# Improvement of the A* Algorithm for Multiple Sequence Alignment

**Hirotada Kobayashi** hirotada@is.s.u-tokyo.ac.jp

**Hiroshi Imai** imai@is.s.u-tokyo.ac.jp

Department of Information Science, Faculty of Science, University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan

**Abstract**

The alignment problem of DNA or protein sequences is very applicable and important in various fields of molecular biology. This problem can be reduced to the shortest path problem and Ikeda and Imai [4] showed that the A* algorithm works efficiently with the estimator utilizing all 2-dimensional sub-alignments.

In this paper we present new powerful estimators utilizing $k \geq 3$ dimensional sub-alignments, and propose a new bounding technique using $V_\Delta$, a set of vertices in the paths whose lengths are at most $\Delta$ longer than the shortest path. We also extend our algorithm to a recursive-estimate version. These algorithms become more efficient when the number of sequences increase, or the similarity among sequences is lower.

## 1 Introduction

Multiple sequence alignment is a problem to find the alignment of multiple sequences with the highest score due to a given scoring criterion between characters. It is used for extracting biologically important commonalities from a set of DNA or protein sequences, and is applicable to various important fields such as the prediction of three dimensional structures of proteins or the inference of phylogenetic tree in molecular biology.

This problem can be solved by finding the shortest path on some directed acyclic mesh-shaped graph, and the famous approach to this problem is Dynamic Programming (DP) . However, both time and space complexity of the DP approach are $O(l^n)$ when the number of sequences is $n$ and the length of each sequence is $O(l)$, thus it is effective only for the cases of small dimension, two or three, and becomes impractical for higher dimensional case, say the dimension of five or six.

The A* algorithm is a well-known heuristic search method in Artificial Intelligence, and finds the shortest path quite more efficiently than DP or the Dijkstra method by utilizing the heuristic estimate for the shortest path length. Spouge [8] introduced the concept of the A* algorithm to the multiple sequence alignment problem in order to bound the search space of DP. Ikeda and Imai [4] suggested using the A* algorithm directly to the multiple sequence alignment problem, and showed that this direct approach works very well with the estimator utilizing all 2-dimensional sub-alignments.

This paper further improves Ikeda and Imai's results by introducing new powerful estimators utilizing high $(k \geq 3)$ dimensional sub-alignments. In order to use the estimators utilizing high $(k \geq 3)$ dimensional sub-alignments, more efficient search-space-bounding techniques are required. Therefore we also proposes a new bounding technique using $V_\Delta$, a set of vertices in the paths whose lengths are at most $\Delta$ longer than the shortest path. We also extend our algorithm to a recursive-estimate version. These algorithms become more efficient when the number of sequences increases, or the similarity among sequences is lower.

# 2 Multiple Sequence Alignment and the Shortest Path Problem

## 2.1 Multiple Sequence Alignment Problem

Here we formally define the multiple sequence alignment problem.

Given a finite alphabet set $\Sigma$ and a family $\mathcal{S} = (\boldsymbol{s}_1, \cdots, \boldsymbol{s}_n)$ of $n$ sequences:

$$\boldsymbol{s}_i = s_{i1}s_{i2}\cdots s_{il_i} \ (1 \le i \le n)$$

where each sequence entry $s_{ij} \in \Sigma$, an alignment of $\mathcal{S}$ is a matrix $A = (a_{ij})_{1\le i\le n, 1\le j\le l}$ where

(i) $a_{ij} \in \Sigma \cup \{-\}$, with "$-$" denoting the *gap* letters,

(ii) each row $\boldsymbol{a}_i = a_{i1}\cdots a_{il}(1 \le i \le n)$ of $A$ is exactly the corresponding sequence $\boldsymbol{s}_i$ if we eliminate all gap letters,

(iii) $A$ has no column which only contains gaps.

We are also given a score function

$$\phi : (\Sigma \cup \{-\})^2 \to \mathbf{R}$$

defined according to *matching scores* of each two characters in $\Sigma$ and a gap penalty (Note that $\phi(-,-) = 0$). For each pair of rows $\boldsymbol{a}_u, \boldsymbol{a}_v$ in $A$, define the *pairs score*

$$p(\boldsymbol{a}_u, \boldsymbol{a}_v) = \sum_{j=1}^{l} \phi(a_{uj}, a_{vj}).$$

Then the *sum of pairs score* for an alignment $A$ is defined by

$$P(A) = \sum_{1\le u<v\le n} p(\boldsymbol{a}_u, \boldsymbol{a}_v),$$

and the multiple sequence alignment problem is a problem to find the alignment $A_{\mathrm{opt}}$ which maximizes the sum of pairs score.

In this definition of the problem, the gap penalty is linear to sum of the length of gap sequences. This kind of gap penalties are called the *linear gap penalties.* If there is some starting gap penalty which is charged for the starting of each gap sequence, it is called the *affine gap penalty.* In this paper, we deal with the linear gap penalty.

## 2.2 Reduction to the Shortest Path Problem

Multiple sequence alignment problem can be solved by finding the shortest path on some directed acyclic mesh-shaped graph. Let us consider a directed acyclic graph $G = (V, E)$ such that

$$\begin{aligned} V &= \{(x_1, \cdots x_n) \mid x_i = 0, 1, \cdots, l_i\} \\ E &= \bigcup_{e\in\{0,1\}^n} \{(v, v+e) \mid v, v+e \in V,\ e \ne 0\}. \end{aligned}$$

A path from the vertex $s = (0, \cdots, 0)$ to the vertex $t = (l_1, \cdots, l_n)$ corresponds to an alignment of sequences.

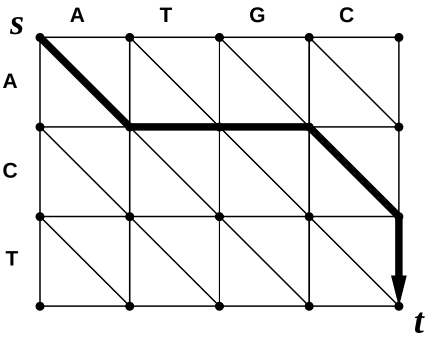

Figure 1: The graph for the alignment of two sequences ATGC and ACT.

For instance, in two dimensional case, the graph $G$ is constructed as Fig. 1. Each row or column in this graph corresponds to each character of first or second sequences, respectively. Each diagonal edge represents a match between corresponding two characters and has its length of corresponding *matching cost*, while each horizontal or vertical edge represents an insertion of a gap and the gap penalty is assigned to its length. When the matching scores represent the similarity between characters, the *matching costs* are obtained by reversing signs of the matching scores.

For more than two dimensional case, a score of an alignment is a sum of all pairs scores. Therefore each edge length in $G$ is defined as a sum of all corresponding edge lengths in the graphs for pairwise alignments. Let $G^{(2)}_{p_1,p_2} = (V^{(2)}_{p_1,p_2}, E^{(2)}_{p_1,p_2})$ denote the graph for the pairwise alignment of $\boldsymbol{s}_{p_1}$, $\boldsymbol{s}_{p_2}$, and $v^{(2)}_{p_1,p_2} \in V^{(2)}_{p_1,p_2}$ denote the corresponding vertex to $v \in V$. Then the length of edge $(u,v) \in E$ is defined as

$$l(u,v) = \sum_{1 \le p_1 < p_2 \le n} l(u^{(2)}_{p_1,p_2}, v^{(2)}_{p_1,p_2}),$$

where $l(u^{(2)}_{p_1,p_2}, v^{(2)}_{p_1,p_2})$ denotes the length of the edge $(u^{(2)}_{p_1,p_2}, v^{(2)}_{p_1,p_2})$ in the graph $G^{(2)}_{p_1,p_2}$ (if $u^{(2)}_{p_1,p_2} = v^{(2)}_{p_1,p_2}$, $l(u^{(2)}_{p_1,p_2}, v^{(2)}_{p_1,p_2}) = 0$). Therefore finding the optimal alignment is equivalent to finding the shortest path from the vertex $s = (0, \cdots, 0)$ to the vertex $t = (l_1, \cdots, l_n)$ in the graph $G$.

# 3 The A* Algorithm and Multiple Sequence Alignment Problem

## 3.1 The A* Algorithm

The A algorithm finds the shortest path from $s$ to $t$ efficiently when all edge lengths are non-negative. It utilizes a heuristic estimate $\hat{h}(v)$ for the shortest path length from each vertex $v \in V$ to $t$. Let $h(v)$ denote the actual shortest path length from $v$ to $t$. If the estimate $\hat{h}(v)$ suffices the condition $\hat{h}(v) \le h(v)$ for all vertices $v \in V$, the algorithm exactly returns the optimal shortest path from $s$ to $t$. In such a case we call the algorithm the A* algorithm. An outline of the A* algorithm is described as follows [3]:

**Algorithm A***

1. *Let $\hat{g}(v)$ denote a temporary estimate for the shortest path length from $s$ to each vertex $v$, and initially $\hat{g}(v) := +\infty$ (for all $v \neq s$), $\hat{g}(s) := 0$.*
   *Let $W$ be a vertex set which is initially $\emptyset$.*

2. *Find a vertex $v \in V \backslash W$ which has the minimum value of $\hat{f}(v) = \hat{g}(v) + \hat{h}(v)$.*
   *Add the vertex $v$ to $W$. If $v = t$, the algorithm stops.*

3. *Expand the vertex $v$. That is, for all the vertices $v'$ that $(v, v') \in E$, if $\hat{g}(v) + l(v, v') < \hat{g}(v')$, renew the value $\hat{g}(v') := \hat{g}(v) + l(v, v')$ and replace the path from $s$ to $v'$ with the shortest path from $s$ to $v$ added with the edge $(v, v')$. If $v \in W$, remove $v$ from $W$.*

4. *Go to step 2.*

In the A* algorithm, unlike Dijkstra method, the shortest path from $s$ may not appear first for each expanded vertex $v$, and a shorter path from $s$ to $v$ may be found in the future search. It makes the algorithm rather inefficient. This disadvantage can be avoided if we use the estimator $\hat{h}$ with appropriate property.

**Definition 1** *The estimator $\hat{h}$ is called dual feasible if and only if $\hat{h}$ satisfies the following constraint:*

$$\forall (u, v) \in E \quad l(u, v) + \hat{h}(v) \geq \hat{h}(u). \tag{1}$$

If the estimator $\hat{h}$ is dual feasible, the A* algorithm never expands the same vertices twice, and searches the shortest path efficiently.

In the A* algorithm, an upper bound $\hat{x}$ for the actual shortest path length $x$ from $s$ to $t$ can be utilized. We can ignore such vertices $v$ in the search that $\hat{f}(v) = \hat{g}(v) + \hat{h}(v) \geq \hat{x}$.

Finally we explain how to modify the length of edges in $G$ in order to exclude edges of negative length. The following theorem is useful [4].

**Theorem 2** *Let $\hat{h}$ be a dual feasible estimator. The Dijkstra method on a graph in which $l(u, v)$ is replaced by $l'(u, v)$ as follows is equivalent to the A* algorithm on the original graph:*

$$l'(u, v) = l(u, v) + \hat{h}(v) - \hat{h}(u).$$

In this modification, each new edge length is non-negative if the estimator $\hat{h}$ is dual feasible. Therefore with the dual feasible estimator, the Dijkstra method on the modified graph works well even if the original graph has edges of negative length.

### 3.2 The A* Algorithm and Multiple Sequence Alignment Problem

In the following part of this paper, we assume that $n \geq 3$ where $n$ is the number of sequences in the alignment problem.

Let us consider the alignment problem of $n$ sequences $\boldsymbol{s}_1, \cdots, \boldsymbol{s}_n$. Let $G = (V, E)$ be the corresponding $n$-dimensional mesh-shaped graph.

For every $k$ sequences $\boldsymbol{s}_{p_1}, \cdots, \boldsymbol{s}_{p_k}$ chosen from $\boldsymbol{s}_1, \cdots, \boldsymbol{s}_n$, we can consider the corresponding $k$-dimensional mesh-shaped subgraph $G^{(k)}_{p_1,\cdots,p_k}$. Let $L^{(k)}_{p_1,\cdots,p_k}(v)$ denote the shortest path length from $v^{(k)}_{p_1,\cdots,p_k}$ to $t^{(k)}_{p_1,\cdots,p_k}$ in $G^{(k)}_{p_1,\cdots,p_k}$, where $v^{(k)}_{p_1,\cdots,p_k}, t^{(k)}_{p_1,\cdots,p_k} \in G^{(k)}_{p_1,\cdots,p_k}$ are the corresponding vertices to $v, t \in G$, respectively.

Ikeda and Imai [4] showed that the following estimator is very useful for the alignment problem.

**Theorem 3 (Ikeda and Imai)** *The estimator $\hat{h}_{\text{pair}}$ defined as follows is dual feasible:*

$$\hat{h}_{\text{pair}}(v) = \sum_{1 \leq p_1 < p_2 \leq n} L^{(2)}_{p_1,p_2}(v).$$

**Proof:** For any edge $(u, v) \in E$,

$$l(u, v) + \hat{h}_{\text{pair}}(v) = \sum_{1 \leq p_1 < p_2 \leq n} \left( l(u^{(2)}_{p_1,p_2}, v^{(2)}_{p_1,p_2}) + L^{(2)}_{p_1,p_2}(v) \right) \geq \sum_{1 \leq p_1 < p_2 \leq n} L^{(2)}_{p_1,p_2}(u) = \hat{h}_{\text{pair}}(u).$$

□

Ikeda and Imai [4] also noted that, with the estimator $\hat{h}_{\text{pair}}$, the A* algorithm utilizing an upper bound for the shortest path length from $s$ to $t$ is equivalent to using the branch-and-bound techniques proposed by Spouge [8] or the return-cost pruning implemented in `MSA` program by Gupta et al. [2].

# 4 New Estimators Utilizing High Dimensional Sub-alignments

In another point of view, the estimator $\hat{h}$ in the A$^*$ algorithm gives a lower bound for the actual path length $h$. The tighter bound the estimator gives, the more efficiently the algorithm searches the shortest path.

The estimator $h_{\text{pair}}$ only utilizes the information of every two dimensional sub-alignment. If we use the information of $k \geq 3$ dimensional sub-alignments, we can obtain more powerful estimators.

First, we consider the natural extension of $h_{\text{pair}}$ by utilizing the information of all $k$-dimensional sub-alignments.

**Theorem 4** *The estimator $\hat{h}_{\text{all-}k}$ defined as follows is dual feasible:*

$$\hat{h}_{\text{all-}k}(v) = \frac{1}{\binom{n-2}{k-2}} \sum_{1 \leq p_1 < \cdots < p_k \leq n} L^{(k)}_{p_1,\cdots,p_k}(v).$$

**Proof:** For any edge $(u,v) \in E$,

$$\begin{aligned}
l(u,v) + \hat{h}_{\text{all-}k}(v) &= \frac{1}{\binom{n-2}{k-2}} \left( \binom{n-2}{k-2} \cdot \sum_{1 \leq i < j \leq n} l(u^{(2)}_{i,j}, v^{(2)}_{i,j}) + \sum_{1 \leq p_1 < \cdots < p_k \leq n} L^{(k)}_{p_1,\cdots,p_k}(v) \right) \\
&= \frac{1}{\binom{n-2}{k-2}} \sum_{1 \leq p_1 < \cdots < p_k \leq n} \left( \sum_{i,j \in p_1,\cdots,p_k, i<j} l(u^{(2)}_{i,j}, v^{(2)}_{i,j}) + L^{(k)}_{p_1,\cdots,p_k}(v) \right) \\
&\geq \frac{1}{\binom{n-2}{k-2}} \sum_{1 \leq p_1 < \cdots < p_k \leq n} L^{(k)}_{p_1,\cdots,p_k}(u) = \hat{h}_{\text{all-}k}(u).
\end{aligned}$$

□

The estimator $\hat{h}_{\text{all-}k}$ satisfies the following property.

**Proposition 5** *$\hat{h}_{\text{pair}}(v) \leq \hat{h}_{\text{all-}k}(v) \leq h(v)$ for all $v \in V$.*

**Proof:** For any vertex $v \in V$, $L^{(k)}_{p_1,\cdots,p_k}(v)$ satisfies that

$$L^{(k)}_{p_1,\cdots,p_k}(v) \geq \sum_{q_1,q_2 \in p_1,\cdots,p_k} L^{(2)}_{q_1,q_2}(v).$$

If we take every $p_1, \cdots, p_k$ in the region of $1 \leq p_1 < \cdots < p_k \leq n$, each $L^{(2)}_{q_1,q_2}(v)$ for fixed $q_1, q_2$ on the rightside of the inequality appears $\binom{n-2}{k-2}$ times. Thus we obtain the following inequality:

$$\hat{h}_{\text{all-}k}(v) = \frac{1}{\binom{n-2}{k-2}} \sum_{1 \leq p_1 < \cdots < p_k \leq n} L^{(k)}_{p_1,\cdots,p_k}(v) \geq \sum_{1 \leq p_1 < p_2 \leq n} L^{(2)}_{p_1,p_2}(v) = \hat{h}_{\text{pair}}(v).$$

On the other hand, for the shortest path from $v$ to $t$ in the $n$-dimensional graph $G$, we can consider the projection of this to each $k$-dimensional subgraph $G^{(k)}_{p_1,\cdots,p_k}$. Let $L'^{(k)}_{p_1,\cdots,p_k}(v)$ denote this projection path length from $v^{(k)}_{p_1,\cdots,p_k}$ to $t^{(k)}_{p_1,\cdots,p_k}$ in $G^{(k)}_{p_1,\cdots,p_k}$. Then it is obvious that $L'^{(k)}_{p_1,\cdots,p_k}(v) \geq L^{(k)}_{p_1,\cdots,p_k}(v)$. If we remark that $L'^{(k)}_{p_1,\cdots,p_k}(v) = \sum_{q_1,q_2 \in p_1,\cdots,p_k} L'^{(2)}_{q_1,q_2}(v)$, we can prove that

$$\begin{aligned}
\hat{h}_{\text{all-}k}(v) &= \frac{1}{\binom{n-2}{k-2}} \sum_{1 \leq p_1 < \cdots < p_k \leq n} L^{(k)}_{p_1,\cdots,p_k}(v) \\
&\leq \frac{1}{\binom{n-2}{k-2}} \sum_{1 \leq p_1 < \cdots < p_k \leq n} L'^{(k)}_{p_1,\cdots,p_k}(v) = \sum_{1 \leq p_1 < p_2 \leq n} L'^{(2)}_{p_1,p_2}(v) = h(v)
\end{aligned}$$

by a similar argument above.

Thus we conclude that $\hat{h}_{\text{pair}}(v) \leq \hat{h}_{\text{all-}k}(v) \leq h(v)$ for all $v \in V$, which means that the estimator $\hat{h}_{\text{all-}k}$ gives tighter bound for the actual path length $h$ than $\hat{h}_{\text{pair}}$. □

However, as we shall see later in our experiments, it is quite difficult to compute this estimator efficiently. Therefore instead of utilizing all $k$-dimensional sub-alignments, we use the information of *one* $k$-dimensional sub-alignment and the rest $(n-k)$-dimensional sub-alignment.

**Theorem 6** *Let $\boldsymbol{s}_{p_1}, \cdots, \boldsymbol{s}_{p_k}$ be arbitrary chosen $k$ sequences, and let $\boldsymbol{s}_{p_{k+1}}, \cdots, \boldsymbol{s}_{p_n}$ be the rest $n-k$ sequences. Then the following estimator $\hat{h}_{\text{one-}k}$ is dual feasible:*

$$\hat{h}_{\text{one-}k}(v) = L^{(k)}_{p_1,\cdots,p_k}(v) + L^{(n-k)}_{p_{k+1},\cdots,p_n}(v) + \sum_{i=1}^{k}\sum_{j=k+1}^{n} L^{(2)}_{p_i,p_j}(v).$$

**Proof:** For any edge $(u,v) \in E$,

$$\begin{aligned}
l(u,v) + \hat{h}_{\text{one-}k}(v) &= \sum_{1 \leq i<j \leq n} l(u^{(2)}_{i,j}, v^{(2)}_{i,j}) + L^{(k)}_{p_1,\cdots,p_k}(v) + L^{(n-k)}_{p_{k+1},\cdots,p_n}(v) + \sum_{i=1}^{k}\sum_{j=k+1}^{n} L^{(2)}_{p_i,p_j}(v) \\
&= \left( \sum_{i,j \in p_1,\cdots,p_k, i<j} l(u^{(2)}_{i,j}, v^{(2)}_{i,j}) + L^{(k)}_{p_1,\cdots,p_k}(v) \right) \\
&\quad + \left( \sum_{i,j \in p_{k+1},\cdots,p_n, i<j} l(u^{(2)}_{i,j}, v^{(2)}_{i,j}) + L^{(n-k)}_{p_{k+1},\cdots,p_n}(v) \right) + \sum_{i=1}^{k}\sum_{j=k+1}^{n} \left( l(u^{(2)}_{p_i,p_j}, v^{(2)}_{p_i,p_j}) + L^{(2)}_{p_i,p_j}(v) \right) \\
&\geq L^{(k)}_{p_1,\cdots,p_k}(u) + L^{(n-k)}_{p_{k+1},\cdots,p_n}(u) + \sum_{i=1}^{k}\sum_{j=k+1}^{n} L^{(2)}_{p_i,p_j}(u) = \hat{h}_{\text{one-}k}(u).
\end{aligned}$$

□

The estimator $\hat{h}_{\text{one-}k}$ also gives tighter bound for $h$ than $\hat{h}_{\text{pair}}$.

**Proposition 7** $\hat{h}_{\text{pair}}(v) \leq \hat{h}_{\text{one-}k}(v) \leq h(v)$ *for all $v \in V$.*

**Proof:** With the same argument as the proof of Proposition 5, $L^{(k)}_{p_1,\cdots,p_k}(v)$ and $L^{(n-k)}_{p_{k+1},\cdots,p_n}(v)$ satisfy that

$$\begin{aligned}
L^{(k)}_{p_1,\cdots,p_k}(v) &\geq \sum_{q_1,q_2 \in p_1,\cdots,p_k} L^{(2)}_{q_1,q_2}(v) \\
L^{(n-k)}_{p_{k+1},\cdots,p_n}(v) &\geq \sum_{q_1,q_2 \in p_{k+1},\cdots,p_n} L^{(2)}_{q_1,q_2}(v)
\end{aligned}$$

for any vertex $v \in V$. Thus $\hat{h}_{\text{one-}k}(v) \geq \hat{h}_{\text{pair}}(v)$ is satisfied.

We can also prove that

$$\begin{aligned}
\hat{h}_{\text{one-}k}(v) &= L^{(k)}_{p_1,\cdots,p_k}(v) + L^{(n-k)}_{p_{k+1},\cdots,p_n}(v) + \sum_{i=1}^{k}\sum_{j=k+1}^{n} L^{(2)}_{p_i,p_j}(v) \\
&\leq L'^{(k)}_{p_1,\cdots,p_k}(v) + L'^{(n-k)}_{p_{k+1},\cdots,p_n}(v) + \sum_{i=1}^{k}\sum_{j=k+1}^{n} L'^{(2)}_{p_i,p_j}(v) = \sum_{1 \leq q_1 < q_2 \leq n} L'^{(2)}_{q_1,q_2}(v) = h(v).
\end{aligned}$$

Thus we conclude that $\hat{h}_{\text{pair}}(v) \leq \hat{h}_{\text{one-}k}(v) \leq h(v)$ for any $v \in V$. □

Note that, for fixed $k$, $\hat{h}_{\text{all-}k}$ usually gives the tightest lower bound for $h$ among these three estimators.

From Proposition 5 or Proposition 7 we can see that, with the estimator $\hat{h}_{\text{all-}k}$ or $\hat{h}_{\text{one-}k}$, the A* algorithm utilizing the upper bound always bounds the search space tighter than the branch-and-bound techniques by Spouge [8] or the return-cost pruning by Gupta et al. [2].

## 5 A New Bounding Technique Using $V_\Delta$

In the previous section we have presented two new estimators which utilize the information of $k \geq 3$ dimensional sub-alignments. When we compute a value of estimator $\hat{h}(v)$ which uses the information of $k$-dimensional sub-alignments, we must know the shortest path length from $v^{(k)}_{p_1,\cdots,p_k}$ to $t^{(k)}_{p_1,\cdots,p_k}$ in $G^{(k)}_{p_1,\cdots,p_k}$ for the corresponding vertex $v^{(k)}_{p_1,\cdots,p_k}$ to $v$. However, even if $k=3$, each subgraph $G^{(3)}_{p_1,p_2,p_3}$ contains about $10^8$ vertices when the length of each sequence $l_i \simeq 500 (1 \leq i \leq n)$. Therefore computing the shortest path lengths to $t^{(k)}_{p_1,\cdots,p_k}$ for all the vertices $v^{(k)}_{p_1,\cdots,p_k} \in V^{(k)}_{p_1,\cdots,p_k}$ is quite inefficient. Thus we must consider computing the shortest path lengths in $G^{(k)}_{p_1,\cdots,p_k}$ only that are truly required for the search of the shortest path in the original graph $G$.

Let $x^{(k)}_{p_1,\cdots,p_k}$ denote the shortest path length from $s^{(k)}_{p_1,\cdots,p_k}$ to $t^{(k)}_{p_1,\cdots,p_k}$ in $G^{(k)}_{p_1,\cdots,p_k}$. Then $V^{(k)}_{p_1,\cdots,p_k}(\Delta)$ is defined as the set of vertices $v^{(k)}_{p_1,\cdots,p_k} \in V^{(k)}_{p_1,\cdots,p_k}$ such that there exists at least one path from $s^{(k)}_{p_1,\cdots,p_k}$ to $t^{(k)}_{p_1,\cdots,p_k}$ via $v^{(k)}_{p_1,\cdots,p_k}$ whose length is at most $x^{(k)}_{p_1,\cdots,p_k} + \Delta$. This is the same concept that Shibuya and Imai [6] utilized for finding sub-optimal alignments. The following two theorems hold.

**Theorem 8** *Let $\hat{x}$ denote some existing upper bound for the shortest path length from $s$ to $t$ in $G$. Then we only need to compute the shortest path lengths to $t^{(k)}_{p_1,\cdots,p_k}$ in $G^{(k)}_{p_1,\cdots,p_k}$ for the vertices $v^{(k)}_{p_1,\cdots,p_k} \in V^{(k)}_{p_1,\cdots,p_k}(\Delta)$ for $\Delta$ given by the following constraint:*

$$\Delta = \binom{n-2}{k-2} \cdot \hat{x} - \sum_{1 \leq p_1 < \cdots < p_k \leq n} x^{(k)}_{p_1,\cdots,p_k}.$$

**Proof:** For any vertex $v'^{(k)}_{p_1,\cdots,p_k} \notin V^{(k)}_{p_1,\cdots,p_k}(\Delta)$ for above $\Delta$, we show that the shortest path length $x'$ from $s$ to $t$ via $v' \in V$, which corresponds to $v'^{(k)}_{p_1,\cdots,p_k}$, is always longer than $\hat{x}$.

$$x' > \frac{1}{\binom{n-2}{k-2}} \left( \sum_{1 \leq p_1 < \cdots < p_k \leq n} x^{(k)}_{p_1,\cdots,p_k} + \Delta \right) = \frac{1}{\binom{n-2}{k-2}} \cdot \binom{n-2}{k-2} \cdot \hat{x} = \hat{x}.$$

□

**Theorem 9** *Let $\hat{x}$ denote some existing upper bound for the shortest path length from $s$ to $t$ in $G$. Then we only need to compute the lengths of the shortest path to $t^{(k)}_{p_1,\cdots,p_k}$ in $G^{(k)}_{p_1,\cdots,p_k}$ (or to $t^{(n-k)}_{p_{k+1},\cdots,p_n}$ in $G^{(n-k)}_{p_{k+1},\cdots,p_n}$) for the vertices $v^{(k)}_{p_1,\cdots,p_k} \in V^{(k)}_{p_1,\cdots,p_k}(\Delta)$ (or $v^{(n-k)}_{p_{k+1},\cdots,p_n} \in V^{(n-k)}_{p_{k+1},\cdots,p_n}(\Delta)$, respectively) for $\Delta$ given by the following constraint:*

$$\Delta = \hat{x} - \left( x^{(k)}_{p_1,\cdots,p_k} + x^{(n-k)}_{p_{k+1},\cdots,p_n} + \sum_{i=1}^{k} \sum_{j=k+1}^{n} x^{(2)}_{p_i,p_j} \right).$$

**Proof:** Similar to the proof of Theorem 8. For any vertex $v'^{(k)}_{p_1,\cdots,p_k} \notin V^{(k)}_{p_1,\cdots,p_k}(\Delta)$ (or $v'^{(n-k)}_{p_{k+1},\cdots,p_n} \notin V^{(n-k)}_{p_{k+1},\cdots,p_n}(\Delta)$) for above $\Delta$, we show that the shortest path length $x'$ from $s$ to $t$ via $v' \in V$, which corresponds to $v'^{(k)}_{p_1,\cdots,p_k}$ (or $v'^{(n-k)}_{p_{k+1},\cdots,p_n}$, respectively), is always longer than $\hat{x}$.

$$x' > x^{(k)}_{p_1,\cdots,p_k} + x^{(n-k)}_{p_{k+1},\cdots,p_n} + \sum_{i=1}^{k} \sum_{j=k+1}^{n} x^{(2)}_{p_i,p_j} + \Delta = \hat{x}.$$

□

Note that any vertex $v \in V$ whose corresponding vertex $v^{(k)}_{p_1,\cdots,p_k}(\Delta) \notin V^{(k)}_{p_1,\cdots,p_k}(\Delta)$ for some $k$ and $p_1, \cdots, p_k$ can be ignored in the search of the shortest path $s$ to $t$ in $G$, which is regarded as the extension of Carrillo and Lipman's bounding techniques [1] to $k \geq 3$ dimensional sub-alignments.

Finally we must consider the way of computing $V^{(k)}_{p_1,\cdots,p_k}(\Delta)$ for $G^{(k)}_{p_1,\cdots,p_k}$. Fortunately it is rather easy for the A* algorithm to compute the exact $V^{(k)}_{p_1,\cdots,p_k}(\Delta)$ efficiently.

For simplicity, we use $G, V, V_\Delta, s, t, v, x$ instead of $G^{(k)}_{p_1,\cdots,p_k}$, $V^{(k)}_{p_1,\cdots,p_k}$, $V^{(k)}_{p_1,\cdots,p_k}(\Delta)$, $s^{(k)}_{p_1,\cdots,p_k}$, $t^{(k)}_{p_1,\cdots,p_k}$, $v^{(k)}_{p_1,\cdots,p_k}$, $x^{(k)}_{p_1,\cdots,p_k}$, respectively. The algorithm to compute the shortest path lengths $h(v)$ from $v$ to $t$ for all the vertices $v \in V_\Delta$ for $G$ goes as follows:

**Algorithm $V_\Delta$**

1. *Compute the shortest path length $x$ by the A* algorithm with the estimator $\hat{h} = \hat{h}_{\text{pair}}$ in the direction from $s$ to $t$.*

2. *Continue the search by expanding vertices as far as $\hat{f}(v) = \hat{g}(v) + \hat{h}(v) \leq x + \Delta$ holds. (Note that the actual shortest path length $g(v)$ from $s$ to $v$ has been computed for any expanded vertex $v$ in the A* algorithm with the dual feasible estimator.)*

3. *Obtain the actual shortest path length $h(v) = g_{t\to s}(v)$ from $v$ to $t$ using the A* algorithm in the direction from $t$ to $s$ with the estimator $\hat{h}_{t\to s}(v) = g(v)$ computed in step 2., by continuing the search as far as $\hat{f}_{t\to s}(v) = \hat{g}_{t\to s}(v) + \hat{h}_{t\to s}(v) \leq x + \Delta$ holds.*

## 6 Algorithms

The whole algorithm using the A* algorithm with the estimator $\hat{h}_{\text{all-}k}$ goes as follows:

**Algorithm 1**

1. *For each $i, j (1 \leq i < j \leq n)$, apply DP to the graph $G^{(2)}_{i,j}$ from $t^{(2)}_{i,j}$ and compute the shortest path length from $v^{(2)}_{i,j}$ to $t^{(2)}_{i,j}$ for every vertex $v^{(2)}_{i,j} \in V^{(2)}_{i,j}$.*

2. *Compute $V^{(k)}_{p_1,\cdots,p_k}(\Delta)$ for all $p_1, \cdots, p_k, 1 \leq p_1 < \cdots < p_k \leq n$ by Algorithm $V_\Delta$.*

3. *Apply the A* algorithm with the estimator $\hat{h}_{\text{all-}k}$ to the graph $G$ in the direction from $s$ to $t$.*

Similarly, the whole algorithm using the A* algorithm with the estimator $\hat{h}_{\text{one-}k}$ goes as follows:

**Algorithm 2**

1. *For each $i, j (1 \leq i < j \leq n)$, apply DP to the graph $G^{(2)}_{i,j}$ from $t^{(2)}_{i,j}$ and compute the shortest path length from $v^{(2)}_{i,j}$ to $t^{(2)}_{i,j}$ for every vertex $v^{(2)}_{i,j} \in V^{(2)}_{i,j}$.*

2. *Choose $k$ sequences $\boldsymbol{s}_{p_1}, \cdots, \boldsymbol{s}_{p_k} \in \boldsymbol{s}_{p_1}, \cdots, \boldsymbol{s}_{p_n}$ appropriately, and compute $V^{(k)}_{p_1,\cdots,p_k}(\Delta)$ and $V^{(n-k)}_{p_{k+1},\cdots,p_n}(\Delta)$ by Algorithm $V_\Delta$.*

3. *Apply the A* algorithm with the estimator $\hat{h}_{\text{one-}k}$ to the graph $G$ in the direction from $s$ to $t$.*

Further we can extend Algorithm 2 to a recursive version. In other words, we compute each $V^{(k)}_{p_1,\cdots,p_k}(\Delta)$ with the A* algorithm utilizing the information of $(k-1)$-dimensional subgraph $G^{(k-1)}_{p_1,\cdots,p_{k-1}}$. In Algorithm 2 we compute $V^{(k)}_{p_1,\cdots,p_k}(\Delta)$ using the A* algorithm with the estimator $\hat{h}_{\text{pair}}$ in step 1. of Algorithm $V_\Delta$, which only utilizes the information of 2-dimensional subgraph. This may cause wastefully wide estimate of actual $V^{(k)}_{p_1,\cdots,p_k}(\Delta)$ in step 2. of Algorithm $V_\Delta$. This recursion requires solving a lot of search problems, which may cause the increase of computational time, thus it seems meaningless at a glance. However, by this recursion, we can save the search space we must hold at a time. Since it is often occurred in the large-scale multiple sequence alignment problem that the computation nearly comes to a halt due to a lack of memory caused by the enormous search space, this improvement is not so worthless.

## 7 Experimental Results

In this section, we examine the efficiency of our approach. We performed several experiments aligning actual sequences of proteins. In the experiments, the PAM-250 matrix was used for score function, and gap penalty was $-8$, which is the minimum value in the PAM-250 matrix. As to the upper bound in subsection 3.1, the actual shortest path length was utilized in order to examine the best possible cases. All the experiments were done on Sun Ultra 2 workstation with 1024 megabyte memory.

### 7.1 Cases with High Similarity

We used 9 sequences, elongation factor TU (EF-TU) of Haloarcula marismortui and Methanococcus vannielii, and elongation factor $1\alpha$ (EF-$1\alpha$) of Thermoplasma acidophilum, Thermococcus celer, Sulfolobus acidocaldarius, Entamoeba histolytica, Plasmodium falciparum, Stylonychia lemnae, and Euglena Gracilis. The length of each sequence is about 420-450, and the similarity among sequences is quite high. We computed the optimal alignments of first $7 \leq n \leq 9$ sequences of them. Table 1 shows the results of these experiments. In our programs, we use heaps for management of vertices in graphs. The column of 'max #nodes in heaps' shows at most how many vertices are in heaps at a time, which is roughly in proportion to the required memory space. The column of 'total # $V_\Delta$' shows the sum of the number of vertices in all required $V_\Delta$. 'estimate search' means step 2. of Algorithm 1 or 2, and 'final search' means step 3. of them. '$\hat{h}_{\text{one-}k}$, rec' in the column of 'estimator' means that the computations of the corresponding rows are done with the recursive version algorithm. In the column of 'final search', '#visited' shows the number of vertices whose $\hat{g}$ values are renewed at least once in the search of step 3., while '#expanded' shows the number of vertices whose actual $g$ values have been computed (i.e. expanded).

The estimator $\hat{h}_{\text{one-}k}$ for $k = \lceil n/2 \rceil$ works well in view of computational time in most of cases, and the recursive-estimate algorithm with the estimator $\hat{h}_{\text{one-}k}$ always saves the required search space in comparison with the corresponding non-recursive algorithm with $\hat{h}_{\text{one-}k}$, except for the case $n = 7$ and $k = 5$. When $k$ becomes larger, the estimator $\hat{h}_{\text{one-}k}$ gives tighter bound for $h$, and the final search goes more efficiently. However, the cost of computing $V_\Delta$ increases, and the total efficiency becomes lower. In the rows of '$\hat{h}_{\text{one-}k}$, rec', the column of 'max #nodes in heap' sometimes stays constant for the same $n$. For example, the columns 'max #nodes in heap' of '$\hat{h}_{\text{one-}k}$, rec' for $n = 8$ are always 480,366. This is only because the computation of $V_\Delta$ for $k = 4$ takes search space most, and it does not necessarily hold. However, we can say the effect of recursive-estimate grows larger, if $k$ increases with the same $n$.

The estimator $\hat{h}_{\text{all-}k}$ is inefficient in view of both computational time and required search space, although it gives the tightest bound for $h$ with fixed $k$ and $n$. One reason of this will be that the number of searches required for computing all $V_\Delta$ is $\binom{n}{k}$, which is quite large.

Although the A* algorithm with the estimator $\hat{h}_{\text{pair}}$ could not compute the optimal alignment for 9 sequences in a few hours, the A* algorithm with the estimator $\hat{h}_{\text{one-}k}$ finished computing it in less than one hour. The effect of the new estimator $\hat{h}_{\text{one-}k}$ appears more clearly if the number of sequences $n$ increases.

### 7.2 Cases with Low Similarity

We used 6 sequences, Extracellular globin of Lumbricus terrestris - AIII, Myoglobin of Aplysia limacina, Dimeric myoglobin of Busycon canaliculatum, Monomeric hemoglobin of Chironomus thummi

thummi - VIIA, Monomeric insect hemoglobin of Chironomus thummi thummi - IIIa, and Monomeric hemoglobin of Lampetra fluviatilis. The length of each sequence is about 140-160. The similarity among sequences is quite low. We computed the optimal alignments of first $5 \leq n \leq 6$ sequences of them. Table 2 shows the results of these experiments. The A$^*$ algorithm with the estimator $\hat{h}_{\text{one-}k}$ could compute the optimal alignment of 6 sequences for about one hour, which the A$^*$ algorithm with the estimator $\hat{h}_{\text{pair}}$ could not finish computing in a few hours. The effect of new estimator $\hat{h}_{\text{one-}k}$ appears more clearly than the cases with high similarity, and this time the recursive-estimate algorithm works well in view of both computational time and required search space.

| $n$ | estimator | $k$ | $\Delta$ | max #nodes in heaps | total $\# V_\Delta$ | final search | | time | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | #visited | #expanded | estimate (sec) | final search (sec) | total (sec) |
| 7 | $\hat{h}_{\text{pair}}$ | 2 | - | 48985 | - | 48985 | 48050 | - | 64.00 | 64.00 |
| | $\hat{h}_{\text{one-}k}$ | 4 | 145 | 148403 | 71718 | 7869 | 7769 | 20.79 | 10.87 | 31.66 |
| | | 5 | 102 | 128395 | 22025 | 2436 | 2358 | 28.64 | 2.60 | 31.24 |
| | | 6 | 51 | 271351 | 3590 | 685 | 685 | 48.22 | 0.28 | 48.50 |
| | $\hat{h}_{\text{one-}k}$, rec | 4 | 145 | 130036 | 71718 | 7869 | 7769 | 25.15 | 10.95 | 38.73 |
| | | 5 | 102 | 130036 | 22025 | 2436 | 2358 | 38.20 | 2.84 | 43.69 |
| | | 6 | 51 | 130036 | 3590 | 685 | 685 | 44.94 | 0.28 | 47.86 |
| | $\hat{h}_{\text{all-}k}$ | 3 | 153 | 959750 | 956325 | 3425 | 3339 | 156.49 | 77.74 | 236.20 |
| | | 4 | 153 | 1628248 | 1627406 | 842 | 831 | 699.66 | 17.07 | 716.75 |
| | | 5 | 106 | 456722 | 456133 | 589 | 581 | 555.66 | 2.82 | 558.49 |
| | | 6 | 63 | 270838 | 35643 | 496 | 496 | 369.35 | 0.25 | 369.62 |
| 8 | $\hat{h}_{\text{pair}}$ | 2 | - | 545274 | - | 545274 | 541804 | - | 1902.00 | 1902.00 |
| | $\hat{h}_{\text{one-}k}$ | 4 | 239 | 524319 | 330092 | 47686 | 47541 | 118.68 | 173.48 | 292.16 |
| | | 5 | 203 | 746706 | 218350 | 31209 | 30601 | 210.32 | 108.25 | 318.57 |
| | | 6 | 159 | 896020 | 90307 | 10819 | 10686 | 508.88 | 30.00 | 538.88 |
| | | 7 | 62 | 2581205 | 5095 | 1237 | 1221 | 1245.47 | 1.18 | 1246.65 |
| | $\hat{h}_{\text{one-}k}$, rec | 4 | 239 | 480366 | 330092 | 47686 | 47541 | 138.43 | 172.93 | 314.92 |
| | | 5 | 203 | 480366 | 218350 | 31209 | 30601 | 234.39 | 111.75 | 349.71 |
| | | 6 | 159 | 480366 | 90307 | 10819 | 10686 | 402.83 | 32.33 | 438.73 |
| | | 7 | 62 | 480366 | 5095 | 1237 | 1221 | 456.88 | 1.19 | 461.64 |
| 9 | $\hat{h}_{\text{one-}k}$ | 5 | 307 | 3035758 | 1085619 | 145809 | 144756 | 1345.41 | 1658.33 | 3003.74 |
| | | 6 | 265 | 5404970 | 914629 | 58121 | 57330 | 6158.02 | 536.52 | 6694.54 |
| | $\hat{h}_{\text{one-}k}$, rec | 5 | 307 | 2292052 | 1085619 | 145809 | 144756 | 1410.67 | 1747.02 | 3162.30 |
| | | 6 | 265 | 2649782 | 914629 | 58121 | 57330 | 4542.51 | 672.92 | 5220.04 |

Table 1: Results for the alignments of EF-TU and EF-1$\alpha$.

| $n$ | estimator | $k$ | $\Delta$ | max #nodes in heaps | total $\# V_\Delta$ | final search | | time | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | #visited | #expanded | estimate (sec) | final search (sec) | total (sec) |
| 5 | $\hat{h}_{\text{pair}}$ | 2 | - | 421791 | - | 421791 | 421007 | - | 420.00 | 420.00 |
| | $\hat{h}_{\text{one-}k}$ | 3 | 300 | 308514 | 103947 | 204567 | 203314 | 13.74 | 62.48 | 76.22 |
| | | 4 | 127 | 431782 | 81714 | 26627 | 26479 | 52.30 | 6.50 | 58.80 |
| | $\hat{h}_{\text{one-}k}$, rec | 4 | 127 | 334645 | 81714 | 26627 | 26479 | 58.79 | 6.32 | 65.26 |
| 6 | $\hat{h}_{\text{one-}k}$ | 4 | 421 | 3791554 | 1153970 | 2637584 | 2626672 | 678.75 | 3163.66 | 3842.41 |
| | | 5 | 222 | 9773133 | 858553 | 239686 | 236818 | 7852.13 | 208.17 | 8060.30 |
| | $\hat{h}_{\text{one-}k}$, rec | 4 | 421 | 3010833 | 1153970 | 2637584 | 2626672 | 665.97 | 2938.69 | 3604.89 |
| | | 5 | 222 | 4185955 | 858553 | 239686 | 236818 | 3458.79 | 214.29 | 3673.31 |

Table 2: Results for the alignments of globin.

## 8 Concluding Remarks

In this paper, improvement of the A$^*$ algorithm for the multiple sequence alignment problem has been considered, by introducing two new powerful estimators $\hat{h}_{\text{one-}k}, \hat{h}_{\text{all-}k}$ which utilize $k \geq 3$ dimensional sub-alignments. A new bounding technique using $V_\triangle$ has been also proposed. Our algorithm using estimator $\hat{h}_{\text{one-}k}$ is also extended to a recursive-estimate version.

For fixed $k$ and $n$, the estimator $\hat{h}_{\text{all-}k}$ usually gives the tightest lower bound for actual length $h$ of the shortest path to $t$, among three estimators $\hat{h}_{\text{one-}k}, \hat{h}_{\text{all-}k}$, and the original $\hat{h}_{\text{pair}}$ proposed by Ikeda and Imai. However, computing $\hat{h}_{\text{all-}k}$ requires a lot of searches. In our experiments, using $\hat{h}_{\text{all-}k}$ turned out to be inefficient, although the search goes very effeciently after required values of estimator $\hat{h}_{\text{all-}k}$ has been computed. Thus the main difficulty when using $\hat{h}_{\text{all-}k}$ lays in computing required values of this estimator.

On the other hand, it turned out to be quite efficient to use the estimator $\hat{h}_{\text{one-}k}$ in view of computational time. The effect of using $\hat{h}_{\text{one-}k}$ grows larger when the number of sequences $n$ increases, or the similarity among sequences is lower. The recursive-estimate version of this $\hat{h}_{\text{one-}k}$ saves the required search space, which sometimes decreases computational time. This effect also grows larger when the number of sequences $n$ increases, or the similarity among sequences is lower.

As for future works, we should extend these alignment algorithms based on the A$^*$ algorithm to cope with the affine gap costs. We should also develop more efficient estimators and bounding techniques.

## Acknowledgement

This work was supported in part by the Grant-in-Aid for Scientific Research on Priority Areas, “Genome Science” from the Ministry of Education, Science, Sports and Culture of Japan.

## References

[1] Carrilo, H. and Lipman, D.J., The Multiple Sequence Alignment Problem in Biology, *SIAM J. Appl. Math.*, 48(5):1073–1082, 1988.

[2] Gupta, S.K., Kececioglu, J.D., and Schaffer, A.A., Improving the Practical Space and Time Efficiency of the Shortest-paths Approach to Sum-of-pairs Multiple Sequence Alignment, *J. Computational Biology*, 2(3):459–472, 1995.

[3] Hart, P.E., Nillson, N.J., and Rafael, B., A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *IEEE Trans. Sys. Sci. and Cyb.*, SSC-4:100–107, 1968.

[4] Ikeda, T. and Imai, H., Fast A$^*$ Algorithms for Multiple Sequence Alignment, *Proc. of 5th Workshop on Genome Informatics*, 90–99, 1994.

[5] Ikeda, T., Applications of the A$^*$ Algorithm to Better Routes Findings and Multiple Sequence Alignment, A Master’s Thesis, Department of Information Science, University of Tokyo, 1995.

[6] Shibuya, T. and Imai, H., Enumerating Suboptimal Alignments of Multiple Biological Sequences Efficiently, *Proc. of Pacific Symposium on Biocomputing*, Maui, 409–420, 1997.

[7] Shibuya, T., New Approaches to Flexible Alignment of Multiple Biological Sequences, A Master’s Thesis, Department of Information Science, University of Tokyo, 1997.

[8] Spouge, J.L., Speeding Up Dynamic Programming Algorithms for Finding Optimal Lattice Paths, *SIAM J. Appl. Math.*, 49(5):1552–1566, 1989.