_d)$ corresponds to an alignment of sequences.

For instance, in two-dimensional case, the graph G is constructed as Fig. 1. In this graph, each row and column correspond to each character of first and second sequences, respectively. A diagonal edge represents a match between two characters and both horizontal and vertical edges represent insertions of gaps. Therefore, finding the optimal alignment is equivalent to finding the shortest path from the top left vertex to the bottom right vertex on an appropriate assignment of edge length such that matching cost of two characters, which denotes less similarity of them, is assigned to each diagonal edge and the gap cost is assigned to each horizontal and vertical edge. If matching scores representing the similarity of characters are given, matching costs are obtained by reversing their signs.

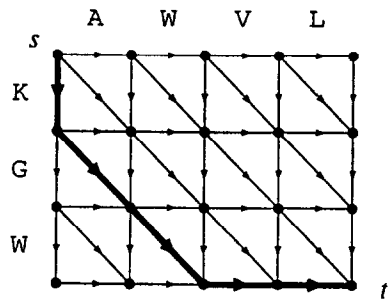


Fig. 1. The graph for the alignment of two sequences, KGW and AWVL. The path from s to t drawn as the bold line represents the alignment of KGW-- and -AWVL.

In more than two-dimensional case, the sum of all scores for pairwise sequence alignments is used as the score for the multiple sequence alignment in general. This corresponds to defining each edge length in the original graph $G = (V,E)$ as the sum of all corresponding edge length in the graphs for pairwise alignments. Let $G_{ij} = (V_{ij},E_{ij})$ denote the graph for the alignment of S_i and S_j $(i < j)$, that is, $V_{ij} = \{v_{ij} = (x_i,x_j) \mid v = (x_1,\ldots,x_d) \in V\}$ and $E_{ij} = \{(u_{ij},v_{ij}) \mid (u,v) \in E, u_{ij} \neq v_{ij}\}$. Then the length of edge (u,v) in E is defined as $l(u,v) = \sum_{1 \leqslant i < j \leqslant d} l(u_{ij},v_{ij})$ where $l(u_{ij},v_{ij})$ denotes the length of edge (u_{ij},v_{ij}) in graph G_{ij} and has been defined.

The method based on Dynamic Programming (DP) is a faithful approach for multiple sequence alignment problem. This method searches all vertices in the graph and has $O(n^d)$ time and space complexity. Although this approach is effective for small dimension of two or three, it is impractical to apply this method directly to higher dimensional problem because n^d is enormous even for a little larger d. In order to avoid this tendency, approximate methods are used for higher dimensional alignment problem since it is difficult to bound the search space without lack of optimality of the result alignment.

On the other hand, the A* algorithm, which is the well-known heuristic search method in Artificial Intelligence, always finds the optimal alignment with less vertices searched [7]. It reduces unnecessary search by utilizing the heuristic estimate for the shortest path length from each vertex to the destination.

The concept of this algorithm has been used in multiple sequence alignment problem to bound the search space of DP [18] (see also [3]). Although the A* algorithm is incompatible with DP, this bounding method overcomes this problem by using an upper bound of the shortest path length. Therefore, this method requires additional time to obtain an upper bound compared with the original A* algorithm and besides its search space is not less than that of the original A* algorithm.

On the other hand, the A* algorithm can be directly used in finding the shortest path on the graph for multiple sequence alignment. In two-dimensional case, the A* algorithm has been directly applied and its effectiveness has been reported [1] (see also [14]).

This paper focuses on more than two-dimensional case, and shows the A* algorithm can find the shortest path by only searching sufficiently small space in the actual application to protein sequence alignment. This paper further improves this approach in order to decrease the necessary space using an upper bound for the shortest path length. This algorithm is never inferior to the approach based on DP using an upper bound with regard to the necessary space. This paper also proposes an efficient approximate algorithm which provides the almost optimal alignment with less necessary time and space. This algorithm is applicable to finding an upper bound as well. Modification of network structure to solve larger problems is also touched upon.

It should be noted that the original idea of these results were presented in [11]. The branch-and-bound techniques implemented in MSA program [6] is very similar to this enhanced A* algorithm. Our framework is based on the edge length transformation to reduce the problem to the shortest path problem, which is more suitable to generalizations to enumerating suboptimal alignments and parametric analysis as done in Shibuya and Imai [15–17].

By this enhanced A* algorithm, optimal multiple alignments of several long sequences ($d = 8$ to 10 for highly homologous sequences and $d = 4, 5$ for sequences with low homology) can be computed in practice, which is shown by computational results.

1.2. k-opt approximate multiple alignments for many sequences

For large d, say much larger than 10, even the above enhanced A* algorithm cannot solve the problem in an optimal way. This is inevitable from the viewpoint of computational complexity theory, since the problem is NP-hard when regarding d as a parameter.

In existing approaches, for large d, approximate algorithms dividing d sequences into two groups and applying two-dimensional DP between the two groups have been used [2, 5]. The resultant alignment is improved gradually by iteration of dividing and aligning. Another method for multiple alignment is a tree-based (iterative) algorithm [9], and in it two-dimensional DP is also used. In connection with these algorithms, Hirosawa et al. [8] employed three-dimensional DP as the basis for an initial alignment to the subsequent iterative algorithm.

This paper investigates a k-group alignment algorithm for multiple alignment, based on Imai and Ikeda [13] together with new larger computational results. In the k-group alignment problem, d sequences are given with k disjoint groups of them, each being internally aligned, and a best alignment among these k groups should be found with only inserting a gap simultaneously in the same position for the alignment of each group. First, it is noted that the enhanced A* approach above can be applied to the group alignment problem. Several ways of applying A* search to this problem are discussed. Then, its connection with the standard iterative improvement algorithm is described.

Through computational experiments, it is demonstrated that the k-group alignment can be performed for $k = 3, 4, 5$ in a practical time depending on the problem size,

and this produces better alignments. For example, for 9 sequences of length about 750, whose similarity is relatively low, 3-group alignment can be performed very fast even starting with a bad initial alignment, and 4-group alignment can be executed for mildly good alignments. For 21 sequences of length about 430, whose similarity is quite high, 5-group alignment can be performed iteratively. About the alignment quality, 3-group alignment yields better solutions compared with 2-group DP almost with a little additional time. 4- and 5-group alignment methods can find better solutions in most cases but require more time.

Furthermore, we can solve much larger problems by our approach. For example, 34 protein sequences with very high homology can be optimally 10-group aligned, and 64 sequences with high homology can be optimally 5-group aligned.

The practicality of k-group alignment is thus shown, and further investigation of elaborating this with other methods and enhancing itself should be done.

2. Preliminaries on shortest path algorithms

This section presents some basic definitions about networks, and prescribes the shortest path problem and the shorter paths problem based on the definitions. Many of these are more or less standard, but the following would be a good summary of shortest path algorithms applicable in genome informatics. For example, in extending the approach in this paper further, it becomes necessary to obtain all matches included in suboptimal alignments whose score is within some factor of the optimal score which corresponds to the shorter path problem (see [15, 16]). Hence, we here try to include these descriptions for completeness.

A *network* $N = (G = (V,E), l)$ is a directed graph $G = (V,E)$ where $l(u,v)$ provides the length of the edge (u,v). A *path* from u to v in a network N is an ordered set of edges $((v_1,v_2), (v_2,v_3), \ldots, (v_{n-1},v_n))$ such that $u = v_1$ and $v = v_n$. The length of this path is given by

$$\sum_{i=1}^{n-1} l(v_i, v_{i+1}).$$

A vertex is on a path or a path passes a vertex if the path contains an edge incident to the vertex. The *shortest path* from s to t in a network N is the path form s to t which has minimum length in all paths from s to t in N.

We assume that each vertex in the network is reachable from s and has no cycle, which is a path from an vertex to the same vertex, with negative length. Then the shortest path between any two vertices necessarily exists. The *shortest path problem* is defined as the problem to find the shortest path from given source vertex to given destination vertex in a network. This paper denotes the source vertex as s and the destination vertex as t.

This paper describes the length of an path with special notation for convenience. In the following, $L_N^P(u,v)$ denotes the length of a part of a path P from u to v in a

network N, and $L_N^*(u,v)$ denotes the shortest path length from u to v in N.

The *shorter paths* from s to t within Δ in the network $N = (G = (V,E), l)$ is the set of paths from s to t such that

$$L_N^P(s,t) \leqslant L_N^*(s,t) + \Delta$$

for each path P in the set. The shorter paths within Δ represents the set of paths longer than the shortest path by at most Δ. This paper denotes the set of edges contained in the better paths within Δ in a network $N = (G = (V,E), l)$ as E_Δ for N.

2.1. The Dijkstra algorithm

The Dijkstra algorithm is a traditional searching algorithm to find the shortest path in the network without negative length edges. This algorithm fixes the shortest path from the source vertex s for each vertex in order of its length. Although the original algorithm by Dijkstra obtains the shortest path to all vertices in the network, this paper calls the following Algorithm DIJKSTRA modified for the shortest path problem between two vertices, which finishes searching after fixing the shortest path from s for the destination vertex t, as the Dijkstra algorithm.

Algorithm DIJKSTRA

Input: A network $N = (G = (V,E), l)$ where $l(u,v) \geqslant 0$ for any edge (u,v) in E, a source vertex s, and a destination vertex t.

Output: The shortest path from s to t.

begin

1. $S \leftarrow \emptyset$; $\overline{S} \leftarrow \{s\}$; $p(s) \leftarrow 0$; $P(s) \leftarrow \emptyset$;

2. repeat

2.1. Let u be the vertex in $\overline{S}$ which minimizes p in $\overline{S}$;

2.2. $S \leftarrow S \cup \{u\}$; $\overline{S} \leftarrow \overline{S} \backslash \{u\}$;

2.3. if $u = t$ **then return** $P(t)$;

2.4. for each $(u,v) \in E$ **do**

if $v \notin \overline{S}$ **or** $p(u) + l(u,v) < p(v)$ **then begin**

$p(v) \leftarrow p(u) + l(u,v)$;

Let $P(v)$ be the path which consists of the path $P(u)$ and the edge (u,v);

$\overline{S} \leftarrow \overline{S} \cup \{v\}$;

end

until false;

end

In this algorithm, $\overline{S}$ is a set of "tentatively labeled" vertices v such that some path from s to v is already found, and S is a set of "permanently labeled" vertices v such that the shortest path to v is already fixed. This algorithm repeats adding a vertex to S and renewing p and P for all its descendant vertices. This paper calls these sequential

operations as an expansion of the vertex. According to this operation, this algorithm keeps the shortest path from the source to each vertex v passing only vertices in S as $P(v)$ and its length as $p(v)$. This paper calls this value $p(v)$ as the potential of the vertex v. The following proposition is established from these observation on the potential.

Proposition 2.1. *When Algorithm DIJKSTRA adds a vertex v to S in Step 2.2,*

$$L_N^*(s,v) \leqslant L_N^*(s,u),$$

for any vertex u in $V\backslash S$.

According to this proposition, the shortest path from s to a vertex in S does not pass any vertex in $S\backslash V$ since the network has no negative length edge. Thus, the following proposition follows.

Proposition 2.2. *In Algorithm DIJKSTRA,*

$$p(v) = L_N^*(s,v)$$

for any vertex v in S.

This proposition shows the correctness of the algorithm. These propositions are strongly based on the restriction that the network contains no negative length edge. This restriction is essential to the Dijkstra algorithm.

Proposition 2.1 indicates that the Dijkstra algorithm adds the vertex nearest to the source to S in each iteration step. This means that it expands each vertex in order of the shortest path from s to it. For the shortest path problem between two vertices, this searching strategy is not suitable from the point of view that the algorithm searches vertices regardless of the destination t. This algorithm equally searches vertices nearer to t and further to t.

2.2. The A algorithm

The A algorithm searches less number of vertices utilizing preliminary information on the network ([7], etc.). In most cases, one can derive some information on the network to be searched from the structure of the problem. The A algorithm uses heuristic estimate for the shortest path length from each vertex to t as preliminary information. Let $h(v)$ be this estimate value for a vertex v. For the destination vertex t, $h(t)$ is defined as zero. This paper calls such an estimate function h as an estimator. The following Algorithm A represents this algorithm.

Algorithm A

Input: A network $N = (G = (V,E), l)$, a source vertex s, and a destination vertex t.

Output: The path from s to t.

begin
1. $S \leftarrow \emptyset$; $\overline{S} \leftarrow \{s\}$; $p(s) \leftarrow 0$; $P(s) \leftarrow \emptyset$;
2. repeat
2.1. Let u be the vertex in $\overline{S}$ which minimizes $p(v) + h(v)$ in $\overline{S}$;
2.2. $S \leftarrow S \cup \{u\}$; $\overline{S} \leftarrow \overline{S} \backslash \{u\}$;
2.3. if $u = t$ **then return** $P(t)$;
2.4. for each $(u, v) \in E$ **do**
if $v \notin S \cup \overline{S}$ **or** $p(u) + l(u, v) < p(v)$ **then begin**
$p(v) \leftarrow p(u) + l(u, v)$;
Let $P(v)$ be the path which consists of the path $P(u)$ and the edge (u, v);
$\overline{S} \leftarrow \overline{S} \cup \{v\}$;
if $v \in S$ **then** $S \leftarrow S \backslash \{v\}$;
end
until false;
end

This algorithm has the same frame as Algorithm DIJKSTRA and works in similar way. This algorithm expands each vertex v in order of the value $p(v) + h(v)$, which represents the estimated shortest path length from s to t via v. This implies that the algorithm preferentially searches the vertices on the paths from s to t estimated to be shorter, if each estimate appropriately reflects the shortest path length from each vertex to t. In such cases, the search space is narrowly bounded since unnecessary expansion of vertices decreases. However, this algorithm has some shortcomings compared with the Dijkstra algorithm.

First, Proposition 2.2 does not follow for this algorithm. The shortest path from s is not fixed for each vertex in S, in other words, shorter paths may be found in future for such a vertex. This is the reason why the algorithm removes a vertex from S when it renews the potential of the vertex in Step 2.4. This operation causes the several times expansion of one vertex and increase of necessary time for searching.

Second, the algorithm does not necessarily output the shortest path from s to t. The A algorithm is an approximate method in this sense. This serious defect of the A algorithm is only due to the quality of the estimator. The A* algorithm in the following section guarantees the shortestness of the result path length with an appropriate estimator.

2.3. The A algorithm*

The A algorithm necessarily finds the shortest path in the network without negative length edges if the estimator satisfies the following condition for each vertex v:

$$h(v) \leqslant L_N^*(v, t). \tag{1}$$

This condition prescribes that the estimate for the shortest path length from each vertex to t does not exceed the actual shortest path length, that is, each estimate is consistent

with the corresponding estimated value. The A algorithm using the estimator satisfying this condition is called as the A* algorithm in particular.

The constant function $h(v) = 0$ is the trivial estimator satisfying above condition, and the A algorithm using this estimator is the A* algorithm. The A* algorithm using this estimator is equivalent to the Dijkstra algorithm as it is. This is due to the fact that this estimator provides no effective information on the shortest path length to t. Hence, the Dijkstra algorithm is a special case of the A* algorithm.

The estimator plays an important role in the A* algorithm. It conditions not only the optimality of the result path length but also the efficiency of the algorithm. It is known that the A* algorithm using an estimator h necessarily expands the vertices expanded by the A* algorithm using an estimator h' if $h'(v) \leqslant h(v)$ for any vertex v in V. This means that the A* algorithm using a stronger estimator expands less vertices than the A* algorithm using a weaker estimator. From the point of view of applying the A* algorithm to problems, utilizing the property of each problem for the estimator holds the key to the fast solution with less search space.

The A* algorithm still has the shortcoming that the shortest path from s is not fixed for all vertices in S unlike the Dijkstra algorithm although the shortest path from s to t is fixed when the algorithm selects the vertex t in Step 2.1. This shortcoming is also due to the quality of the estimator, and can be removed with an appropriate estimator.

An estimator h for the shortest path length to t in the network $N = (G = (V,E), l)$ is *dual feasible* if and only if it satisfies the following condition for any edge (u,v) in E:

$$l(u,v) + h(v) \geqslant h(u).$$

This condition prescribes that the estimate value for the initial vertex of an edge is more than that for the terminal vertex of an edge by at most the length of the edge, that is, estimate values are consistent with each edge length. This is stronger than the condition of the A* algorithm given by expression (1). In fact, if h is a dual feasible estimator,

$$\begin{aligned} L_N^*(v,t) &= \sum_{(u_i,u_i+1)\in P} l(u_i,u_i+1) \\ &\geqslant \sum_{(u_i,u_i+1)\in P} (h(u_i) - h(u_i+1)) \\ &= h(v) - h(t) \\ &= h(v) \end{aligned}$$

for any vertex v in V where P denotes the shortest path from v to t.

If the estimator is dual feasible, the A* algorithm does not renew the potential of a vertex in S and does not remove a vertex from S in Step 2.4. In fact, the following powerful proposition is established.

Proposition 2.3. *Let h be a dual feasible estimator for the shortest path length from each vertex to t in a network $N = (G = (V,E), l)$. Then the Dijkstra algorithm for*

the network $N_h = (G, l_h)$ acts as the A algorithm for the network N, and outputs the shortest path from s to t in the network N, where l_h is defined for any edge (u, v) in E as follows*:

$$l_h(u, v) = l(u, v) + h(v) - h(u).$$

Proof. Let p_D, P_D, S_D, and $\overline{S}_D$ be p, P, S, and $\overline{S}$ in the Dijkstra algorithm for the network N_h. Let $p_A(v)$ denote $L_N^{P_D(v)}(s, v)$, the length of the path $P_D(v)$ in the network N, for any vertex v in V. This proof shows that the Dijkstra algorithm using p_D, P_D, S_D, and $\overline{S}_D$ is equivalent to the A* algorithm using p_A, P_D, S_D, and $\overline{S}_D$.

Suppose that above claim follows after some times iteration in Step 2 in Algorithm DIJKSTRA. Then in the next iteration,

$$\begin{aligned} p_D(v) &= \sum_{(u_i,u_{i+1})\in P_D(v)} l_h(u_i, u_{i+1}) \\ &= \sum_{(u_i,u_{i+1})\in P_D(v)} (l(u_i, u_{i+1}) + h(u_{i+1}) - h(u_i)) \\ &= \sum_{(u_i,u_{i+1})\in P_D(v)} l(u_i, u_{i+1}) + h(v) - h(s) \\ &= p_A(v) + h(v) - h(s) \end{aligned}$$

for any vertex v in $\overline{S}_D$. Since $h(s)$ is a constant, the vertex minimizing $p_D(v)$ in $\overline{S}_D$ corresponds to the vertex minimizing $p_A(v) + h(v)$ in $\overline{S}_D$. Hence operations for P_D, S_D and $\overline{S}_D$ in Step 2 in the Dijkstra algorithm corresponds to those in Step 2 in the A* algorithm. Although the Dijkstra algorithm does not have the step removing a vertex from S in the A* algorithm, this step has no effect on the Dijkstra algorithm for N_h from Propositions 2.1 and 2.2 which follow from the fact that N_h has no negative length edges because of dual feasibility of the estimator h. Therefore the claim follows after this iteration step.

Since the Dijkstra algorithm correctly works as the A* algorithm in Step 1, the proposition follows from induction. □

This proof shows the set S_D in the Dijkstra algorithm for the network N_h corresponds to the set S in the A* algorithm for the network N. Hence, the A* algorithm using dual feasible estimator clearly satisfies Proposition 2.2 for the Dijkstra algorithm. Moreover, for any path P between two vertices u and v in V,

$$\begin{aligned} L_{N_h}^P(u, v) &= \sum_{(v_i,v_{i+1})\in P} l_h(v_i, v_{i+1}) \\ &= \sum_{(v_i,v_{i+1})\in P} (l(v_i, v_{i+1}) + h(v_{i+1}) - h(v_i)) \\ &= \sum_{(v_i,v_{i+1})\in P} l(v_i, v_{i+1}) + h(v) - h(u) \\ &= L_N^P(u, v) + h(v) - h(u), \end{aligned}$$

and then

$$L^*_{N_h}(u,v) = L^*_N(u,v) + h(v) - h(u), \tag{2}$$

because $h(v)$ and $h(u)$ do not depend on the path P. Therefore, the following proposition follows from Proposition 2.1 for the Dijkstra algorithm.

Proposition 2.4. *When Algorithm A using a dual feasible estimator adds a vertex v to S in Step* 2.2,

$$L^*_N(s,v) + h(v) \leqslant L^*_N(s,u) + h(u),$$

for any vertex u in $V\backslash S$.

In Proposition 2.3, it is remarkable that $l_h(u,v)$ is not negative for any edge (u,v) in E if the estimator is dual feasible. The Dijkstra algorithm is applicable to the network N_h even though the network N has negative length edges in this case. This means that the A* algorithm using dual feasible estimator correctly works for the network with negative length edges.

The constant function $h(v) = 0$ is dual feasible only if the network has no negative length edges. Obviously, no constant function is dual feasible for such a network. This corresponds to the fact that the Dijkstra algorithm is only valid for the network without negative length edges.

3. Preliminaries on multiple sequence alignment problem

3.1. Basic concepts

We here present some basic definitions and describes the relation between this problem and the shortest path problem.

In the multiple sequence alignment problem, a fixed set of characters C including a special character representing a gap and a fixed scoring function $w : C \times C \to R$ which provides the similarity between characters are given. This problem concerns only characters in the set C. This paper assumes the set C consists of capital alphabets which denote normal characters and "–" which denotes a gap character.

A *sequence* is an ordered set of characters $(c_1, c_2, \ldots, c_m)$ where $c_i \in C$ for all $1 \leqslant i \leqslant m$. This paper simply denotes this sequence as $c_1 c_2 \cdots c_m$. The length of a sequence is the number of characters in the sequence.

An *alignment* of d sequences

$$\sigma_1 = c_{11}\ c_{12}\ \cdots\ c_{1m_1},$$

$$\vdots$$

$$\sigma_d = c_{d1}\ c_{d2}\ \cdots\ c_{dm_d}$$

is a set of d sequences with equal length

$$\overline{\sigma}_1 = \overline{c}_{11}\,\overline{c}_{12}\,\cdots\,\overline{c}_{1m},$$
$$\vdots$$
$$\overline{\sigma}_d = \overline{c}_{d1}\,\overline{c}_{d2}\,\cdots\,\overline{c}_{dm}$$

satisfying the following conditions: (i) For each $1 \leqslant i \leqslant d$, there exists an increasing sequence of numbers $n_{i,1}, n_{i,2}, \ldots, n_{i,m_i}$ such that

$$\sigma_i = \overline{c}_{in_{i,1}}\,\overline{c}_{in_{i,2}}\,\cdots\,\overline{c}_{in_{i,m_i}},$$

and $\overline{c}_{ij_i}$ equals to a gap character for all $1 \leqslant j_i \leqslant m$ that is not contained in the sequence of numbers, (ii) For each $1 \leqslant j \leqslant m$, there exists at least one sequence $\overline{\sigma}_i$ such that $\overline{c}_{ij}$ is not a gap character.

This definition prescribes that an alignment of sequences consists of the sequences in which gaps are inserted to make their length equal and that there exists no location where any sequence has a gap character. This paper calls the set of the characters occupying the same location in the sequences as a column. Each column contains at least one normal character, and denotes the characters to be aligned. A sequence has a gap character in a column if normal characters in the sequence does not used in the alignment corresponding to the column.

The *score* of an alignment

$$\overline{\sigma}_1 = \overline{c}_{11}\,\overline{c}_{12}\,\cdots\,\overline{c}_{1m},$$
$$\vdots$$
$$\overline{\sigma}_d = \overline{c}_{d1}\,\overline{c}_{d2}\,\cdots\,\overline{c}_{dm}$$

is given by

$$\sum_{1 \leqslant i < j \leqslant d} \sum_{n=1}^{m} w(c_{in}, c_{jn}).$$

This paper calls the number of sequences in an alignment as the dimension of the alignment. The score of an alignment of two dimensions is defined as the sum of scores for all aligned characters, in other words, the sum of degree of similarities between aligned two characters. Although a special score is sometimes used for sequential gaps in the field of molecular biology, this paper does not concerns such a score for the sake of simplicity. The score of an alignment of more than two dimensions is defined as the sum of scores for all pairwise alignments. This scoring system for an alignment of more than two dimensions is called as sum of pairs (SP) score and is generally utilized for aligning protein and DNA sequences. In this case, the score between two gaps is set to zero.

The *sequence alignment problem* is defined as the problem to find the alignment of given sequences with maximal score. Particularly, the sequence alignment problem for

more than two sequences is called as the *multiple sequence alignment problem*. Since the score of an alignment represents degree of similarity between the sequences in it, the alignment with maximum score is the optimal alignment from the point of view of aligning similar characters as much as possible.

This problem corresponds to the shortest path on some network. Let d be the number of given sequences and $\sigma_i = c_{i1}c_{i2}\cdots c_{im_i}$ denotes the sequence for $i \leqslant i \leqslant d$. Then in the graph $G^A = (V^A, E^A)$ such that

$$V^A = \{(x_1,\ldots,x_d) \mid x_i = 0,1,\ldots,m_i\},$$
$$E^A = \{(u,v) \mid u = (x_1,\ldots,x_d) \in V^A,\ v = (x_1+e_1,\ldots,x_d+e_d) \in V^A,$$
$$u \neq v,\ e_i = 0,1 \text{ for all } 1 \leqslant i \leqslant d\}$$

a path from the vertex $s = (0,\ldots,0)$ to the vertex $t = (m_1,\ldots,m_d)$ corresponds to an alignment of sequences. A vertex $(x_1,\ldots,x_d)$ represents the state where the first x_i characters of each sequence σ_i is aligned with each other in the same part of the result alignment. An edge (u,v) represents an alignment in a column. For an edge (u,v) where $u = (x_1,\ldots,x_d)$ and $v = (x_1+e_1,\ldots,x_d+e_d)$, the vertex u denotes that the first x_i characters forms a certain number of columns in the alignment, and $(e_1,\ldots,e_d)$ stands for an alignment of characters in the next column. A gap character is inserted to the sequence σ_i in this column if e_i equals to zero, and the (x_i+e_i)th character in σ_i is aligned in this column if e_i equals to one.

For instance, the graph G^A is constructed as Fig. 1 for an alignment of two sequences, KGW and AWVL. In this graph, a diagonal edge represents an alignment of two characters and both horizontal and vertical edges represent insertions of gaps. The bold line denotes the path from the top left vertex s to the bottom right vertex v corresponding to alignment of KGW-- and -AWVL. In this case, finding the optimal alignment is equivalent to finding the shortest path from s to t in the network based on this graph where a minus quantity of score between corresponding two characters is assigned to each edge.

For the multiple sequence alignment problem, each edge length is given by the sum of all corresponding edge length in the networks for pairwise alignment problems. Let $N_{ij}^A = (G_{ij}^A = (V_{ij}^A, E_{ij}^A), l_{ij}^A)$ be the network for the pairwise alignment of σ_i and σ_j where $i < j$, and let v_{ij} denote the vertex (v_i, v_j) in V_{ij}^A corresponding to the vertex $v = (v_1,\ldots,v_d)$ in V^A. In the network N_{ij}^A, the length of each edge (u_{ij}, v_{ij}) such that $u_{ij} = (u_i, u_j)$ and $v_{ij} = (v_i, v_j)$ is defined as follows based on the above observation on the sequence alignment problem of two dimensions:

$$l_{ij}^A(u_{ij}, v_{ij}) = \begin{cases} -w(c_{iv_i}, c_{jv_j}) & \text{if } u_i \neq v_i \text{ and } u_j \neq v_j, \\ -w(c_{iv_i}, \text{“-”}) & \text{if } u_i \neq v_i \text{ and } u_j = v_j, \\ -w(\text{“-”}, c_{jv_j}) & \text{if } u_i = v_i \text{ and } u_j \neq v_j. \end{cases}$$

Then the multiple sequence alignment corresponds to the shortest path problem for the network $N^A = (G^A, l^A)$ where

$$l^A(u,v) = \sum_{1 \leqslant i < j \leqslant d} l_{ij}^A(u_{ij}, v_{ij})$$

for each edge (u,v) in E^A. Since the score between characters is not necessary negative in practical sequence alignment problems, the network N_A is possible to have negative length edges.

The method based on Dynamic Programming (DP) is a traditional approach for the sequence alignment problem. Since the network has no cycle, this method can expand vertices in topological order. The shortest path from s is fixed for expanded vertices, and then it finishes searching when it finds the last vertex t. Thus it searches each vertex in the network only once. This method is applicable to the network with negative length edges. Although this method is sufficiently simple and is effective for the problem of two or three dimensions, it is impractical to apply this method directly to the problem of higher dimensions because the number of vertices $\prod_{i=1}^{d}(m_i+1)$ is enormous even for a little larger d.

3.2. Previous approach based on the A* algorithm

The A* paradigm has been already utilized for the multiple sequence alignment problem. Spouge has proposed the method to bound the search space of DP based on the concept of the A* algorithm [18]. This method specifies the vertices to be expanded by the A* algorithm utilizing the upper bound for the shortest path length from s to t. This section first reviews this application of the A* algorithm and discusses its merits and demerits.

The A* algorithm using a dual feasible estimator satisfies Proposition 2.4, and it finishes searching if it selects the destination vertex t as the vertex to be expanded. Hence, a vertex v expanded by this algorithm satisfies

$$L^*_{N^A}(s,v)+h(v) \leqslant L^*_{N^A}(s,t)+h(t) = L^*_{N^A}(s,t)$$

where h be a dual feasible estimator used in this algorithm. Let $L^+_N(u,v)$ denote the upper bound for the shortest path length from u to v in the network N. Then

$$L^*_{N^A}(s,t) \leqslant L^+_{N^A}(s,t),$$

and the shortest path from s to t does not pass any vertex v such that

$$L^*_{N^A}(s,v)+h(v) > L^+_{N^A}(s,t).$$

Spouge has utilized this concept for bounding the search space of DP and has proposed the algorithm based on DP which does not expand vertices satisfying above condition. This paper calls this algorithm as enhanced DP in convenience.

The merit of this approach is that it can omit the management of set $\overline{S}$ in the A* algorithm. The A* algorithm utilizes set $\overline{S}$ for maintaining the search order and apply the operations to extract the element with minimum key and to decrease the key of specified element to this set. Each operation takes $O(\log|\overline{S}|)$ time even if a binary heap is utilized as the implementation of $\overline{S}$. On the other hand, DP searches vertices in topological order and does not require additional data such as $\overline{S}$.

The demerit of enhanced DP is that it uses an upper bound of the shortest path length for bounding its search space. Although the effect of this bounding is due to tightness of the upper bound, it is expensive to obtain a tighter upper bound. If it takes a large amount of time to calculate an upper bound, the merit of enhanced DP will be canceled out. In addition to this, the number of expanded vertices in enhanced DP is larger than that in the A* algorithm unless the upper bound is tight. This number corresponds to the number of iterations in each algorithm, and are proportional to the execution time of each algorithm. As the upper bound becomes looser, the difference between them becomes larger. Although Spouge has proposed to modify the upper bound $L^+_{NA}(s,t)$ into $L^*_{NA}(s,v)+L^+_{NA}(v,t)$ dynamically when it expands some vertex v in order to make the upper bound tighter, it requires still more execution time.

These demerits of enhanced DP are due to the incompatibility between DP and the A* algorithm. In consideration of these demerits, this algorithm is not for practical use although its unique concept to apply the A* algorithm to DP is remarkable.

On the other hand, Araki et al. have directly applied the A* algorithm to the sequence alignment problem of two dimension [1]. In this case, the A* algorithm reduces 90% of the search space using the estimator constructed from information on scores between characters.

In these applications of the A* Algorithm, negative length edges in the network troubled the authors. Spouge has restricted the object of the application to a network without negative length edges. Araki et al. has proposed the method to make the length of edges in the network nonnegative. However, the A* Algorithm is applicable to the network with negative length edges based on Proposition 2.3 if the estimator in the algorithm is dual feasible. This paper does not pay special attention to the network with negative length edges with regard to the application of the A* algorithm from this point of view.

4. Enhanced A* algorithm for multiple alignment

4.1. Direct application of the A algorithm*

In this section, this paper discusses the direct application of the A* algorithm to the network for the multiple sequence alignment problem with attention to the estimator of the A* algorithm.

This paper adopts the same estimator has been proposed for enhanced DP by Spouge [18]. The following h^A denotes this estimator:

$$h^A(v) = \sum_{1\leqslant i<j\leqslant d} L^*_{NA_{ij}}(v_{ij},t_{ij}).$$

Recall that the length of a path in the multiple alignment problem is defined as the sum of all length of corresponding paths in pairwise alignment problems. This estimator utilizes the shortest path length in the pairwise alignment problem as the estimate for the length of the path corresponding to the shortest path in the multiple alignment problem. In the problem of higher dimensions, necessary time and space for solving pairwise

Table 1
Proteins used in computational experiments

Species	Protein	Length
Haloarcula Marismortui	EF-1α	421
Methanococcus Vannielii	EF-1α	428
Thermoplasma Acidophilum	EF-1α	424
Thermococcus Celer	EF-1α	428
Sulfolobus Acidocaldarius	EF-1α	435
Entamoeba Histolytica	EF-1α	430
Plasmodium Falciparum	EF-1α	443
Stylonychia Lemnae	EF-1α	446
Euglena Gracilis	EF-1α	445
Dictyostelium Discoideum	EF-1α	456
Lycopersicon Esculentum	EF-1α	448
Arabidopsis Thaliana	EF-1α	449
Absidia Glauca	EF-1α	458
Rhizomucor Racemosus	EF-1α	458
Candida Albicans	EF-1α	458
Saccharomyces Cerevisiae	EF-1α	458
Onchocerca Volvulus	EF-1α	464
Artemia Salina	EF-1α	462
Drosophila Melanogaster	EF-1α	463
Xenopus Laevis	EF-TU	462
Homo Sapiens	EF-TU	462

problems is negligible compared with those for solving an original problem. Therefore it is possible to use the estimator which provides more exact estimates utilizing information on all pairwise alignments.

This estimator h^A is dual feasible because

$$l^A_{ij}(u_{ij}, v_{ij}) + L^*_{N^A_{ij}}(v_{ij}, t_{ij}) \geqslant L^*_{N^A_{ij}}(u_{ij}, t_{ij})$$

in any network N^A_{ij} for the pairwise alignment of σ_i and σ_j. Hence the A* algorithm using this estimator is applicable to networks which have negative length edges based on Proposition 2.3. Thus this paper proposes the following approach which consists of two phases.

1. For arbitrary pair of i and j satisfying that $1 \leqslant i < j \leqslant d$, apply DP to network N^A_{ij} from vertex t_{ij} and calculate $L^*_{N^A_{ij}}(v_{ij}, t_{ij})$ for any v_{ij} in V^A_{ij}.
2. Apply the Dijkstra algorithm to the network $N^A_h = (G^A, l^A_h)$ from vertex s where

   $$l^A_h(u, v) = l^A(u, v) + h^A(v) - h^A(u)$$

   for each edge (u, v) in E^A.

In Phase 1, this approach utilizes DP for finding the shortest path length from any vertex to t in the network N^A_{ij}. In Phase 2, this algorithm applies the A* algorithm based on Proposition 2.3.

In order to investigate the actual efficiency of this approach, the experiment aligning actual sequences of proteins has been performed. In this experiment, optimal alignments

Table 2
The result of the experiment aligning actual sequences of proteins with the A* algorithm

d	3	4	5	6	7
Score	3970	7709	12314	18101	24912
$\|V\|$	7.6×10^7	3.3×10^{10}	1.4×10^{13}	6.1×10^{15}	2.7×10^{18}
Expanded	465	950	2731	5942	48521
Searched	3082	9094	32219	104267	838812
Time (s)	3	6	13	44	14174

of the first d sequences of eukaryotic and archaebacterial proteins in Table 1 have been calculated based on this approach for $3 \leqslant d \leqslant 7$. The PAM-250 matrix (integer-valued) has been used for giving scores between normal characters [4]. Since this matrix is symmetric, its upper right part is omitted. Each element in this matrix denotes the score between two characters corresponding to its row and column. With regard to scores between gaps and normal characters, the minimum value in the PAM-250 matrix, -8, has been used uniformly according to general methods.

Table 2 displays the score of the result optimal alignment (Score), the number of vertices in the network ($|V^A|$), the number of expanded vertices (Expanded), the number of searched vertices (Searched), and the execution time by SPARC Station 20 with 64 megabytes memory (Time) in this experiment. The number of expanded vertices corresponds to the size of S when algorithm finish searching, while the number of searched vertices corresponds to the size of $S \cup \overline{S}$ at that point. Then the number of searched vertices corresponds to the necessary space for the algorithm. The execution time stands for not the CPU time but the time spent for actual execution of the implemented program. It includes the time spent for preprocessing phase to calculate the shortest path length in the pairwise alignment problem.

The result of this experiment shows the algorithm searches significantly small number of vertices in this approach, while the number of vertices in the graph runs into astronomical figures. The optimal alignment of seven sequences with length about 430 is obtained by this algorithm. Fig. 2 shows this alignment. This is better than the result by Spouge that the optimal alignment of six sequences with length about 200 was obtained with enhanced DP. This result is mainly due to the fact that the estimator adopted in this approach provides rather exact estimate sufficiently close to the actual shortest path length. However, in spite of this effective bounding of the search space, the execution time increases and exceeds 3 hs for a seven-dimensional problem. This implies that this approach is effective for the multiple sequence alignment problem of at most about six dimensions in practice. The algorithm cannot defeat the exponential explosion of vertices in the network. Thus, this paper proposes more practical approach using an approximate method in the following section.

4.2. *Approximate method based on the A algorithm*

Ikeda et al. [12] shows that the method based on the A algorithm using a stronger estimator than the original estimator for the A* algorithm reduces the search space.

```
     1                                                                  80
Hal  MS-DEQHQNLAIIGHVDHGKSTLVGRLLYETGSVPEHVIEQHKEEAEEKGKGGFEFAYVMDNLAEERERGVTIDIAHQEF
Met  MAKTKPILNVAFIGHVDAGKSTTVGRLLLDGGAIDPQLIVRLRKEAEEKGKAGFEFAYVMDGLKEERERGVTIDVAHKKF
Tha  MASQKPHLNLITIGHVDHGKSTLVGRLLYEHGEIPAHIIEEYRKEAEQKGKATFEFAWVMDRFKEERERGVTIDLAHRKF
Thc  MAKEKPHINIVFIGHVDHGKSTTIGRLLFDTANIPENIIKKFE-EMGEKGK-SFKFAWVMDRLKEERERGITIDVAHTKF
Sul  MS-QKPHLNLIVIGHVDHGKSTLIGRLLMDRGFIDEKTVKEAEEAAKKLGKDSEKYAFLMDRLKEERERGVTINLSFMRF
Ent  MPKEKTHINIVVIGHVDSGKSTTTGHLIYKCGGIDQRTIEKFEKESAEMGKGSFKYAWVLDNLKAERERGITIDISLWKF
Pla  MGKEKTHINLVVIGHVDSGKSTTTGHIIYKLGGIDRRTIEKFEKESAEMGKGSFKYAWVLDKLKAERERGITIDIALWKF

     81                                                                 160
Hal  STDTYDFTIVDCPGHRDFVKNMITGASQADNAVLVVAA-D---D-GV-QP-QTQEHVFLARTLGIGELIVAVNKMDLVD-
Met  PTAKYEVTIVDCPGHRDFIKNMITGASQADAAVLVVNVDDA--KSGI-QP-QTREHVFLIRTLGVRQLAVAVNKMDTVN-
Tha  ETDKYYFTLIDAPGHRDFVKNMITGTSQADAAILVISARDG--E-GV-ME-QTREHAFLARTLGVPQMVVAINKMDATSP
Thc  ETPHRYITIIDAPGHRDFVKNMITGASQADAAVLVVAV-T---D-GV-MP-QTKEHAFLARTLGINNILVAVNKMDMVN-
Sul  ETRKYFFTVIDAPGHRDFVKNMITGASQADAAILVVSAKKGEYEAGMSAEGQTREHIILSKTMGINQVIVAINKMDLADT
Ent  ETSKYYFTIIDAPGHRDFIKNMITGTSQADVAILIVAAGTGEFEAGISKNGQTREHILLSYTLGVKQMIVGVNKMDAIQ-
Pla  ETPRYFFTVIDAPGHKDFIKNMITGTSQADVALLVVPADVGGFDGAFSKEGQTKEHVLLAFTLGVKQIVVGVNKMDTVK-

     161                                                                240
Hal  -YGESEYKQVVEEV-KDLLTQVRFDSENAKFIPVSAFEGDNIAEESEHTGWYDGEILLEALNELPAPEPPTDAPLRLPIQ
Met  -FSEADYNELKKMIGDQLLKMIGFNPEQINFVPVASLHGDNVFKKSERNPWYKGPTIAEVIDGFQPPEKPTNLPLRLPIQ
Tha  PYSEKRYNEVKADA-EKLLRSIGFK-D-ISFVPISGYKGDNVTKPSPNMPWYKGPTLLQALDAFKVPEKPINKPLRIPVE
Thc  -YDEKKFKAVAEQV-KKLLMMLGYK-N-FPIIPISAWEGDNVVKKSDKMPWYNGPTLIEALDQMPEPPKPTDKPLRIPIQ
Sul  PYDEKRFKEIVDTV-SKFMKSFGFDMNKVKFVPVVAPDGDNVTHKSTKMPWYNGPTLEELLDQLEIPPKPVDKPLRIPIQ
Ent  -YKQERYEEIKKEI-SAFLKKTGYNPDKIPFVPISGFQGDNMIEPSTNMPWYKGPTLIGALDSVTPPERPVDKPLRLPLQ
Pla  -YSEDRYEEIKKEV-KDYLKKVGYQADKVDFIPISGFEGDNLIEKSDKTPWYKGRTLIEALDTMQPPKRPYDKPLRIPLQ

     241                                                                320
Hal  DVYTISGIGTVPVGRVETGILNTGDNVSFQPSD-V----S-GEVKTVEMHHEEVPKAEPGDNVGFNVRGVGKDDIRRGDV
Met  DVYTITGVGTVPVGRVETGIIKPGDKVVFEPAG-A----I-GEIKTVEMHHEQLPSAEPGDNIGFNVRGVGKKDIKRGDV
Tha  DVYSITGIGTVPVGRVETGVLKPGDKVIFLPAD-K----Q-GDVKSIEMHHEPLQQAEPGDNIGFNVRGIAKNDIKRGDV
Thc  DVYSIKGVGTVPVGRVETGVLRVGDVVIFEPASTIFHKPIQGEVKSIEMHHEPMQEALPGDNIGFNVRGVGKNDIKRGDV
Sul  EVYSISGVGVVPVGRIESGVLKVGDKIVFMPVG-K----I-GEVRSIETHHTKIDKAEPGDNIGFNVRGVEKKDVKRGDV
Ent  DVYKISGIGTVPVGRVETGILKPGTIVQFAPSG-V----S-SECKSIEMHHTALAQAIPGDNVGFNVRNLTVKDIKRGNV
Pla  GVYKIGGIGTVPVGRVETGILKAGMVLNFAPSA-V----V-SECKSVEMHKEVLEEARPGDNIGFNVKNVSVKEIKRGYV

     321                                                                400
Hal  CGPADDPPSVA--ET-FQAQIVVMQHPSVITEGYTPVFHAHTAQVACTVESIDKKIDPSSGEVAE-ENPDFIQNGDAAVV
Met  LGHTTNPPTVA--TD-FTAQIVVLQHPSVLTDGYTPVFHTHTAQIACTFAEIQKKLNPATGEVLE-ENPDFLKAGDAAIV
Tha  CGHLDTPPTVV--KA-FTAQIIVLNHPSVIAPGYKPVFHVHTAQVACRIDEIVKTLNPKDGTTLK-EKPDFIKNGDVAIV
Thc  AGHTNNPPTVVRPKDTFKAQIIVLNHPTAITVGYTPVLHAHTLQVAVRFEQLLAKLDPRTGNIVE-ENPQFIKTGDSAIV
Sul  AGSVQNPPTVA--DE-FTAQVIVIWHPTAVGVGYTPVLHVHTASIACRVSEITSRIDPKTGKEAE-KNPQFIKAGDSAIV
Ent  ASDAKNQPAVG-CED-FTAQVIVMNHPGQIRKGYTPVLDCHTSHIACKFEELLSKIDRRTGKSMEGGEPEYIKNGDSALV
Pla  ASDTKNEPAKG-CSK-FTAQVIILNHPGEIKNGYTPLLDCHTSHISCKFLNIDSKIDKRSGKVVE-ENPKAIKSGDSALV

     401                                                455
Hal  TVRPQKPLSIEPSSEIPELGSFAIRDMGQTIAAGKV-------LG-VN-E----R
Met  KLIPTKPMVIESVKEIPQLGRFAIRDMGMTVAAGMA-------IQ-VTAK--N-K
Tha  KVIPDKPLVIEKVSEIPQLGRFAVLDMGQTVAAGQC-------ID-LE-K----R
Thc  VLRPTKPMVIEPVKEIPQMGRFAIRDMGQTVAAGMV-------IS-IQ-K--A-E
Sul  KFKPIKELVAEKFREFPALGRFAMRDMGKTVGVGVI-------ID-VKPRKVEVK
Ent  KIVPTKPLCVEEFAKFPPLGRFAVRDMKQTVAVGVV-------KA-VT------P
Pla  SLEPKKPMVVETFTEYPPLGRFAIRDMRQTIAVGIINQLKRKNLGAVTAKAPAKK
```

Fig. 2. The optimal alignment of seven sequences of proteins obtained by the A* algorithm.

This paper applies the same method to the multiple sequence alignment problem. In other words, the A algorithm using the following estimator h_k^A is applied to the multiple sequence alignment problem where k is some constant:

$$h_k^A(v) = k \cdot h^A(v) = k \sum_{1 \leqslant i < j \leqslant d} L^*_{N_{ij}^A}(v_{ij}, t_{ij}).$$

This algorithm is not the A* algorithm if k is more than one, because h_k^A breaks the condition of the A* algorithm given by expression (1). If k equals to one, it obviously corresponds to the A* algorithm.

The estimator h_k^A is k times as large as the original estimator h^A adopted in the A* algorithm. This means that this algorithm regards the cost from a vertex to t k times more important than the cost from s to the vertex. Hence, this algorithm searches vertices near t preferentially if the estimate $h_k^A(v)$ is nonnegative for each vertex v in V^A. If there exists a vertex v such that $h_k^A(v)$ is negative, this algorithm does not provide expected results. In this case, the estimated cost from the vertex v to t, $h_k^A(v)$, is smaller than the estimated cost from the vertex t to t, $h_k^A(t)$, which equals to zero. This means that the vertex v is estimated to be nearer to t than t itself. Hence this algorithm searches vertices near v more preferentially than vertices near t.

This paper makes each edge length in the network nonnegative based on the method proposed by Araki et al. [1], and applies the A algorithm to modified network without negative length edges. Although their method is for a two-dimensional problem, it is easily extended for the multiple sequence alignment problem.

Let w^* be the maximum score between characters,

$$\max_{x,y\in C}\{w(x,y)\}.$$

In the network $N_{ij}^{A+} = (G_{ij}^A = (V_{ij}^A, E_{ij}^A), l_{ij}^{A+})$ for the pairwise alignment of σ_i and σ_j where

$$l_{ij}^{A+}(u_{ij},v_{ij}) = \begin{cases} 2*w^* - w(c_{iv_i}, c_{jv_j}) & \text{if } u_i \neq v_i \text{ and } u_j \neq v_j, \\ w^* - w(c_{iv_i}, \text{“-”}) & \text{if } u_i \neq v_i \text{ and } u_j = v_j, \\ w^* - w(\text{“-”}, c_{jv_j}) & \text{if } u_i = v_i \text{ and } u_j \neq v_j \end{cases}$$

for any edge (u_{ij}, v_{ij}) in E_{ij}, any path P from $s_{ij} = (0,0)$ to $t_{ij} = (m_i, m_j)$ satisfies the following condition:

$$\begin{aligned} L_{N_{ij}^{A+}}^P(s_{ij},t_{ij}) &= \sum_{(u_{ij},v_{ij})\in P} l_{ij}^{A+}(u_{ij},v_{ij}) \\ &= w^*(m_i+m_j) + \sum_{(u_{ij},v_{ij})\in P} l_{ij}^{A}(u_{ij},v_{ij}) \\ &= w^*(m_i+m_j) + L_{N_{ij}^A}^P(s_{ij},t_{ij}). \end{aligned}$$

Similarly, in the network $N^{A+} = (G^A = (V^A, E^A), l^{A+})$ for the multiple sequence alignment problem where

$$l^{A+}(u,v) = \sum_{1\leqslant i<j\leqslant d} l_{ij}^{A+}(u_{ij},v_{ij})$$

for any edge (u,v) in E, any path P from $s = (0,\ldots,0)$ to $t = (m_1,\ldots,m_d)$ satisfies the following condition:

$$L_{N^{A+}}^P(s,t) = \sum_{(u,v)\in P} l^{A+}(u,v)$$

Table 3
The result of the experiment aligning seven sequences of proteins with A algorithm using k times stronger estimator

k	1	1.001	1.005	1.01	1.05	1.1
Score	24912	24912	24903	24880	24692	24060
Expanded	48521	965	618	492	456	456
Searched	838812	77384	64364	58495	55935	55686
Time (s)	14174	30	26	23	23	23

Table 4
The result of the experiment aligning sequences of proteins with A algorithm using the estimator $h^A_{1.01}$

d	3	4	5	6	7	8	9	10
Score	3790	7709	12298	18085	24889	33108	43009	54162
Expanded	431	451	454	454	495	605	911	3954
Searched	2988	6547	13631	27804	57909	125733	292010	1074066
Time (s)	3	5	9	14	23	47	119	5732

$$= w^* \sum_{i=1}^{d} m_i + \sum_{(u,v)\in P} l^A(u,v)$$

$$= w^* \sum_{i=1}^{d} m_i + L^P_{N^A}(s,t).$$

This indicates that the shortest path from s to t in N^{A+} corresponds to the shortest path from s to t in N^A. Each edge length is clearly nonnegative in the network N^{A+}. Hence, $h^A_k(v)$ is nonnegative for any vertices in this network, and the A algorithm using this estimator is valid for this network.

Table 3 displays the result of the application of this algorithm to first seven sequences in Table 1. The case that $k = 1$ corresponds to applying the A* algorithm and the score for this case is the optimal. The result shows the great effect of this algorithm. While searched vertices are reduced with a little larger k than unity, the score of the alignment decreases little. Particularly in the case $k = 1.001$, the optimal alignment is obtained in 30 s with only 965 vertices expanded.

Table 4 shows the result of the experiment to apply the A algorithm using the estimator $h^A_{1.01}$ to first d sequences in Table 1 for $3 \leqslant d \leqslant 10$. This result indicates that this algorithm expands much less vertices than the A* algorithm. This algorithm produces an alignment of nine sequences in practical time. However, the number of searched vertices increases excessively as the dimension grows, and execution time for the ten-dimensional problem exceeds one hour. This implies that this algorithm also requires impractical amount of time for problems of higher dimensions.

On the other hand, the number of searched vertices is much less than the number of expanded vertices and increases slowly as the dimension grows. This means that the algorithm searches a large number of unnecessary vertices since the result path from s to t only passes vertices expanded by the algorithm. This phenomenon is mainly due to the fact that the A algorithm searches all $2^d - 1$ descendant vertices when it expands

Table 5
The result of the experiment aligning actual sequences of proteins with the A* algorithm using an upper bound for the shortest path length

d	3	4	5	6	7
Expanded	465	945	2771	5881	48 755
Searched	465	979	2804	6032	48 998
Time (s)	2	6	12	20	201

a vertex and is also observed in the result for the application of the A* algorithm. In the following section, this paper presents the method to cope with this problem.

4.3. Improvement of algorithms: Enhanced A*

4.3.1. Utilization of upper bound

This section proposes the approach to utilize the upper bound for the shortest path length for bounding the search space like enhanced DP. Recall that the A* algorithm expands only such a vertex v that

$$L^*_{N^A}(s,v) + h^A(v) \leqslant L^+_{N^A}(s,t).$$

This indicates that the vertex violating this condition remains not expanded if the vertex is searched by the A* algorithm. Hence it is meaningless and needless to add such a vertex v that

$$p(v) + h^A(v) \leqslant L^+_{N^A}(s,t)$$

to $\overline{S}$ in Step 2.4 in Algorithm A. The same approach is also applicable to the A algorithm using the estimator $h^A_k(v)$.

In this approach, although the algorithm still investigates all 2^d-1 descendant vertices when it expands a vertex in order to verify whether each descendant vertex satisfies above condition, the necessary space for this algorithm sufficiently decreases because $\overline{S}$ does not contain any vertex breaking the condition.

Table 5 displays the result of the experiment applying the A* algorithm based on this approach to first d sequences in Table 1 for $3 \leqslant d \leqslant 7$. The number of searched vertices means the number of vertices added to $\overline{S}$. In this experiment, the actual shortest path length has been utilized as the upper bound in order to examine the best possible case. Each execution time does not contain the time for obtaining the upper bound. The result shows this approach sufficiently prevents the A* algorithm from adding unnecessary vertices to $\overline{S}$. The number of searched vertices is close to the number of expanded vertices in all cases. The efficiency of the A* algorithm is significantly improved especially for higher dimensional case. In fact, the seven-dimensional problem is solved in four minutes. Although there remains the problem how to obtain a tighter upper bound in sufficiently small time, the A algorithm proposed in the previous section provides the solution for this problem.

Table 6 shows the result of the experiment using the A algorithm with the estimator $h^A_{1.01}$ instead of the A* algorithm in the previous experiments. In this experiment,

Table 6
The result of the experiment aligning actual sequences of proteins with the A algorithm with the estimator $h^A_{1.01}$ using an upper bound for the shortest path length

d	3	4	5	6	7	8	9	10
Expanded	431	451	454	454	495	605	911	3954
Searched	454	803	2072	3767	14 975	51 569	163 208	504 845
Time (s)	3	4	8	13	23	49	135	1258

the length of the path obtained by the A algorithm using the estimator $h^A_{1.01}$ without bounding the search space is adopted as the upper bound for the shortest length. The result implies that bounding the search space based on this approach has a small effect on the A algorithm. Although the number of searched vertices also decreases in this case, it is still much larger than the the number of expanded vertices. This is due to the fact that this approach specifies the vertices to be searched as the vertices necessarily expanded by the A* algorithm. Since the A algorithm using the estimator h^A_k originally expands less vertices than the A* algorithm, this approach is not effective for it.

4.3.2. Modification of network structure

This section presents another approach to the problem that the algorithm searches all $2^d - 1$ descendant vertices when it expands a vertex. This section proposes a new searching strategy for the multiple sequence problem, and modifies the structure of the network based on this searching strategy.

In the network $N^A = (G^A = (V^A, E^A), l^A)$, each edge represents a part of the alignment in a column. When the algorithm expands a vertex, it investigates all $2^d - 1$ descendant vertices. This corresponds to the operation to investigate all possible alignment in the next column. The algorithm composes all possible states where the character in the next column are fixed for all d sequences, and searches all the states. Thus, this searching strategy decides the setting of all d characters at once.

On the other hand, it is possible to adopt the searching strategy which decides the setting of one character at once. The algorithm based on this searching strategy composes the states where a character to be placed in the next column is newly fixed for one sequence, and searches these states. There exists only two characters to be placed in the next column, the next character in the sequence or a gap character. Hence, in the network corresponding to this searching strategy, each vertex has at most two descendant vertices. The length corresponding to the additional score according to the increase of pairwise alignments in a column where two characters are fixed is assigned to each edge in this network.

The network $\overline{N}^A = (\overline{G}^A = (\overline{V}^A, \overline{E}^A), \overline{l}^A)$ such that

$$\overline{G}^A = \{(x_1, \ldots, x_d)_{\alpha_1 \cdots \alpha_n} \mid x_i = 0, \ldots, m_i \text{ for all } 0 \leqslant i \leqslant d,$$
$$0 \leqslant n \leqslant d,\ \alpha_i = 0, 1 \text{ for all } 0 \leqslant i \leqslant n\},$$

$$\overline{E}^A = \{(u,v) \mid u = (x_1,\ldots,x_d)_{\alpha_1\cdots\alpha_{n-1}} \in V,\ v = (x_1,\ldots,x_d)_{\alpha_1\cdots\alpha_n} \in V,$$
$$\alpha_1\cdots\alpha_n \neq 0\cdots 0 \text{ if } n = d\},$$
$$\overline{l}^A((x_1,\ldots,x_d)_{\alpha_1\cdots\alpha_{n-1}},(x_1,\ldots,x_d)_{\alpha_1\cdots\alpha_n}) = \sum_{i=1}^{n-1} l_{in}^A((x_i,x_n),(x_i+\alpha_i,x_n+\alpha_n))$$

where $(x_1,\ldots,x_d)_{\alpha_1\cdots\alpha_d}$ and $(x_1+\alpha_1,\ldots,x_d+\alpha_d)$ denote the same vertex corresponding to above searching strategy. A vertex $(x_1,\ldots,x_d)_{\alpha_1\cdots\alpha_n}$ stands for the state where the first x_i characters of each sequence σ_i are aligned in the same part of the result alignment and the characters set to the next column are fixed for first n sequences. Each α_i denotes this fixed character for the sequence σ_i. A gap character is placed in the next column if α_i equals to zero, and the $(x_i+\alpha_i)$th character if α_i equals to one.

In the network N^A, 2^d-1 edges go out of a vertex and constitute a directed tree with depth one and degree 2^d-1. This part is modified into a binary tree with depth d in the network $\overline{N}^A$. This approach aims to keep the number of searched vertices nearer to the number of expanded vertices subdividing the edges in N^A and unifying the common parts. Although the network $\overline{N}^A$ has vertices 2^d-1 times as many as the network N^A, the algorithm searches only two vertices in $\overline{N}^A$ when it expands a vertex.

The following estimator $\overline{h}^A$ for the network $\overline{N}^A$ is clearly dual feasible from the definition of the edge length function $\overline{l}^A$:

$$\begin{aligned}\overline{h}^A((x_1,\ldots,x_d)_{\alpha_1\cdots\alpha_n}) = &\sum_{1\leqslant i<j\leqslant n} L^*_{N^A_{ij}}((x_i+\alpha_i,x_j+\alpha_j),(m_i,m_j))\\ &+\sum_{1\leqslant i\leqslant n<j\leqslant d} L^*_{N^A_{ij}}((x_i+\alpha_i,x_j),(m_i,m_j))\\ &+\sum_{n\leqslant i<j\leqslant d} L^*_{N^A_{ij}}((x_i,x_j),(m_i,m_j)).\end{aligned}$$

Hence the A* algorithm is applicable to this network and the A algorithm utilizing the k times stronger estimator $\overline{l}^A_k$ such that $\overline{l}^A_k(v) = k\cdot\overline{l}^A(v)$ for all vertices in $\overline{E}^A$ is also usable for this network. In this case, applying the same approach to the network N^{A+} instead of N^A solves the problem on negative length edges.

Table 7 shows the result of the experiment applying the A* algorithm to the network $\overline{N}^A$ for aligning first d sequences in Table 1 for $3\leqslant 7$. Although the number of searched vertices increases from the result for the original network N^A because of newly introduced vertices in this network, the execution time decreases for the problem of less than seven dimensions. However, the algorithm requires much more time for the seven-dimensional problem. The number of searched vertices is at most twice as many as the number of expanded vertices even in this case. This implies that the increase of vertices in the network has a bad influence on the efficiency of the algorithm if the algorithm expands a large number of vertices.

On the other hand, this approach is effective for the A algorithm using a stronger estimator. Table 8 shows the result of the experiment with the A algorithm using the estimator $\overline{h}^A_{1.01}$. In this case, the number of searched vertices sufficiently decreases and the alignment of ten sequences are obtained in only about one minute. The score of

Table 7
The result of the experiment aligning actual sequences of proteins with the A* algorithm for the network $\overline{N}^A$

d	3	4	5	6	7
Expanded	2336	8887	41 035	128 772	1 759 637
Searched	4132	15 073	66 033	211 247	2 731 262
Time (s)	2	6	12	29	47 722

Table 8
The result of the experiment aligning actual sequences of proteins with the A algorithm with the estimator $\overline{h}^A_{1.01}$ for the network $\overline{N}^A$

d	3	4	5	6	7	8	9	10
Score	3970	7709	12 302	18 092	24 895	33 094	43 018	54 180
Expanded	1729	2336	2967	3426	4773	8752	19 834	155 540
Searched	3446	4661	5853	6786	9333	16 803	37 228	286 357
Time (s)	3	5	8	11	16	22	30	65

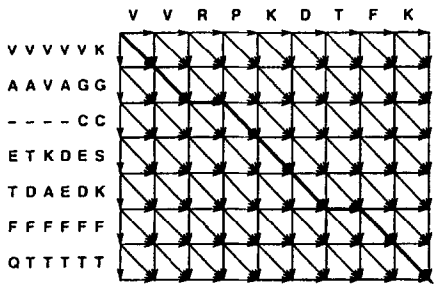
the result alignment is not inferior to that with the A algorithm for the network N^{A+}. Moreover, this approach enables to align all 21 sequences in Table 1 in a few minutes.

5. Group alignment and its use in iterative algorithms

5.1. k-Group alignment

Since the multiple alignment problem becomes hard to solve when d is large, as a subproblem, the group alignment is considered. Originally, in the group alignment, d sequences are divided into two groups, say d' sequences and $d - d'$ sequences ($0 < d' < d$), and then fixing the alignment in each group, it solves a two-dimensional alignment problem between two groups. In this two-dimensional problem, since the alignment in each group is fixed, when a gap is inserted into a group, it is simultaneously inserted in the same position; see Fig. 3.

For general $k > 2$ ($k \leq d$), the k-group alignment problem can be defined similarly. In this problem, d sequences are given as k disjoint groups, and each group is associated with some alignment of sequences in the group. In a typical case, for an alignment of d sequences, k aligned groups can be obtained simply by dividing this alignment into k groups (and removing trivial gaps inside each group alignment). Then, the k-group alignment problem finds a best-score alignment of d sequences under a condition that, in each group, each column of its alignment should be fixed. Hence, when a gap is inserted into a sequence in some group, the same gap should be inserted in the same position in every sequence in the group. In this case, a k-dimensional grid-like graph is used to solve the k-group alignment problem, as in the original k-dimensional alignment problem. See an example of $k = 2$ and $d = 7$ in Fig. 3. Since the score function is defined to be the sum of scores of all pairs, the A* approach can be directly extended to this k-group alignment problem by virtue of the principle of optimality.



(a)

```
Thc VVRPKDTFK
Tha VV--KA-FT
Hal VA--ET-FQ
Met VA--TD-FT
Sul VA--DE-FT
Ent VG-CED-FT
Pla KG-CSK-FT
```

(b)

Fig. 3. (a) Aligning seven sequences by a 2-group alignment for Thc and others; (b) an obtained alignment (- is a newly inserted gap).

Two types of A* search algorithm may be considered for k-group alignment.

- (Group-based precomputing strategy) For each pair of groups, compute the score of 2-group alignment between the two groups, and make the summation of those scores as a lower-bound estimator in solving the k-group alignment problem.
- (Sequence-based preprocessing strategy) In the initial stage of the algorithm, solve 2-dimensional alignment between every pair of sequences as preprocessing. Then, in solving each k-group alignment problem, compute the summation of scores of all pairs of sequences contained in different groups.

The former A* estimate is stronger than the latter. On the other hand, to obtain the former estimate we may have to solve $\binom{k}{2}$ 2-group alignment problems, while in the latter only preprocessing in the beginning is sufficient. Even when solving 2-group alignment problems between general two groups, we can make use of the scores of all-pairs of sequences computed in the preprocessing stage to solve it by the A* search by the sequence-based strategy.

5.2. *Iterative improvement*

The standard randomized iterative improvement method proposed by Berger and Munson [2] works as follows:

1. Construct an initial alignment by some method.
2. Divide d sequences into two groups randomly.
3. Remove trivial gaps in each group.
4. Solve the 2-group alignment to obtain a new alignment.
5. If the score decreases, update the current alignment to the new one.
6. If a stopping condition is met, stop; otherwise return to step 2.

In the step 2 above, all the sequences are divided into two groups randomly. There is a method of dividing them into a group of one sequence and a group consisting of the other $d-1$ sequences. This partition is called a restricted partition in [9]. Also in

that case, instead of using randomization, one sequence for a group of a single element may be changed in a round-robin fashion. There are many other methods (see [9]).

It is rather natural to extend the iterative algorithm in a way that in step 2 it divides the sequences into k groups for $k \geq 2$. We call such iterative algorithm the k-group iterative algorithm. The A* algorithm for group alignments can be utilized in the k-group algorithm. Two methods of dividing d sequences into k groups can be considered by directly generalizing the above-mentioned existing methods for $k = 2$.

- (k-random-grouping; simply called *random*) This method divides d sequences randomly into k groups so that no group becomes empty. This will be denoted by RA(k) in the computational results below.
- (k-restricted-grouping; simply called *restricted*) This method divides them into $k-1$ groups consisting of a single sequence and another group consisting of the other sequences where each sequence in the former $k-1$ groups is chosen randomly. This will be denoted by RI(k) in the next section.

In [8], the latter method with $k = 2$ was used to derive a best-first iterative improvement algorithm, and it was observed that the restricted-grouping strategy produces favorably nice solutions compared to the random-grouping method.

When the A* is used in the iterative improvement algorithm with the restricted-grouping strategy, solving $\binom{k-1}{2}$ subproblems out of $\binom{k}{2}$ can be dispensed with if we do the same preprocessing as in sequence-based preprocessing strategy in the preceding subsection.

5.3. Computational results

In order to investigate the actual efficiency of this approach, experiments aligning actual sequences of proteins have been performed. Our implementation is designed to evaluate several strategies in a system, and is coded in such a general setting. This causes in some places redundant computation which can be avoided by cleverly using the information obtained in the preprocessing stage. This point should be remarked especially in observing timing results in the computational results. For example, the following points can be improved further from the current code.

- When the A* is used in the iterative improvement algorithm with the restricted-grouping, as described in the preceding section, solving $\binom{k-1}{2}$ subproblems out of $\binom{k}{2}$ can be dispensed with. However, in the current implementation, these subproblems are solved from scratch every time.
- Even when solving 2-group alignment problems between general two groups, we can make use of the scores of all-pairs of sequences computed in the preprocessing stage to solve it by A* search. However, we do not incorporate this in the code. Instead, a kind of lazy two-dimensional DP algorithm is implemented. Here, "lazy" execution may be done in many ways. For example, in the beginning only a fraction of DP table is computed, say in a constant-bandwidth region, and values of other elements are computed when necessary.

- In the experiment, the linear gap system is used instead of the affine gap system (see below). To obtain more meaningful alignments, the affine gap system is considered to be better, and performing the experiment with the affine gap system is important. For d-dimensional DP alignments, it is known that with the affine gap system it takes more time compared with the case of the linear gap system as d increases. From this, one might think that k-group alignment method by A* with the affine gap system performs much worse than that with the linear gap system as k becomes larger, but we suspect that for $k = 3$ (and maybe 4,5) the slow-down would be a small constant factor since by A* the search space is drastically cut off. Of course, this point should be actually tested, which is left as future work. Concerning the performance of our A* approach, we suspect that changing the alignment cost system from the simple pairwise sum of 2-alignments to the weighted sum may affect more.

Concerning the score matrix, the PAM-250 matrix has been used in assigning edge length with each sign of score reversed. The linear gap system ax for the gap of length x is used (extending the current system to the affine gap system $ax + b$ for the gap of length x would be practically important work). With regard to the gap penalty, the minimum value in the PAM-250 matrix, $a = -8$, has been adopted. All the experiments were done on SPARCStation 20 with 128 megabytes memory.

5.3.1. Case with higher similarity

In this experiment, elongation factor TU (EF-TU) and elongation factor 1α (EF-1α) are used as in the previous section. The number d of sequences is 21, and the length n of each sequence is about 450. The cost of the best alignment found by the experiment is 294 813 with length 482. The average score per amino pair is $294\,813/482 \cdot \binom{21}{2} \approx 2.91$, and is higher than in the experiment in the next subsection.

As an initial alignment, we adopted a solution of the A algorithm described in the previous section with parameter 81/80. The score of this initial alignment is 294 201. By using a tree-based DP, better initial solutions can be obtained, but those solutions are processed by group DP, while the solution by the A algorithm is not. Starting with the solution by the A algorithm, the alignment score is improved fast initially.

Starting from this initial solution, we tested 10 series of 100 iterations by using different random numbers. Both of the k-random-grouping RA(k) and k-restricted-grouping RI(k) are examined. Some of computational results are given in Fig. 4. Box plots are used, where a box plot comprises of these elements: (1) a box with (1a) a central line showing the median, (1b) a lower line showing the first quartile, (1c) an upper line showing the third quartile; (2) 2 lines extending from the central box of maximal length 3/2 the interquartile range but not extending past the range of the data; (3) outliers, points that lie outside the extent of the previous elements.

For smaller k such as 2 and 3, the k-random-grouping method RA(k) tends to produce worse solutions than the k-restricted-grouping method RI(k). For large k such as 4 and 5, this tendency becomes less clear. In these experiments, the 3-restricted grouping method may be said to be the best one in regard to both computation time

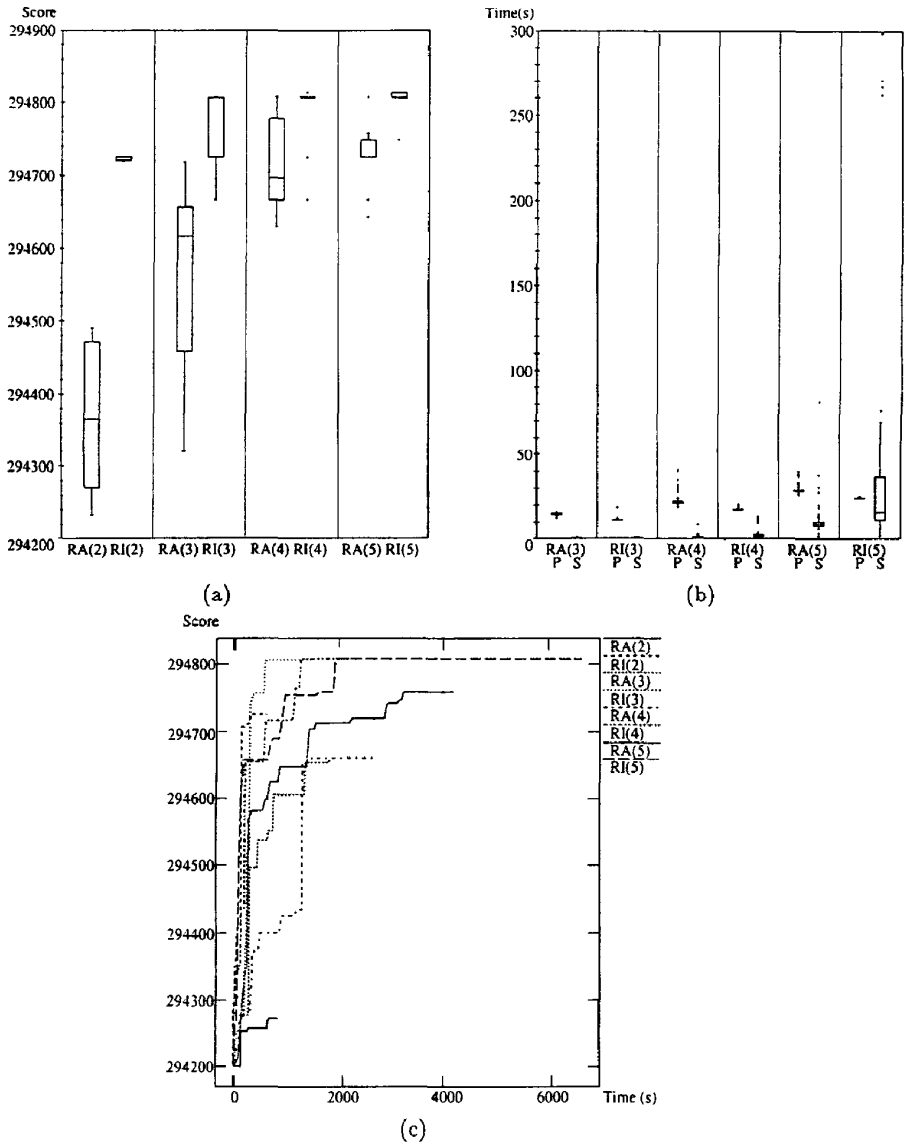


Fig. 4. Computational results for EF-1α: (a) A box plot of final scores after 100 iterations (10 series), (b) A box plot of running times for the estimator precomputation ("P" in short; on the left side) and A* search ("S" and on the right side) per step in the computation of typical series in (c), (c) Score improvement processes of typical series such that its final score attains the median score among the first three trials for each k and strategy.

and solution quality. Regarding the solution quality, the 3-restricted grouping produces better solutions than 2-restricted grouping, as seen in Fig. 4(a) and even if we use the scores of the 2-restricted grouping with 200 iterations, results are almost same. Here, it should be stressed that this speed of 3-group alignment is achieved by the use of

A*. It is also observed that, even for $k = 4, 5$, k-alignment problem can be solved in a reasonable time even for this rather large-scale problem. Concerning the effect of A*, for $k = 3$ (and 4), its estimates are nice enough to reduce the search space drastically. In fact, the A* search takes much less time than the precomputation of estimator in this case (Fig. 4(b)). For $k = 5$, the estimates become less effective, and sometimes the A* search space becomes large, although it is still manageable.

5.3.2. Case with lower similarity

In the next experiment, $d = 9$ sequences of Chitin Synthase of lengths 710, 717, 723, 728, 730, 732, 756, 760, 762 are investigated. The best alignment score found by repeated applications of RI(4) is 59606 with length 811. The average score per amino pair is $59606/811 \cdot \binom{9}{2} \approx 2.04$, and is lower than the number 2.91 for EF-1α above. In this respect, the similarity is less compared with EF-1α, but each sequence is about twice longer.

We adopted a left-aligned alignment as an initial one. Since the initial alignment is simply left-aligned, the initial score is bad and -25992. Even for this bad initial alignment, 3-group alignments with A* can be performed efficiently, and yet 4-group alignment for such a bad alignment is hard to execute with A*. The score improvement processes for 2- and 3-group alignments are shown in Fig. 5(c).

To see the effectiveness of 4-group alignments, we adopted another initial alignment for $k = 4$ obtained by a series of 3-restricted-grouping iterative improvement alignment after 50 iterations starting with the above-mentioned bad alignment. This alignment has score 59 409. Note that there are many ways of obtaining rather good initial solutions, and in this regard this kind of initial alignment for $k = 4$ is easily obtained. After 50 iterations from this initial solution for $k = 4$, the 3-restricted-grouping method finds an alignment of score 59 499 (the best score among three trials of 100 iterations using different random numbers for $k = 2, 3$). Starting with this initial solution for $k = 4$, 4-group alignments RI(4) are used for the iterative algorithm with 20 iterations. Some of computational results are shown in Figs. 5(a) and (b) where box plots are again used (see the explanation in the previous subsection).

It would be rather striking that even for this set of long sequences with lower similarity, the A* method solves the 3-alignment problems very well, as in the above experiments. Again, the actual A* search time is less than the precomputation time for its estimator. Also, 4-group alignment could be run fast enough to improve nice solutions more. Since the similarity is relatively low, the A* search often requires more time than the estimator precomputation time for $k = 4$ here, unlike it starts to occur for $k = 5$ in the preceding case of high similarity.

5.3.3. Much larger cases

We also perform computational experiments to see how large-size problems this k-group alignment approach can handle. We have tested two cases:[3]

[3] The authors would like to thank Dr. Cao of TIT for kindly providing these test data for our program.

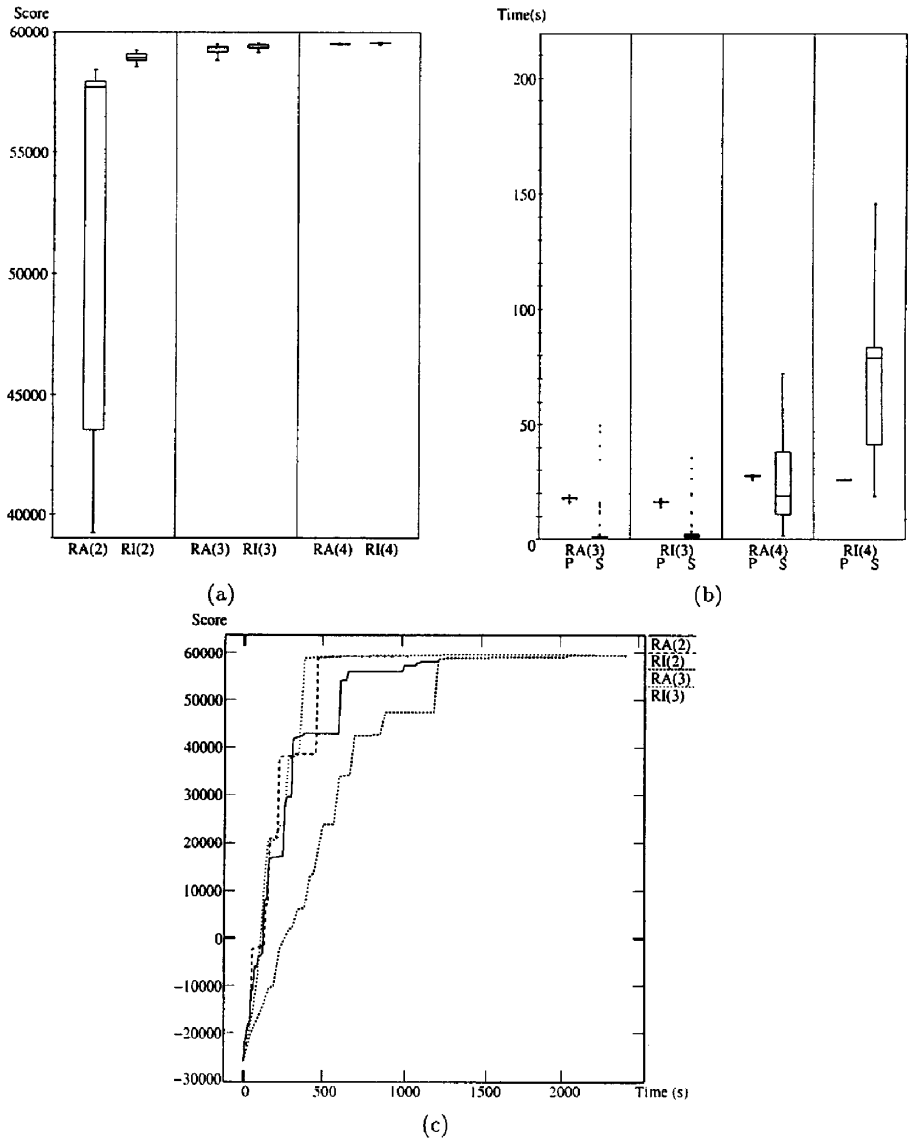


Fig. 5. Computational results for 9 sequences of Chitin Synthase: (a) A box plot of final scores (100 iterations from the left-aligned initial solution for $k = 2,3$; 20 iterations from the intermediate one for $k = 4$; 10 series for each), (b) A box plot of running times for estimate precomputation ("P"; left) and A* search ("S"; right) per step in the computation of typical series in (a,c), (c) Score improvement processes of typical series from the left-aligned initial alignment such that its final score attains the median score among the first three trials for each k and strategy,

1. Mitochondrial genome protein sequences of 34 species in Table 9, where the first one is an artificial one denoting high matches in this case.
2. Elongation factor protein sequences of 64 species in Table 10.

Table 9
34 protein sequences of mitochondrial genome

Gap #
SB17F Homo sapiens (African) # D38112
CHIMP Pan troglodytes (chimpanzee) # D38113
PyGC Pan paniscus (bonobo) # D38116
GORIL Gorilla gorilla (gorilla) # D38114
ORANG Pongo pygmaeus (orangutan) # D38115
Hylla Hylobates lar (common gibbon) # X99256 (lar gibbon)
Ponpy Pongo pygmaeus abelii (Sumatran orangutan) # X97707
Phovi Phoca vitulina (harbor seal) # X63726
Halgr Halichoerus grypus (grey seal) # X72004
Felca Felis catus (cat) # U20753
Equca Equus caballus (horse) # X79547
Rhiun Rhinoceros unicornis (Indian rhinoceros) # X97336
Bosta Bos taurus (cow) # J01394
Balph Balaenoptera physalus (fin whale) # X61145
Balmu Balaenoptera musculus (blue whale) # X72204
Orycu Oryctolagus cuniculus (rabbit) # Gissi
Cavpo Cavia procellus (guinea pig) # D'Erchia
Ratno Rattus norvegicus (rat) # X14848
Musmu Mus musculus (mouse) # J01420
Erieu Erinaceus europaeus (hedgehog) # X88898
Macro Macropus robustus (wallaroo) # Y10524
Didvi Didelphis virginiana (opossum) # Z29573
Ornan Ornithorhynchus anatinus (platypus) # X83427
Galga Gallus gallus (chicken) # X52392
Xenla Xenopus laevis (African clawed toad) # M10217
Prodo Protopterus dolloi (lungfish) # L42813
Latch Latimeria chalumnae (coelacanth) # U82228
Cypca Cyprinus carpio (carp) # X61010
Crola Crossostoma lacustre (loach) # M91245
Oncmy Oncorhynchus mykiss (trout) # L29771
Gadmo Gadus morhua (Atlantic cod) # X99772
Polor Polypterus ornatipinnis (bichir) # U62532
Petma Petromyzon marinus (sea lamprey) # U11880

For these data, we used Workstation Ultra 1 Model 170E with 256MB memory. For the first case, we could apply 10 to 12-group alignments, and for the second case, 5 to 6-group alignments, both in practical time. These two cases might be the cases with very high homology, but the computational results would show that our k-group alignment approach definitely provides a much more robust and powerful tool compared to 2-group alignments.

5.4. Observations on computational results

Although the above-mentioned results are still preliminary ones, the following may be observed.

- We have tested two methods of dividing d sequences into k groups. It is observed that the k-restricted-grouping method RI(k) tends to produce better-score alignments than the k-random-grouping method RA(k) on the average.

Table 10
64 sequences of Elongation Factor

Absgl ABSIDIA GLAUCA (PIN MOULD) # P28295; FUNGI; ZYGOMYCOTINA 458AA.
Ajeca AJELLOMYCES CAPSULATA (HISTOPLASMA CAPSULATUM) # P40911; FUNGI; ASCOMYCOTINA; 460AA.
Apime APIS MELLIFERA (HONEYBEE) # P19039; METAZOA; ARTHROPODA; INSECTA 461AA
Arath ARABIDOPSIS THALIANA (MOUSEEAR CRESS) # P13905; PLANTA; EMBRYOPHYTA 449AA.
Artsa ARTEMIA SALINA (BRINE SHRIMP) # P02993; METAZOA; ARTHROPODA; CRUSTACEA 461AA
Arxad ARXULA ADENINIVORANS # P41745; FUNGI; DEUTEROMYCOTINA 459AA.
Ashgo ASHBYA GOSSYPII # P41752; FUNGI; 458AA.
Blaho BLASTOCYSTIS HOMINIS # P54959; 434AA.
Bommo BOMBYX MORI (SILK MOTH) # P29520; METAZOA; ARTHROPODA; INSECTA; 463AA.
Caeel CAENORHABDITIS ELEGANS # P53013; METAZOA; ACOELOMATES; NEMATODA; 463AA.
Canal CANDIDA ALBICANS (YEAST) # P16017; FUNGI; DEUTEROMYCOTINA 458AA.
Ratno RATTUS NORVEGICUS (RAT) # P20001; METAZOA; CHORDATA; VERTEBRATA; 462AA.
Desmo DESULFUROCOCCUS MOBILIS #P41203; ARCHAEBACTERIA; CRENARCHAEOTA; 456AA.
Enthi ENTAMOEBA HISTOLYTICA # P31018; 430AA.
Euggr EUGLENA GRACILIS # P14963; 445AA.
Giala GIARDIA LAMBLIA # Q08046; DIPLOMONADIDA; 396AA.
Helvi HELIOTHIS VIRESCENS (NOCTUID MOTH) # P55276; METAZOA; ARTHROPODA; INSECTA; 413AA.
Horvu HORDEUM VULGARE (BARLEY) # P34824; PLANTA; EMBRYOPHYTA; ANGIOSPERMAE; MONOCOTYLEDONEAE; 447AA.
Hydat HYDRA ATTENUATA (HYDRA) # P51554; METAZOA; CNIDARIA; 468AA.
Lyces LYCOPERSICON ESCULENTUM (TOMATO) # P17786; PLANTA; EMBRYOPHYTA; ANGIOSPERMAE; DICOTYLEDONEAE; 448AA.
Musmu MUS MUSCULUS (MOUSE) # P10126; 462AA.
Oncvo ONCHOCERCA VOLVULUS # P27592; METAZOA; ACOELOMATES; NEMATODA 464AA.
Plafa PLASMODIUM FALCIPARUM # Q00080; APICOMPLEXA; 443AA.
Pucgra PUCCINIA GRAMINIS # P32186; FUNGI; BASIDIOMYCOTINA; 463AA.
Pyrwo PYROCOCCUS WOESEI # P26751; ARCHAEBACTERIA; CRENARCHAEOTA; 430AA.
Rhyam RHYNCHOSCIARA AMERICANA (FUNGUS GNAT) # P27634; METAZOA; ARTHROPODA; INSECTA; 412AA.
Schpo SCHIZOSACCHAROMYCES POMBE # P50522; FUNGI; ASCOMYCOTINA; 461AA.
Glyma GLYCINE MAX (SOYBEAN) # P25698; PLANTA; EMBRYOPHYTA; ANGIOSPERMAE; DICOTYLEDONEAE; 447AA.
Style STYLONYCHIA LEMNAE # P25166; PROTOZOA; CILIOPHORA; CILIATA; SPIROTRICHA; HYPOTRICHIDA 446AA.
Sulac SULFOLOBUS ACIDOCALDARIUS # P17196; ARCHAEBACTERIA; CRENARCHAEOTA; 435AA.
Sulso SULFOLOBUS SOLFATARICUS # P35021; ARCHAEBACTERIA; CRENARCHAEOTA; 435AA.
Tetpy TETRAHYMENA PYRIFORMIS # Q04634; PROTOZOA; CILIOPHORA; CILIATA; 435AA.
Theac THERMOPLASMA ACIDOPHILUM # P19486; 424AA.
Thece THERMOCOCCUS CELER # P17197; ARCHAEBACTERIA; CRENARCHAEOTA; 428AA.
Nicta NICOTIANA TABACUM (TOBACCO) # P43643; PLANTA; EMBRYOPHYTA; ANGIOSPERMAE; DICOTYLEDONEAE; 447AA.
Trire TRICHODERMA REESEI # P34825; FUNGI; DEUTEROMYCOTINA 460AA.
Trybr TRYPANOSOMA BRUCEI BRUCEI # P41166; 449AA.
Triae TRITICUM AESTIVUM (WHEAT) # Q03033; PLANTA; EMBRYOPHYTA; ANGIOSPERMAE; MONOCOTYLEDONEAE; 447AA.
Sacce SACCHAROMYCES CEREVISIAE # P02994; FUNGI; ASCOMYCOTINA; 458AA.
Ana Anatoli (sponge) # 457AA.
Schma1 Schistosoma mansoni # Y08487; Metazoa; Platyhelminthes, Trematoda; 465AA.
Erego Eremothecium gossypii # X73978; Fungi; Ascomycota 458AA.
Rhira Rhizomucor racemosus # J02605; M16352; Fungi; Zygomycota; 458AA.
Neucr Neurospora crassa #Q01372 (D45837);Fungi; Ascomycota; 460AA.
Podcu Podospora curvicolla #Q01765 (X96614); Fungi; Ascomycota; 461AA.
Sorma Sordaria macrospora # Q09069 (X96615); Fungi; Ascomycota; 460AA.
Homsa Homo sapiens (human) # P04720 (J04617; J04616) 462AA.
Aneer Anemonia erythraea # Q16995 (D49922); Metazoa; Cnidaria; 466AA.
Borcu Beroe cucumis # Q17125 (D49923); Metazoa; Ctenophora; Beroida 415AA.
Dugja Dugesia japonica # Q23959 (D49924); Metazoa; Platyhelminthes; 407AA.
Drome Drosophila melanogaster # P08736 (X06869); Metazoa; Arthropoda; 463AA.
Drome2 Drosophila melanogaster # P05303 (X06870); Metazoa; Arthropoda; 462AA.
Ephfl Ephydatia fluviatilis # Q24784 (D49925); Metazoa; Porifera; 401AA.
Eugja Eugymnanthea japonica # Q24867 (D49926); Metazoa; Cnidaria; Hydrozoa 410AA.
Hydma Hydra magnipapillata # Q25099 (D79977); Metazoa; Cnidaria; Hydrozoa; 468AA.
Hydvu Hydra vulgaris # P51554 (Z68181); Metazoa; Cnidaria; Hydrozoa; 468AA.
Schma2 Schistosoma mansoni (blood fluke) # Y08487; Metazoa; Platyhelminthes; 465AA.
Orycu Oryctolagus cuniculus (rabbit) #P04720 (X62245); 462AA.
Dauca Daucus carota (carrot) # P29521 (X60302); 449AA.
Pissa Pisum sativum (pea) # Q41011 (X96555); 447AA.
Glupl Glugea plecoglossi # Q25002 (D84253); 465AA.
Brare Brachydanio rerio #Q92005 (L47669); Metazoa; Chordata; Vertebrata; Actinopterygii; 462AA.
Galga Gallus gallus #Q90835 (L00677); 462AA.
Xenla Xenopus laevis #P13549 (X55324); 462AA.

- As k becomes larger, the k-grouping takes more time, but produces on the average better alignment results. Although for $k = 4, 5$ it takes definitely larger time compared with 2- and 3-group alignments, by these results it is verified that 4- (and 5- with high similarity) group alignments can be practically used to polish up an obtained alignment further.

- The A* search greatly reduces the running time required by DP. In fact, in these experiments, 3-group alignments can be solved in time within a small constant factor of the running time of standard two-dimensional DP. Furthermore, their solution quality is definitely better than that of 2-group methods.
- Related to a widely known fact concerning k-opt local search algorithms for combinatorial problems, one should realize that there are so many local optima in such combinatorial problems, and there do exist many local optima for 2-group alignments which are not local optima for 3- and higher order group alignments. By increasing k, one might think it takes more time to check the local optimality of current solution which makes this approach less practical, but this is not a problem at all since 3- and higher order group alignments can produce better solution than 2-grouping method within reasonable additional time and checking the local optimality is less important in this respect.

Thus, this paper proposed the use of k-group alignment for $k \geqslant 3$, and showed its power through computational experiments. There are still many points which can be improved further in the current code, and developing a refined system would be very interesting as future work.

6. Concluding remarks

The results of this paper has further been extended by Shibuya and Imai [15, 17, 16] to enumerate meaningful suboptimal alignments and many types of parametric analysis of multiple alignments. It may be said that the framework presented here is flexible enough to obtain such extensions in a natural manner.

Acknowledgements

The work of the second author was supported in part by the Grant-in-Aid for Priority Areas 'Genome Science' of the Ministry of Education, Science, Sports and Culture of Japan.

References

[1] S. Araki, M. Goshima, S. Mori, H. Nakashima, S. Tomita, Y. Akiyama, and M. Kanehisa, Application of parallelized DP and A* algorithm to multiple sequence alignment, Proc. Genome Informatics Workshop IV, 1993, pp. 94–102.

[2] M.P. Berger, P.J. Munson, A novel randomized iterative strategy for aligning multiple protein sequences, Comput. Appl. Biosci. 7 (1991) 479–484.

[3] H. Carrillo, D.J. Lipman, The multiple sequence alignment problem in biology, SIAM J. Appl. Math. 48 (1988) 1073-1082.

[4] M.O. Dayhoff, R.M. Schwartz, B.C. Orcutt, Atlas of Protein Sequence and Structure, National Biomedical Research Foundation, Washington, DC, 1978.

[5] O. Gotoh, Optimal alignment between groups of sequences and its application to multiple sequence alignment, Comput. Appl. Biosci. 9 (1993) 361–370.
[6] S.K. Gupta, J.D. Kececioglu, A.A. Schaffer, Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment, J. Comput. Biol. 2 (3) (1995) 459–472.
[7] P.E. Hart, N.J. Nillson, B. Rafael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Systems Sci. and Cybernet SSC-4 (1968) 100–107.
[8] M. Hirosawa, M. Hoshida, M. Ishikawa, T. Toya, MASCOT: multiple alignment system for protein sequence based on three-way dynamic programming, Comput. Appl. Biosci. 9 (1993) 161–167.
[9] M. Hirosawa, Y. Totoki, M. Hoshida, M. Ishikawa, Comprehensive study on iterative algorithms of multiple sequence alignment, Comput. Appl. Biosci. 11 (1) (1995) 13–18.
[10] T. Ikeda, Applications of the A* algorithm to better routes finding and multiple sequence alignment, Master's Thesis, Department of Information Science, University of Tokyo, 1995. Available at http://naomi.is.s.u-tokyo.ac.jp/.
[11] T. Ikeda, H. Imai: Fast A* Algorithms for multiple sequence alignment, Proc. Genome Informatics Workshop, 1994, pp. 90–99.
[12] T. Ikeda, M. Hsu, H. Imai, S. Nishimura, H. Shimoura, T. Hashimoto, K. Temmoku, K. Mitoh, A fast algorithm for finding better routes by AI search techniques, Proc. Internat. Conf. on Vehicle Navigation & Information Systems (VNIS'94), 1994, pp. 291–296.
[13] H. Imai, T. Ikeda, *k*-Group multiple alignment based on A* search, Proc. Genome Informatics Workshop, Yokohama, 1995, pp. 9–18.
[14] G. Shibayama, H. Imai, Finding *K*-best alignment of multiple sequences, Proc. Genome Informatics Workshop IV, 1993, pp. 120–129.
[15] T. Shibuya, H. Imai, Enumerating suboptimal alignments of multiple biological sequences efficiently, Proc. Pacific Symp. on Biocomputing, 1997, pp. 409–420.
[16] T. Shibuya, H. Imai, Parametric alignment of multiple biological sequences, Proc. Genome Informatics Workshop, 1996, pp. 41–50.
[17] T. Shibuya, H. Imai, New Flexible approaches for multiple sequence alignment, J. Comput. Biol. 4 (3) (1997) pp. 385–413; a preliminary version was presented at Proc. 1st ACM Ann. Internat. Conf. on Computational Molecular Biology, 1997, pp. 267–276.
[18] J.L. Spouge, Speeding up dynamic programming algorithms for finding optimal lattice paths, SIAM J. Appl. Math. 49 (1989) 1552–1566.
[19] M.S. Waterman, Introduction to Computational Biology: Maps, Sequences and Genomes, Chapman & Hall, London, 1995.