New Flexible Approaches for Multiple Sequence Alignment

Tetsuo SHIBUYA and Hiroshi IMAI

Department of Information Science, Faculty of Science, University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan
Tel: +81-3-3812-2111 ext.4102 Fax: +81-3-5800-6933

Keywords: multiple sequence alignment, algorithm, suboptimal, enumeration, parametric

Abstract

The multiple sequence alignment problem is applicable and important in various fields in molecular biology such as the prediction of three dimensional structures of proteins and the inference of phylogenetic tree. However, the optimal alignment based on the scoring criterion is not always the biologically most significant alignment. We here propose two flexible and efficient approaches to solve this problem.

One approach is to provide many suboptimal alignments as alternatives for the optimal one. It has been considered almost impossible to investigate such suboptimal alignments of more than two sequences because of the enormous size of the problem. We propose techniques for enumeration of suboptimal alignments using Eppstein algorithm. We also discuss what kind of suboptimal alignment is unnecessary to enumerate, and propose an efficient enumeration algorithm to enumerate only necessary alignments.

The other approach is parametric analysis. The obtained optimal solution with fixed parameters such as gap penalties is not always the biologically best alignment. Thus, it is required to vary parameters and check how the optimal alignments change. The way to vary parameters has been studied well on the problem of two sequences, but not on the multiple alignment problem because of the difficulty of computing the optimal solution. We propose techniques for this parametric multiple alignment problem, and examine the features of obtained alignments by various parametric analyses.

For both approaches, this paper performs experiments on various groups of actual protein sequences, and examines the efficiency of these algorithms and property of sequence groups.

1 Introduction

The multiple alignment is a problem to obtain the alignment of multiple sequences with the highest score based on some given scoring criterion between characters. This problem appears in various fields of molecular biology such as the prediction of three dimensional structures of proteins and the inference of phylogenetic tree.

The method using dynamic programming (DP) is well-known for the alignment problems (Dayhoff et al. 1978, Gotoh 1995, Spouge 1989, Waterman 1995). This method requires $O(n^d)$ time and space for d sequences of length at most n. This method can be applied when n is not so large and d is 2 or 3, but for larger problems, it is impractical. The A* algorithm is a well-known algorithm for the general optimization and search problems. This algorithm can reduce the search space dramatically if a powerful estimator is used. Thus the A* algorithm with upper bounding operation is proposed recently for computing the optimal alignment of multiple sequences (Ikeda & Imai 1994, Ikeda 1995).

The optimal alignment based on the scoring criterion is not always the biologically most significant alignment. We propose two methods to cope with this problem. One is to enumerate suboptimal solutions and the other is to do parametric analysis.

A suboptimal alignment is an alignment whose score is close to the optimal one. In fact, in case of aligning two sequences, the suboptimal alignments problem is well-studied (Chao 1994, Naor et al. 1993, Shibayama et al. 1993, Zuker 1991) and used for many applications such as predicting protein structure and so on (Saqi et al. 1991, Saqi et al. 1992, Waterman et al. 1987). In multiple alignment problem, we can see suboptimal alignments of each pair of sequences with these methods for only two sequences as in (Zuker 1991), but these are not the accurate suboptimal alignments of all the sequences.

Enumeration of the suboptimal alignments had not been considered as very practical even in the case of aligning two sequences (Naor et al. 1993, Zuker 1991). But such enumeration becomes easier because a new efficient algorithm for the k shortest paths problem is proposed by Eppstein (Eppstein 1994, Shibuya et al. 1995). This algorithm enumerates the lengths of the k shortest paths in $O(k+n+m)$ time and space if we are given the shortest path tree from the source or to the destination for any graph with non-negative m edges and n vertices. Even if we have to output the paths themselves, this algorithm requires only time linear to the output size, add to the time given above. Note that the shortest path tree can be constructed with DP or A* algorithm.

For this approach, we first discuss the method to obtain E_Δ, which can be done with some extension of the A* algorithm. E_Δ represents all aligned groups of residues in optimal and suboptimal alignments which are at most Δ worse than the optimal. Furthermore, based on this extended A* algorithm and the Eppstein algorithm, we go on to discuss the methods for the enumeration problem.

Number of suboptimal solutions is very large, and we should restrict the outputs to important solutions if we know what kind of solutions is important. Hence we discuss what kind of suboptimal alignment is unnecessary to enumerate, and propose an efficient technique to enumerate only necessary alignments. This technique is so splendid that it remains output size sensitive even though we restricted solutions only to the necessary alignments in our concept.

For the other approach of parametric analysis, we first review the basic techniques for parametric analysis (Gusfield et al. 1992, Huang et al. 1994, Vingron et al. 1994, Waterman 1994, Waterman et al. 1992, Zimmer 1997). and propose new techniques for multiple alignments. As for the techniques, we use Eppstein algorithm to examine all the optimal solutions for one fixed parameter, and upper bounding technique for the parametric alignment. In most of previous works, they computed only one optimal solution for one fixed parameter in parametric analysis. We enumerate all the optimal solutions because the parametric analysis is the analysis of optimal solutions and we consider we should examine all optimal solutions. Fortunately, it is reasonable to obtain with Eppstein algorithm.

Then we do a parametric study on gap penalties, score tables and weight matrices. Related with the weight matrices, we show the (enhanced) A* algorithm is applicable for the weighted multiple alignment problem. Weighted problem is considered only when the phylogenetic tree is given, but our approach enables more flexible study of the weighted multiple alignment problem. This technique may also be useful in constructing or tuning phylogenetic tree.

For both approaches, this paper performs experiments on various groups of actual protein sequences, and examines the efficiency and practicality of these algorithms and property of sequence groups we use.

2 Preliminaries

In this section, we first explain the definition of the multiple alignment problem, and then survey the exact algorithms for it briefly.

2.1 Multiple Sequence Alignment Problem

In this section, we describe what the multiple sequence alignment problem is. These are given in this problem:

- A fixed set of alphabets Σ including a gap which is denoted by `-`.
- A score function between characters $s : \Sigma \times \Sigma \rightarrow R$.

Each member in Σ except for a gap is called a character, and a finite string of characters is called a sequence. On the other hand, a finite string of members in Σ is called a padded sequence. Then the set of ith elements of two or more padded sequences is called a ith column of the set of the sequences. Furthermore, a set of consecutive columns is called a region. A set of padded sequences $(S'_1, S'_2, \ldots, S'_d)$ is called a d-alignment or simply an alignment of $(S_1, S_2, \ldots, S_d)$ if and only if all of the padded sequences have same length l, ith column contains at least one character for any $i(1 \leq i \leq l)$, and $(S'_1, S'_2, \ldots, S'_d)$ becomes $(S_1, S_2, \ldots, S_d)$ if all gaps are deleted. Furthermore, the (pairwise) projection A_{ij} of an alignment $A = (S'_1, S'_2, \ldots, S'_d)$ is defined as the alignment obtained from (S'_i, S'_j) by removing columns without any characters.

The score of a 2-alignment is defined as the summation of scores of all columns obtained directly with the score function. The score of a d-alignment is defined as the summation of the scores of all the pairwise projections of the d-alignment. Finally, we can define the problem: the alignment problem is a problem of finding the alignment of given sequences with the largest score.

In this definition of the problem, the gap penalty is linear to sum of the length of gap sequences. This kind of a gap penalty is called the linear gap penalty. If there is some starting gap penalty which is costed to the starting of the gap sequence, it is called the affine gap penalty. In this thesis, we mainly deal with the linear gap penalty.

The weighted sum-of-pairs multiple alignment problem (Altsuchul 1989, Gotoh 1995) is a generalization of the simple sum-of-pairs multiple alignment problem described above. This version of the problem is often used when the phylogenetic tree is given. In this problem, we optimize sum of weighted score of each pairwise sequences alignment: we multiply the score of the alignment of the ith and the jth sequence by w_{ij}. We call (w_{ij}) a weight matrix. Note that (w_{ii}) is always 0 for any i. We deal with this kind of problem only in the section 4.3.3.

The number of characters in Σ is 20 in the case of aligning protein sequences, and it is 4 in the case of aligning DNA sequences. The score function is usually given by a score table. Table 1 shows the most famous and widely-used score table called PAM-250 (Altsuchul 1991, Dayhoff et al. 1978). It is one of the variations of PAM matrices proposed by Dayhoff (Dayhoff et al. 1978).

The multiple alignment problem can be easily transformed to the shortest path problem on some grid-like directed acyclic graph with no negative edges. Let S_k be the kth sequence of d

sequences to be aligned, and $n_k = O(n)$ be the length of S_k. Then suppose a directed acyclic graph $G = (V, E)$ such that $V = \{(x_1, \ldots, x_d) | x_i = 0, 1, \ldots, n_i\}$ and $E = \{(v, v + e) | v \in V, e \in [0, 1]^d, e \neq \mathbf{0}\}$. In this graph, a path from $s = (0, \ldots, 0)$ to $t = (n_1, \ldots, n_d)$ corresponds to an alignment of the sequences.

In the alignment problem of two sequences, the length of an edge is defined from the score table between characters, and the length of a path from s to t equals the score of the corresponding alignment. Figure 1 shows an example of it. In the graph, each diagonal edge corresponds to the match of characters, and each horizontal or vertical edges corresponds to the inserted gaps. In the multiple alignment problem, the sum of all the scores for alignments of pairwise sequences is generally used as the score. This score of the alignment equals the length of the corresponding path, if we define the length of each edge as the sum of the lengths of the corresponding edges in the pairwise projections of the alignment. In this way, the longest path problem from s to t in this graph is equivalent to the original alignment problem. This longest path problem can be easily transformed to the shortest path problem by reversing the signs of the lengths (Gupta et al. 1995, Ikeda & Imai 1994, Ikeda 1995). Thus we can use shortest path algorithms for graphs like above to compute the optimal alignment of given sequences.

2.2 Exact Algorithms for Alignment Problem

2.2.1 Dynamic Programming

The alignment graph presented in the section 2.1 is a layered graph and we can easily use dynamic programming (DP) to obtain the shortest path (Dayhoff et al. 1978, Gotoh 1992, Gotoh 1995, Spouge 1989, Waterman 1995) in time linear to the size of the graph size.

Algorithm 1 (Dynamic Programming) *Let $l(u, v)$ be the length of the edge (u, v) in the alignment graph, s be the source $(0, 0, \ldots, 0)$ and t be the destination $(n_1, n_2, \ldots, n_d)$.*

1. *Let $p(s)$ be 0.*
2. *For $i = 1$ to $i = \sum_{1 \le j \le d} n_j$ do the following:*
   For all $v = (x_0, x_1, \ldots, x_d)$ such that $\sum_{1 \le k \le d} x_k = i$, compute the following value $p(v)$, and let previous(v) be $v - e$ which satisfies this equation (1). Note that this $p(v)$ equals to the shortest path length from s to v.

$$p(v) = \min_{e \in [0,1]^d, e \neq \mathbf{0}} (p(v - e) + l(v - e, v)) \qquad (1)$$

3. *We can obtain the shortest path by tracing back previous(v) from the destination t.*

But the DP requires $O(n^d)$ memory space where n is the length of the longest sequence and d is the number of sequences, because it needs to store all vertices of the graph in memory. Thus the DP requires too much memory in the case of large d, and the DP can only deal with alignment of 2 or 3 sequences in ordinary case of aligning actual protein sequences.

2.2.2 Dijkstra Method

Dijkstra method is the most famous algorithm for the shortest path problem on graphs without negative edges. The outline of this algorithm is as follows:

Algorithm 2 (Dijkstra Method) *Let the graph in assumption be $G = (V, E)$, $l(v, w)$ be the length of the edge (v, w) which is always non-negative, s be the source and t be the destination.*

1. *Let $p(v)$ be the potential of a vertex v, which is initialized as $+\infty$ except for the source s whose potential is zero, and S be an empty set.*

2. *Add vertex v_0 which has the minimum potential in $V - S$. Then, stop if v_0 is t.*

3. *For all edges (v_0, v) in E, if $p(v_0) + l(v_0, v)$ is smaller than $p(v)$, replace $p(v)$ with $p(v_0) + l(v_0, v)$ and replace the path kept in v with the shortest path from s to v_0 added with the edge (v_0, v).*

4. *Go to step 2.*

This algorithm requires $O(m + n \log n)$ time where m is the number of the edges and n is the number of the vertices in the graph, which is worse than DP, and this algorithm itself is not so useful for alignment problem. But it will be remarkably more efficient if it is extended to the A* algorithm as in the next section 2.2.3.

2.2.3 A* algorithm

The A* algorithm will not search the whole graph in finding the shortest path if a good estimate for the shortest path length from each vertex to the destination t can be used.

The basic algorithm for the A* algorithm is like following:

Algorithm 3 (A* Algorithm) *Let the graph in assumption be $G = (V, E)$, $l(v, w)$ be the length of the edge (v, w) which is always non-negative, s and t be the source and the destination, and $h(v)$ be heuristic estimate for the length of the shortest path to t which is not longer than the actual one.*

1. *Let $p(v)$ be the potential of a vertex v, which is initialized as $+\infty$ except for the source s whose potential is zero, and S be an empty set.*

2. *Add vertex v_0, whose value of $p(v) + h(v)$ is smallest in $V - S$, to S. Then, stop if v_0 is t.*

3. *For all edges (v_0, v) in E, if $p(v_0) + l(v_0, v)$ is smaller than $p(v)$, replace $p(v)$ with $p(v_0) + l(v_0, v)$ and replace the path kept in v with the shortest path from s to v_0 added with the edge (v_0, v). If v is in S, remove it from S.*

4. *Go to step 2.*

In this algorithm, $p(v)$ for vertex v in S is also the length of the shortest path from s to v. $p(v) + h(v)$ is the estimate for the shortest path from s to t via v. The searched vertices by the A* algorithm is always within searched vertices by the Dijkstra method. In this way, the A*

algorithm can get the shortest path more effectively. If the estimate $h(v)$ equals to the actual shortest path length from v to t, this algorithm searches only on the shortest path.

The estimate $h(v)$ must not be longer than the shortest path length from v to t, because the final obtained path must not be longer than the other paths. If some of the estimate $h(v)$ is longer than the actual shortest path, this algorithm is called A algorithm, which is often used as an approximate algorithm for the shortest path problem.

In the A* algorithm, the shortest path from s may not appear first, and a shorter path may be found in the future search, which is the reason of the removal of vertices from S in step 3. It makes this algorithm rather inefficient. This can be avoided if the estimator is dual feasible. The definition of 'dual feasible' is as follows:

Definition 1 *The estimator h for the shortest path to t is called dual feasible if and only if h satisfies the following constraint:*

$$\forall (u,v) \in E \quad l(u,v) + h(v) \geq h(u) \tag{2}$$

If the estimator is dual feasible, the A* algorithm can be easily translated to the Dijkstra method by modifying the length of the edges (Hsu 1994, Ikeda et al. 1994, Ikeda 1995):

Theorem 1 (Ikeda Hsu et al.) *Let h be a dual feasible estimator for s. The Dijkstra method on a graph in which the length of the edge (u,v), or $l(u,v)$ is replaced by $l'(u,v)$ as follows is equivalent to the A^* algorithm on the original graph.*

$$l'(u,v) = l(u,v) + h(v) - h(u) \tag{3}$$

Proof: $l'(u,v)$ is non-negative because of dual feasibility of h. Thus, the shortest path can be searched with the Dijkstra method in the modified graph. Let p be the shortest path from s to v. Then the potential of v used in searching with the Dijkstra method on the modified graph is described as follows:

$$\begin{aligned} p(v) &= \sum_{(u,v)\in p} l'(u,v) \\ &= \sum_{(u,v)\in p} l(u,v) + h(v) - h(s) \end{aligned}$$

This means the Dijkstra method on the new graph is equivalent to the A* algorithm on the original graph, because $h(s)$ is constant. □

Ikeda and Imai (Ikeda & Imai 1994) show the following estimator is very useful for alignment problem in case $d > 2$. Let G_{ij} be the corresponding graph to the alignment of S_i and S_j, v_{ij} be the corresponding vertex in G_{ij} to v in G, and $L^*(u,v)$ be the shortest path length from u to v. Then $h(v) = \sum_{1\leq i<j\leq d} L^*(u_{ij}, v_{ij})$ can be used as a powerful estimator for the multiple alignment problem. This estimator is easily be shown to be dual feasible, *i.e.* $l(u,v) + h(v) \geq h(u)$. Hence the A* algorithm can be applied as following.

Algorithm 4 (A* algorithm for the Alignment Problem)

1. *For each of i and j $(1 \leq i < j \leq d)$, apply DP to graph G_{ij} from t_{ij} to calculate $L^*(v_{ij}, t_{ij})$ for each v_{ij} in V_{ij}. Then let $h(v)$ be $\sum_{1\leq i<j\leq d} L^*(u_{ij}, v_{ij})$.*

2. *Modify the length of edge (u, v) in G as follows, using $h(v)$ above, and compute the shortest path with Dijkstra method.*

$$l'(u,v) = l(u,v) + h(v) - h(u) \tag{4}$$

Note that the time and space used for the DP in the step 1 is negligible, if d is large. This A* algorithm can deal with alignment problem of 5 to 6 normal sequences in reasonable time.

A vertex in the graph for the multiple alignment has $2^d - 1$ edges going out from it, and the A* algorithm examines all the descendant vertices and keeps in a heap the information about all of them. If an upper bound $L^+(s,t)$ for the s-t shortest path, which corresponds to the lower bound of the score of the alignment, is given, the necessary space for the heap can be reduced (Ikeda 1995): we can ignore w such that $L^*(s,v) + l(v,w) > L^+(s,t)$, when we examine the descendant vertices of v. If the necessary space for the heap is reduced, the computing time of the A* algorithm will be also reduced. This is called the enhanced A* algorithm.

Note that the branch-and-bound techniques implemented in `MSA` program (Gupta et al. 1995) is equivalent to this enhanced A* algorithm, and Araki et al. (Araki et al. 1993) showed that the estimated score computed directly by the score matrix is useful for the 2-alignment problem.

3 Enumeration of Suboptimal Alignments of Multiple Sequences

In this section, we deal with the first approach to the flexible alignment, that is enumeration of suboptimal solutions. At first, we introduce the Eppstein algorithm (Eppstein 1994) which is very efficient for enumeration of suboptimal paths in ordinary graphs. We then consider how to obtain E_Δ efficiently for multiple alignment problem. Furthermore, we also discuss how to enumerate the suboptimal solutions, introducing the new notation of classes of suboptimal solutions. Based on these algorithms, we also do experiments on actual protein sequences.

3.1 Eppstein Algorithm

Eppstein (Eppstein 1994) proposed an algorithm which finds implicitly the k shortest paths for the graph G with non-negative m edges and n vertices regardless of cycles, in $O(m+n+k)$ time after the shortest path tree is constructed. Eppstein (Eppstein 1994) also proposed an easier algorithm of $O(m + n \log n + k)$ time. Note that before the proposition of Eppstein, the best algorithm for such a problem was $O(k(m + n \log n))$.

In the algorithm, we use $\delta(u, v)$ for the edge (u, v) as in equation (6). This $\delta(u, v)$ denotes how much longer the path will be using the edge (u, v) than the optimal path by way of v, and therefore this value is always non-negative.

If an edge (u, v) is on the shortest path tree, $\delta(u, v)$ is zero, otherwise, it is called a sidetrack and $\delta(u, v)$ may not be zero. If we go along an s-t path p other than the shortest path, there must be one or more sidetracks on p, and we define $\mathit{sidetrack}(p)$ as the nearest sidetrack from s within them.

Let $(\mathit{tail}(p), \mathit{head}(p))$ be $\mathit{sidetrack}(p)$. Then we can suppose a heap, in which the parent of a path p is a path which is same as p from $\mathit{head}(p)$ to t, but go along the shortest path from s to $\mathit{head}(p)$ instead of using $\mathit{sidetrack}(p)$. We define this parent of p as $\mathit{parent}(p)$ and we call p a child of $\mathit{parent}(p)$. The root of the heap is the shortest path, and all the paths from s to t appear in the heap once. In this heap, p is $\delta(\mathit{sidetrack}(p))$ longer than $\mathit{parent}(p)$. Figure 2 shows the path heap of the alignment graph of two sequences `KN` and `QK`.

We call a heap i-heap if the node of the heap has only i children at most (it is not required to be balanced). The basic concept of the Eppstein algorithm is to modify this path heap to 4-heap, sharing as many nodes as possible. Figure 3 shows an example of this compact version of path heap of the same alignment graph in Figure 2, in which some of the nodes are shared and the number of nodes in it is reduced.

From this heap, we can obtain the k shortest paths in $O(k)$ time (Frederickson 1993), or $O(k \log k)$ time in sorted form. The following is the outline of this algorithm:

Algorithm 5 (Eppstein)

1. *Construct the shortest path tree from s to all the other vertices.*

2. *For each vertex v, construct $H_G(v)$, that is, a 3-heap of sidetracks (u', u), such that u is on the shortest path from s to v, ordered by $\delta(u', u)$. Let the length from the root of $H_G(v)$ to a node n be $\delta(u, v)$ if n represents sidetrack (u, v).*

(a) *For each vertex v, construct $H_{out}(v)$, that is a 2-heap in which the root has only one child, of sidetracks (v', v) ordered by $\delta(v', v)$.*

(b) *For each vertex v, construct $H_T(v)$, that is, a 2-heap of vertices on the shortest path from s to v ordered by the value δ of the root of the heap made in step 2-(a).*

(c) *Merge $H_{out}(v)$ and $H_T(v)$ to make $H_G(v)$. Then let the length of the edge from node n_1 to node n_2 in this heap be $\delta(n_2) - \delta(n_1)$.*

3. *For each v in G, make an edge from each node in $H_G(v)$ which represents a sidetrack (u', u) to the root of $H_G(u')$, and define the length of this new edge as the value of the root.*

4. *Make a new node for each v in G, and make an edge from this node to the root of $H_G(v)$. Let the length of this edge be δ of the root. Let this new graph be $P(G)$.*

Then we can find a heap $H_v(G)$ in $P(G)$ for any v, considering the root as the node made in step 4 for v, and the value of a node as the length from the root to the node. There is a one-to-one correspondence between the nodes in $H_v(G)$ and the paths from v to t in G, and the k smallest nodes in this virtual heap $H_v(G)$ correspond to the k shortest path. Moreover, we can easily restore the path from the node of the heap, which can be done in $O(n')$ time where n' is the size of the output alignment.

In this algorithm, we can also compute the k shortest paths using the shortest path tree to t, which is the original method described in the Eppstein's paper (Eppstein 1994). Note that the shortest path tree in step 1 is constructed generally by the Dijkstra method, but for problems such as the alignment problem, we can also use DP. Eppstein showed the step 2 can be done in $O(n+m)$ time, but this is a complicated algorithm and takes much time in practice, and we use a far more easier algorithm that can be done in $O(n \log n + m)$ time, which is also proposed by Eppstein (Eppstein 1994): we make $H_G(v)$ one by one from s to the other vertices along the shortest path tree, sharing as many nodes as possible.

3.2 Upper Bounding Technique for Computing E_Δ

E_Δ is a set of vertices which are used by the s-t paths whose lengths are at most Δ longer than the shortest path, and it corresponds to all aligned groups of residues in optimal and suboptimal alignments in original problem. The problem to compute it is well-studied (Ikeda 1995, Naor et al. 1993, Shibayama et al. 1993, Zuker 1991). Here we show how to compute this set E_Δ with A* algorithm (Ikeda 1995). For any path p from s to t, the modified path length by the expression (4) is only $h(t) - h(s)$ longer than the original length. This value is not relevant to p, thus E_Δ on the modified graph is same as the original one.

Theorem 2 *Any paths from s to t on a graph in which the length of the edge (u, v), or $l(u, v)$ is replaced by $l'(u, v)$ as in (4) are a constant shorter than those on the original graph.*

Proof: Let p be a path from s to t, and h be a dual feasible estimator for the shortest path length to t. Then the length of a path p or $length'(p)$ in new graph is described by the length of p or $length(p)$ in the original graph as follows:

$$length'(p) \quad = \quad \sum_{(u,v)\in p} l'(u, v)$$

$$
\begin{aligned}
&= \sum_{(u,v)\in p} l(u,v) + h(t) - h(s) \\
&= length(p) + h(t) - h(s) \qquad (5)
\end{aligned}
$$

According to this, all the paths on the new graph from s to t are $h(s) - h(t)$, which is constant, shorter than those on the original graph. □

Note that the following corollaries can be easily derived from this Theorem 2.

Corollary 1 *E_Δ related to the paths from s to t on a graph in which the length of the edge (u, v), or $l(u, v)$ is replaced by $l'(u, v)$ as in (4) are same as that on the original graph.*

Corollary 2 *The k shortest paths from s to t on a graph in which the length of the edge (u, v), or $l(u, v)$ is replaced by $l'(u, v)$ as in (4) are same as those on the original graph.*

Hence, first we modify the edge lengths with some dual feasible estimator, and then we can obtain E_Δ with the Dijkstra method as follows (Ikeda 1995) on this modified graph.

Algorithm 6 (E_Δ)

1. *Search from s by the Dijkstra method until the shortest path from s to t is discovered.*
2. *Search successively until a vertex v, to which the shortest path from s is more than Δ longer than the s-t shortest path, is discovered.*
3. *Modify the length of each edge (u, v) to $\delta(u, v)$ as follows:*

$$\delta(u, v) = l(u, v) + L^*(s, u) - L^*(s, v) \qquad (6)$$

4. *Apply the Dijkstra method from t until a vertex from which the shortest path to t is longer than Δ is discovered in this modified graph. E_Δ is the set of vertices searched in this step.*

A vertex in the graph for the multiple alignment has $2^d - 1$ edges going out from it, and the Dijkstra algorithm examines all the descendant vertices and keeps the information about all of them. If an upper bound $L^+(s, t)$ for the s-t shortest path is given, we can also reduce the necessary space for heap as in the case of computing the optimal solution with enhanced A* algorithm: we can ignore w such that $L^*(s, v) + l(v, w) > L^+(s, t) + \Delta$, when we examine the descendant vertices of v.

In general, such kind of an upper bound is difficult to obtain. However, we can use the actual shortest path length obtained in step 1 for the upper bound in step 2: we can ignore w such that $L^*(s, v) + l(v, w) > L^*(s, t) + \Delta$. Note that if we are given some upper bound of the solution before computing the optimal, we can of course use it too.

3.3 Extending Eppstein Algorithm to Reduce Memory Space

In this section, we discuss how to enumerate efficiently all the suboptimal alignments whose scores are at most Δ lower than the optimal one. The original Eppstein algorithm requires searching all over the graph, and requires much memory. But it is evident that we only have to apply the Eppstein algorithm in the subset $E_\triangle$ after computing $E_\triangle$ as in the previous section, and then search the Eppstein's heap structure with the depth first method.

However, if we use the easier $O(n \log n + m)$ algorithm in step 2 (b) of Eppstein algorithm (algorithm 5) in section 3.1, we do not have to compute $E_\triangle$ additionally.

First, we must take the step 1 and 2 in the section 3.2, using upper bounding technique. These procedures cannot be skipped. After these procedures, we implement the Eppstein algorithm as follows:

Algorithm 7 (Eppstein algorithm with A*)

1. *Construct the Eppstein's heap structure only on the shortest path.*
2. *Search for suboptimal solutions which are at most Δ worse than the optimal (root) from the root of $H_s(G)$ with depth first search method. If we encounter $H_G(v)$ which has not been constructed yet, we construct the heap structures of vertices on the shortest path from s to v for which we have not not constructed heaps yet.*

With this method, when we finish enumerating all the suboptimal alignments, the set of vertices for which we constructed the Eppstein heap is also $E_\triangle$.

Theorem 3 *The sets S of the vertices for which the Eppstein's heap structures are computed in the algorithm 7 is $E_\triangle$.*

Proof: We construct $H_G(v)$ for all the vertices in $E_\triangle$ in the algorithm 7. Thus we can easily see that $E_\triangle \in S$. A newly encountered vertex v for which $H_G(v)$ has not constructed is in $E_\triangle$, because we only search suboptimal paths which are Δ longer than the shortest path at most. Add to that, to obtain $H_G(v)$ for some vertex v, we only have to compute $H_G(u)$ for each vertex u on the shortest path from the source s to v in the easier method of the step 2(b) in algorithm 5, which is also trivially in $E_\triangle$. Thus, we do not construct Eppstein's heap structures for vertices outside of $E_\triangle$. Hence, we conclude that $S = E_\triangle$. $\square$

Thus we do not have to compute $E_\triangle$ additionally. Notice that this technique can be also used in general graphs other than the graphs for alignments.

3.4 Classification of Suboptimal Alignments

In this section, we classify suboptimal alignments. We introduce a notion of alignment class D_i as follows:

Definition 2 *D_i is a class of alignments which have i regions different from the optimal alignment, which is in D_0.*

Figure 5 shows some examples of suboptimal multiple alignments of protein fragments. (a) is the optimal alignment, and (b), (c) and (d) are suboptimal alignments. The regions bounded by boxes are the regions which are different from the optimal alignment. (b) and (c) have only one such region. On the other hand, (d) has two, both of which appear also in (b) or (c).

According to our notion of classification, the optimal alignment (a) is in the class D_0 (and none of the others are in this class), suboptimal alignments (b) and (c) are in the class D_1, and suboptimal alignment (d) is in the class D_2.

Considering the fact that we can easily reconstruct (d) from (b) and (c), (d) is not so important as (b) nor (c) and is sometimes unnecessary to enumerate. Thus, it will be a good news if we can efficiently enumerate only the alignments in the classes D_0 and D_1.

3.5 Avoiding Unnecessary Alignments

The paths to which the alignments in the class D_i corresponds branch off i times from the shortest path to which the alignment in the class D_0 corresponds. Hence, we can consider a very easy branch-and-bound technique to avoid such alignments in D_i ($i \geq 2$) in enumeration of suboptimal alignments: when we search the Eppstein's heap structure, if $head(p)$ of s-t path p is on the s-t shortest path and $parent(p)$ is not the shortest path, ignore p and its all conceptual children (defined in section 3.1). Figure 6 shows the concept of this technique.

Recall that one of the good feature of the Eppstein algorithm is that we can obtain the solutions in the time linear to the output size after having constructed the Eppstein heap. But the above technique is not output sensitive, because it requires checking some of the nodes which are in the class D_2. If there are many such solutions, the computing time becomes rather large.

Thus we modified the Eppstein algorithm to overcome this problem. Here we show how to construct a heap structure whose nodes represent only alignments in classes D_0 (the root or the optimal solution) and D_1:

Algorithm 8 (Modified Eppstein Algorithm)

1. *Construct the shortest path tree from s to all the other vertices.*

2. *For each vertex v, construct $H_G(v)$, that is, a 3-heap of sidetracks (u', u) ordered by $\delta(u', u)$ as follows. Let the length from the root of $H_G(v)$ to a node n be $\delta(u, v)$ if n represents sidetrack (u, v).*

   (a) *For each vertex v, construct $H_{out}(v)$, that is a 2-heap in which the root has only one child, of sidetracks (v', v) ordered by $\delta(v', v)$.*

   (b) *For each vertex v that is not on the s-t shortest path, construct $H_T(v)$, that is, a 2-heap of vertices on the s-v shortest path but not on the s-t shortest path ordered by the value δ of the root of the heap made in step 2-(a).*

   (c) *For each vertex v on the s-t shortest path, construct $H_T(v)$, that is, a 2-heap of vertices on the s-v shortest path ordered by the value δ of the root of the heap made in step 2-(a).*

   (d) *Merge $H_{out}(v)$ and $H_T(v)$ to make $H_G(v)$. Then let the length of the edge from node n_1 to node n_2 in this heap be $\delta(n_2) - \delta(n_1)$.*

3. *For each v on the s-t shortest path, make a node N_v.*

4. *For each v in G, make an edge from each node in $H_G(v)$ which represents a sidetrack (u', u) to the root of $H_G(u')$ if u' is not on the s-t shortest path, and define the length of this new edge as the value of the root. Otherwise, make an edge from the node to $N_{u'}$.*

5. *Make a new node for each v in G, and make an edge from this node to the root of $H_G(v)$. Let the length of this edge be δ of the root. Let this new graph be $P(G)$.*

Once the heap was constructed, what we should do next is same as in the original Eppstein algorithm: we only have to search from the root of the virtual heap. Accordingly, we can efficiently enumerate only the alignments in class D_1, and apparently it is output sensitive algorithm once the heap is given. We can easily see that this algorithm can be also used with A* algorithm using the techniques in section 3.3.

We can also extend this algorithm for enumeration of alignments in class D_i $(i \leq c)$ for any c in the same way, though the algorithm becomes much more complicated:

Algorithm 9

1. *Construct the shortest path tree from s to all the other vertices.*
2. *For each vertex v, construct heaps* $H_G^{(i)}(v)$ $(0 \leq i < c$ *for the vertices on the s-t shortest path, and* $0 < i \leq c$ *for the others), a 3-heap of sidetracks* (u', u) *as follows. Let the length from the root of* $H_G^{(i)}(v)$ *to a node n be* $\delta(u, v)$ *if n represents sidetrack* (u, v).
   (a) *For each vertex v, construct* $H_{out}^{(i)}(v)$ $(0 \leq i < c$ *for the vertices on the s-t shortest path, and* $0 < i \leq c$ *for the others), that is a 2-heap in which the root has only one child, of sidetracks* (v', v) *ordered by* $\delta(v', v)$.
   (b) *For each vertex v that is not on the s-t shortest path, construct* $c - 2$ *heaps* $H_T^{(i)}(v)$ $(0 < i < c)$, *all of which are 2-heaps of vertices on the s-v shortest path ordered by the value* δ *of the root of the heap made in step 2-(a).*
   (c) *For each vertex v that is not on the s-t shortest path, construct heap* $H_T^{(c)}(v)$, *that is, 2-heap of vertices on the s-v shortest path but not on the s-t shortest path, ordered by the value* δ *of the root of the heap made in step 2-(a).*
   (d) *For each vertex v on the s-t shortest path, construct* $H_T^{(i)}(v)$ $(0 \leq i)$, *that is, a 2-heap of vertices on the s-v shortest path ordered by the value* δ *of the root of the heap made in step 2-(a).*
   (e) *For each i and v, merge* $H_{out}^{(i)}(v)$ *and* $H_T^{(i)}(v)$ *to make* $H_G^{(i)}(v)$. *Then let the length of the edge from node* n_1 *to node* n_2 *in these heaps be* $\delta(n_2) - \delta(n_1)$.
3. *For each v on the s-t shortest path, make a node* N_v.
4. *For each v in G, make an edge from each node in* $H_G^{(i)}(v)$ *which represents a sidetrack* (u', u) *to the root of* $H_G^{(i)}(u')$ *if* u' *is not on the s-t shortest path, and define the length of this new edge as the value of the root. If* u' *is on the s-t shortest path, make an edge to the root of* $H_G^{(i+1)}(u')$ *if* $i + 1 < c$, *or to* $N_{u'}$ *if* $i + 1 = c$.
5. *Make a new node for each v on the s-t shortest path, and make an edge from this node to the root of* $H_G^{(0)}(v)$. *Let the length of this edge be* δ *of the root. Let this new graph be* $P(G)$.

This algorithm requires $O(c(n + m))$ time to construct the heap structure. In contrast, the branch-and-bound technique we mentioned at first in this section can deal with this kind of problem easily: we only have to remember how many branches from the *s-t* shortest path the parent path has, in searching suboptimal paths in the Eppstein heap structure. Thus, when c is large, which technique is more efficient may depends on cases. But, note that the problem whose c is large is not so important as that whose c is small, *i.e.* 1 or 2.

3.6 Extracting Knowledge from Eppstein Heap

As mentioned by Eppstein (Eppstein 1994), the Eppstein heap has a good feature: some of numerical values for each suboptimal solution can be obtained in $O(1)$ time with some simple pre-process of $O(|E_\Delta|)$ time. In the case of multiple alignment problem, these can be obtained in such an efficient way for example: the number of aligned groups in which all residues are same, the number of gaps, the score computed with another score table, the length of the alignment, and so on. Our algorithm in the section 3.5 which enumerates only alignments in the class D_1 have the same feature too.

Let us consider the case of computing the number of gaps in each obtained suboptimal solution. Let the number of gaps p we want to know be $gaps(p)$. Then the algorithm will be like this:

Algorithm 10

1. *For each vertex v in V, compute the number of gaps contained in the part of the s-v shortest path. This can be done in $O(n)$ time, by the depth first search on the shortest path tree to t. Then let this value be $g(v)$.*
2. *For each sidetrack (u, v), compute following $\delta'(u, v)$:*
$$\delta'(u,v) = g(u) - g(v) \tag{7}$$
3. *When we search in the path heap and obtained a path p (which represents some alignment), we compute the $gaps(p)$ as follows from the value of $gaps(parent(p))$.*
$$gaps(p) = gaps(parent(p)) + \delta'(sidetrack(p)) \tag{8}$$

In the same way, we can get many kinds of values for each solution. But note that there is some values that cannot be obtained in $O(1)$ time. For example, affine gap cost is one of them, though it is very important value.

3.7 Experimental Results

In this section, we examine the efficiency of our approach and investigate the properties of suboptimal alignments through experiments. In the experiment, we used the PAM-250 matrix, and linear gap penalty bx where x is the gap length and b is the minimum value in the PAM-250 matrix, -8. All the experiments are done on Sun Ultra 1 workstation with 128 megabyte memory.

3.7.1 Case with High Similarity

We first did experiments on a group of 8 sequences with high similarity in Table 2. According to it, the average scores per amino pair of these pairwise alignments are about 2.5 to 4. Add to this, the optimal score of multiple alignment of all these 8 sequences is 33129 and its length is 456, thus the average score per amino pair of this alignment is $\frac{33129}{456 \cdot \binom{8}{2}} \approx 2.59$ (Table 3). These are higher than in the experiment in the next section 3.7.2. Figure 7 shows one of the optimal alignments of all the 8 sequences in Table 2. You can easily recognize the high similarity.

As for computing alignments of less than 8 sequences, we could apply the simple A* algorithm. However, for alignment of the 8 sequences, we used the upper bounding technique (enhanced A*) because 128 megabyte memory is not enough for computing with the simple A* algorithm: we used in the experiment the optimal solution as the upper bound to see the best efficiency of this enhanced algorithm. In any case, we used the upper bounding technique after the optimal solution is obtained.

According to Table 3, the DP takes a lot of time compared with the A* algorithm when d is small, but it is negligible when d is large. This table also shows that, the additional searching time required for computing suboptimal solutions is not so much as long as Δ is not much larger than in these experiments: it requires at most twice the time in total as in the case of computing only the optimal alignment in these experiments if $\Delta \leq 40$.

Figure 8, Table 4 and Table 5 show the results of enumerating the suboptimal alignments. Figure 8(a) shows that there are enormous number of suboptimal alignments, and the number increases exponentially as Δ increases. However, in Figure 8(b), we can see the number of suboptimal solutions is dramatically reduced by ignoring alignments in class D_i ($i \geq 2$). The number of the alignments enumerated in this way is only 0.0003% ($d = 4$) to 0.4% ($d = 8$) of all the alignments in case $\Delta = 30$ (see Table 5): it seems difficult to check significance of all the suboptimal alignments at most 10 worse than the optimal, but in our method, we can do it. Accordingly, the enumeration time is also reduced drastically(see Table 5).

Figure 9 shows an example of suboptimal alignments. The score of it is 33139, which is only 10 worse than the optimal. In this figure, $*$ in the first line indicates the regions different from the optimal alignment in Figure 7. This alignment has 5 such regions, which means this alignment is in the class D_5. The existence of such alignments makes enumeration of suboptimal alignments more difficult. It is the reason that our enumeration approach which avoids enumerating such alignments is very efficient.

Observing Figure 8(a), the number of the suboptimal alignments seems to be similar and irrelevant to d. It is an interesting fact, but this comparison is unfair. The number must be compared between the cases which have same value of $\frac{\Delta}{\binom{d}{2}}$: we must consider Δ per amino pair. For example, it is all right to compare the case $\Delta = 28$ ($d = 8$) and the case $\Delta = 10$ ($d = 5$). In this case, the number of suboptimal alignments in the former case is $\frac{7168718}{16112} \approx 444.9$ times as that of the latter case.

According to Table 4, $|E_\Delta|$ and the size of Eppstein heap for this size of Δ is not so large. Thus, the enumeration time in Table 5(b) is small, though it includes times for constructing the heap. In Figure 8(b), the number of suboptimal alignments in class D_1 increases much when $d = 2, 7, 8$ compared with other cases. The reason of this is seen in Table 4. The $|E_\Delta|$ in cases

$d = 2, 7, 8$ is larger than the others: there may be many alignments which have a large region different from the optimal. On the other hand, in Figure 8(a), the number of the alignments in case $d = 4$ is large compared with others, but much of these must be combinations of small number of 'necessary' alignments.

3.7.2 Case with Low Similarity

We next did experiments on 5 globin sequences as in Table 6. According to Table 6, the average scores per amino acid pair of pairwise alignments of them are about 0.2 to 1.3. The score of the optimal multiple alignment of these 5 sequences is 543 and its length is 165, thus the average score per amid acid pair of this alignment is $\frac{543}{165 \cdot \binom{5}{2}} \approx 0.33$, which is lower than the previous case. Figure 10 shows the optimal alignment of the sequences in Table 6. It shows the dissimilarity compared with the case of the previous experiments.

Figure 11 and Table 7 show the result of the experiments. According to Table 7, the searching time by simple A* algorithm is far longer than in the previous experiments for same d, though the length is short. It is because the estimator of the A* algorithm is not so powerful in case with low similarity.

According to Figure 11, the number of alignments in this experiment is also drastically reduced as in the experiments in the section 3.7.1 by ignoring alignments in class D_i ($i \geq 2$): the number of alignments in D_1 is only 0.004% of that of all the suboptimal alignments in case $d = 5$ and $\Delta = 20$.

As mentioned by Zuker (Zuker 1991), the alignments with low score are not always insignificant. In general, if the lengths of sequences to be aligned are short, the size of E_Δ will be small. However, the size of E_Δ is larger than in the previous experiments for same d and Δ. Hence we can conclude that sequences we use in this experiment is not so significant as in the previous experiment. In this way, we can use the size of E_Δ as a factor of the significance of the alignment.

4 Parametric Analysis of Multiple Sequence Alignment

In this section, we describe the techniques for parametric analysis of multiple sequence alignment problem. Furthermore, we also do experiments on actual protein sequences.

4.1 Basic Techniques

In this section, we describe basic methods to check how the optimal solution varies as the parameters such as gap penalties change. The easiest approach for this kind of problem is to change the parameter little by little and check the optimal solution, but we cannot know how little we should change the parameter. Recently the techniques for parametric analysis are developed (Gusfield et al. 1992, Huang et al. 1994, Vingron et al. 1994, Waterman 1994, Waterman 1995,waterman et al. 1992, Zimmer 1997). In those previous works, they also did parametric analysis which deal with more than one parameters, but algorithms for them are not so efficient as the one-parameter case and it will often be nonsense if the parameters are not related each other. Thus we deal with only one parameter at one time in this thesis.

We consider the case in which the score of some alignment A_i is expressed with parameter p as follows:

$$s_i(p) = a(A_i) + b(A_i) \cdot p \tag{9}$$

Gap penalty satisfies this expression for example.

From here, we explain how to divide 1-parameter (1-dimensional) space to regions in which the optimal alignments are always same. Let a_i be $a(A_i)$ and b_i be $b(A_i)$. Let p_i and p_j be the values of the parameter which satisfies $p_i < p_j$ and has different optimal solutions. Let the alignment A_i be the alignment with smallest value of b among the optimal alignments at $p = p_i$ and A_j be the alignment with largest value of b among the optimal alignments at $p = p_j$. Then this two alignments A_i and A_j has the same score at $p = p_{ij} = -\dfrac{a_i - a_j}{b_i - b_j}$. If the optimal score at $p = p_{ij}$ equals to $s_i(p_{ij}) = s_j(p_{ij})$, there are only two regions between p_i and p_j. Otherwise, we can apply the same technique recursively (*i.e.* apply between p_i and p_{ij} and between p_{ij} and p_j) to obtain such division. Figure 12 shows an example of this procedure.

Letting n be the number of regions which we want to obtain, we only have to compute the optimal solutions $2n - 1$ times. Thus we can efficiently do parametric analysis in the case of one parameter.

In the algorithm, the alignments with the largest or smallest value of b among the optimal alignments at some fixed parameter are required. These can be easily obtained by some lexicographical extension of DP (Vingron et al. 1994, Waterman 1994, Waterman 1995, Waterman et al. 1992). This technique can be applied also to the (enhanced) A* algorithm, but the aim of the parametric analysis is to examine all the optimal solutions. Accordingly, it is not preferable to ignore most optimal solutions if there are many. Thus, for this purpose, we enumerate the optimal solutions by the Eppstein algorithm in the section 3.1 and pick up the solutions with the largest or smallest value of b.

4.2 Upper Bounding Technique for Parametric Alignment

As we stated in the section 2.2, the A* algorithm will be more efficient if some upper bounding value for the optimal solution is given (it is called the enhanced A* algorithm). In the parametric alignment problem, $s_i(p_{ij}) = s_j(p_{ij})$ in the section 4.1 is a lower bound of the score of the optimal alignment at $p = p_{ij}$. Thus it can be used as this upper bounding value in computing the optimal alignments at $p = p_{ij}$ with A* algorithm. Note that the enhanced A* algorithm will show best performance if the $s_i(p_{ij}) = s_j(p_{ij})$ is the optimal score at $p = p_{ij}$, which always happens at the final stage of the parametric analysis.

4.3 Experimental Results

In this section, we do parametric analysis on groups of actual protein sequences. We use the famous PAM-250 matrix for score matrix in the experiments in sections 4.3.1 and 4.3.3. Here, all the experiments are also done on Sun Ultra 1 workstation with 128 megabyte memory in sections 4.3.1 and 4.3.3 and on Sun Ultra 1 workstation with 256 megabyte memory in section 4.3.2.

4.3.1 Parametric Gap Penalty

Previous works on parametric analysis mainly dealt with gap penalty, because it is a very important factor of the alignment problem (Gusfield et al. 1992, Huang et al. 1994, Vingron et al. 1994, Waterman 1994, Waterman et al. 1992). But these works dealt only with the 2-alignment problem.

We did parametric analysis of gap penalty using the top d sequences ($d < 7$) in Table 2. In general, the most popular gap penalty is the minimum value in the score matrix, which is -8 in this PAM-250 case. We did parametric analysis for d-sequence alignment ($2 \leq d \leq 6$) with gap penalty between -2 and -16.

Table 8 shows the result of the experiment. In Table 8, the first row of each entry of d shows the boundaries of the regions, but several of the ends are not the boundaries: the ends with $-$ in #Max and #Min entry are not boundaries. The second row shows the numbers of the optimal solutions at the value. The last two rows show the numbers of optimal solutions with largest/smallest value of b in the section 4.1. Thus, these values equal to the number of the optimal solutions between the boundaries.

According to the table, it is observed that the intervals of the regions become smaller (*i.e.* the optimal solution is not stable), as the penalty increases regardless of d. It also shows that there are not so much difference between different d's, which means we can do parametric analysis as easily as in the 2-alignment case. The table also shows that there are more than 1 optimal solution in all cases in the experiments.

Figure 13 shows the number of visited nodes by A* algorithm in computing the all the optimal alignments under various gap penalties. According to this figure, the number of the visited nodes increases drastically as the gap penalty increases especially when gap penalty is larger than -4. This is the reason why we analyzed gap penalty only up to -2.5 or -3.5 when $d \geq 5$: the required space was too large to compute when the gap penalty is around -2.

We think this increase of the search space is caused by the instability of the optimal solution, based on many various experiments. It will be more clear in the next section 4.3.2. In general, it is known that the number of required space for A* algorithm is large if the similarity among the group is low, probably for same origin.

We also did experiments on 10 TNF-α sequences in Table 9, which are very similar to each other. Table 11 shows the result of it. In the experiments, we examined between gap penalties -16 and -2.5, for $d = 2, 4, 6, 8, 10$ sequences. It is between -16 and -4.5 in case of $d = 10$, because of computational difficulty of obtaining the optimal solution with gap penalties near to 0. There are fewer regions than the EF-1α case. The reason of this may be the high similarity among this group.

4.3.2 Parametric Score Matrix

In this section, we deal with the parametric analysis of score matrix. Score matrix is very important parameter and it deeply affects the quality of the output alignment. Thus parametric analysis of it is not only very important but also requires much care to deal with.

There are $\frac{(n+2)(n-1)}{2}$ parameters, where n is the number of characters, to change in the score matrix, thus what we can do is very limited simple analysis. Some of the previous works dealt with this topic. For example, Vingron et al. (Vingron et al. 1994) analyzed the varying solutions as they added some constant (which is the parameter to change) to each element in the score matrix. But it is not so interesting because those constants does not have any meaning at all. We should analyze in more practical and interesting way.

We implemented a program to analyze how the optimal solutions will change as the score matrix changes linearly between two score matrices $(s_{ij}^{(1)})$ and $(s_{ij}^{(2)})$: we used $(p \cdot s_{ij}^{(1)} + (1-p) \cdot s_{ij}^{(2)})$ as the score matrix and considered p as the parameter to be changed. Note that our program can of course deal with the analysis what Vingron et al. did (Vingron et al. 1994).

At first, we did experiments on 8 sequences (fragments) of rhodopsin superfamily shown in table 12. Figure 14 shows the optimal alignment of 5 sequences taken from table 12. But, in the biologists' view, it is not the best alignment (Blanck et al. 1987, Hargrave 1991). According to them, the `K`'s (lycine) which are marked with $*$ at the first line in figure 14 must be aligned.

There must be various methods to cope with this kind of problems, but one of the simplest method is to check the optimal alignment obtained with a score table whose score for `K-K` is large. But, too large score for `K-K` makes irrelevant `K`'s aligned and the alignment will be unnatural. Thus, we should choose as small score as possible in the constraint that we can obtain a biologically good solution. In this way, we can align reasonably those `K`'s.

In this point of view, we did parametric analysis on these sequences, letting the score for `K-K` be the parameter to change. In the experiment, we let the scores other than it are same as those in the PAM-250 score matrix in Table 1. In the PAM-250 score matrix, the score for `K-K` is 5, hence we only examined in the region where its score is larger than 5.

Table 14 shows the result of this experiment. The first row of each entry shows the score for `K-K` which is boundary of regions except for several ends with - in #Max and #Min entries. The second row shows the number of the optimal solutions and the other two rows shows the number of the optimal solutions with largest/smallest b in the section 4.1. In the table, $*$ denotes the smallest score for K-K which induces the optimal alignment whose K's are aligned desirably. Figure 15 shows the alignment of 5 sequences obtained with smallest score for K-K such that K's are aligned desirably. This alignment is biologically much better than that obtained with the ordinary PAM-250 score matrix.

Figure 16 shows how the number of visited nodes by the A* algorithm change as the score for K-K changes. This figure shows the case of 4 and 5 sequences. In this figure, we can easily see that the number is large where the boundary in the parametric analysis exists. Thus, as we mentioned in the section 4.3.1, we conclude that computing the optimal solution will be more difficult if the solution is instable.

It is also interesting that in the case of the alignment of 4 rhodopsin sequences, the number of visited nodes is fixed when the score for K-K is larger than 84. It is because the all possible K's are aligned and thus letting the score larger makes no difference.

PAM-t score table is computed based on the evolutionary permutation probability of amid acids per period proportional to t (Dayhoff et al. 1978), and we can consider various PAM score tables such as PAM-120, PAM-250 and so on. Altschul (Altsuchul 1991) suggested that it is better to align sequences with two or three different PAM scores based on statistical analysis. Thus parametric analysis on this t is useful. Because PAM-t converges into some matrix as t becomes larger, parametric analysis on t can be approximated by the linear parametric analysis between PAM matrices if t is large enough. But note that, because of the same reason, the optimal solutions obtained with different large t's are often same. Note that the exact parametric analysis on this t is difficult and remains as one of future works.

Table 15 shows the result of this kind of experiment using 6 TNF-α sequences in Table 9. The first row of each entry shows the value p of $(p \cdot s_{ij}^{(1)} + (1-p) \cdot s_{ij}^{(1)})$ which is boundary of regions except for several ends with - in #Max and #Min entries, where $(s^{(1)})$ is PAM-250 score matrix and $(s^{(2)})$ is PAM-320 matrix. The second row shows the number of the optimal solutions and the other two rows shows the number of the optimal solutions with largest/smallest b in the section 4.1. Notice that there are no other optimal solutions other than those at both ends between the PAM-250 score matrix and the PAM-320 score matrix.

4.3.3 Parametric Weight Matrix

In this section, we deal with weighted version of the multiple alignment problem as we mentioned in the section 2.1. Computing the optimal solution of this problem by the (enhanced) A* algorithm is rather easy: all we have to do is using $h(v) = \sum_{1 \le i < j \le d} w_{ij} \cdot L^*(u_{ij}, v_{ij})$ as the estimator. Thus we can use same techniques as in previous experiments.

A weight matrix for aligning sequences whose phylogenetic tree is known can be made if divergence between sequences are given (Altsuchul 1989). But what should we do if the divergence are ambiguous? In such case, parametric analysis between reasonable two weight matrices helps. Thus, parametric analysis of weight matrix may helps tuning parameters of a phylogenetic tree.

There are $\dfrac{(d-2)(d+1)}{2}$ parameters to change in the weight matrix, thus what we can do is very limited simple analysis. We implemented a program to analyze how the optimal solutions change as weight matrix changes linearly between two weight matrices, like in the case of the section 4.3.2.

We did experiments between following two weight matrices of (w_{ij}) and $(w_{ij}^{(16,n)})$:

$$w_{ij} = \begin{cases} 0 & i = j \\ 1 & i \neq j \end{cases} \tag{10}$$

$$w_{ij}^{(p,n)} = \begin{cases} p \cdot w_{ij} & i = n \text{ or } j = n \\ w_{ij} & \text{otherwise} \end{cases} \tag{11}$$

In this equation, (w_{ij}) corresponds to the simple sum-of-pairs multiple alignment, and $w_{ij}^{(p,n)}$ increases the importance of nth sequence to p times as the simple sum-of-pairs multiple alignment. If biologically good alignment is discovered in the experiment, we can estimate the importance of the sequence which was increased.

Table 16 shows experiment results using the top 6 EF-1α sequences in Table 2. The first column is the name of the sequence whose importance was increased. The first row of each entry shows the value of p of $w_{ij}^{(p,n)}$ which is boundary of regions except for several ends with - in #Max and #Min entries. The second row shows the number of the optimal solutions and the other two rows shows the number of the optimal solutions with largest/smallest b in the section 4.1.

In this experiment, we notice that the optimal solutions will change even when only $p = 1.33$ in some of the cases (cases of **Tha** and **Ent**). It means we should take more care of the weight matrix. This experiment also show that there are more than 1 optimal solution in all the cases in this experiment. In the experiment, the number of the regions are not too large to deal with (6 to 10 in this experiment). This means this approach is very reasonable to take.

We also did experiments on 5 EF-2 sequences in Table 17, which are very similar to each other and very long. The experiments are done also between weight matrices of (w_{ij}) and $(w_{ij}^{(16,n)})$. In the experiments, we increased importance of one of the sequences. Table 18 shows the results.

5 Conclusions

In multiple alignment problem, it is often said that the optimal solution obtained with only the scoring criteria is not always the biologically best one. Thus, in this thesis, we took two flexible approaches to flexible multiple alignment problem to overcome this problem. One approach is by suboptimal analysis and the other is by parametric analysis.

We first investigated method for enumerating suboptimal alignments of multiple sequences. Previous works related to this subject could not deal with enumeration in practice because of the enormous amount of the suboptimal alignment.

At first, we introduced the Eppstein algorithm (Eppstein 1994). We then proposed efficient upper bounding technique to obtain E_Δ efficiently for multiple alignment problem using A* algorithm. Using this technique, we extended Eppstein algorithm to reduce the memory space.

Furthermore, we classified the suboptimal alignments into classes D_i based on the number of the different regions from the optimal solution. We showed that the suboptimal alignments in classes D_i $(i \geq 2)$ can be easily constructed from the alignments in the class D_0 (the optimal solution) and the class D_1, and suggested that the alignments in the classes D_i $(i \geq 2)$ is not necessary to enumerate. In this point of view, we proposed an algorithm to enumerate only the suboptimal alignments in the class D_1. We also discussed how to obtain information from the Eppstein's heap structure. We showed that some of the important values of the alignments can be obtained in constant time after some preprocess.

Based on these algorithms and observations, we did experiments on actual protein sequences to see the efficiency of our algorithms and to examine the property of the sequences. We did experiments on both groups with high similarity and with low similarity and we saw our method was very dramatical.

Next, we investigated parametric optimization of the multiple alignment problem. The previous works studied this topic only on 2-alignment case, but multiple alignment case is also very important.

We first described the basic technique of parametric analysis, which is a technique to divide the parameter space into regions where the optimal solution is always same. Then we discussed how to use upper bounding technique of the enhanced A* algorithm in this parametric analysis.

In multiple alignment problem, there are three kind of parameters, *i.e.* gap penalty, score matrix, and weight matrix. We did parametric experiments on each of them, and examined the property of the multiple alignment problem in the view of parametric optimization. In the experiments, we showed that the parametric analysis of multiple alignment problem is as practical as that of 2-alignment.

As for future works, there are several problems to do. At first, we should develop more flexible method, such as parametric analysis of suboptimal solutions. Using affine gap cost which makes the problem much difficult also remains as one of future works. Algorithms we proposed are not only for the alignment problem and can also be applied to many other optimization problems, which we should do as a future work too.

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

Altsuchul S. F. 1991. Amid Acid Substitution Matrices from an Information Theoretic Perspective, *J. Mol. Biol.,* 219, 555-565.

Altsuchul S. F., Carroll R. J., and Lipman D. J. 1989. Weights for Data Related by a Tree, *J. Mol. Biol.,* 207, 647-653.

Altschul S. F. and Erickson B. W. 1986. Optimal Sequence Alignment Using Affine Gap Costs, *Bull. Math. Biol.,* 48, 603-616.

Araki S., Goshima M., Mori S., Nakashima H., Tomita S., Akiyama Y., and Kanehisa M. 1993. Application of Parallelized DP and A* Algorithm to Multiple Sequence Alignment, *Proc. Genome Informatics Workshop IV*, 94-102.

Blanck A. and Oesterhelt D. 1987. The Halo-opsin Gene. II. Sequence, Primary Structure of Halorhodopsin and Comparison with Bacteriorhodopsin, *EMBO J.,* 6, 265-273.

Chao K. M. 1994. Computing All Suboptimal Alignments in Linear Space, *Proc. 5th Symposium on Combinatorial Pattern Matching,* 31-42. LNCS 807, Springer-Verlag, New York.

Dayhoff M. O., Schwartz R. M., and Orcutt B. C. 1978. *Atlas of Protein Sequence and Structure* (M. O. Dayhoff ed.), 5, suppl. 3, 345-352, National Biomedical Research Foundation, Washington D. C.

Eppstein D. 1994. Finding the k Shortest Paths, *Proc. 25th IEEE Annual Symposium on Foundation of Computer Science*, 154-165.

Frederickson G. N. 1993. An Optimal Algorithm for Selection in a Min-Heap, *Information and Computation,* 104, 197-214.

Gotoh O. 1982. An Improved Algorithm for Matching Biological Sequences, *J. Mol. Biol.* 162, 705-708.

Gotoh O. 1995. A Weighting System and Algorithm for Aligning Many Phylogenetically Related Sequences, *Comput. Applic. Biosci.,* 11, 543-551.

Gotoh O. 1993. Optimal Alignment between Groups of Sequences and its Application to Multiple Sequence Alignment, *Comput. Applic. Biosci.,* 9, 361-370.

Gotoh O. 1995. *Parity*, 10, No. 12, 13-19, in Japanese.

Gracy J., Chiche L. and Sallantin J. 1993. Improved alignment of weakly homologous protein sequences using structural information, *Protein Engineering* 6, 821-829.

Gupta S. K., Kececioglu J. D., and Schaffer A. A. 1995. Improving the Practical Space and Time Efficiency of the Shortest-paths Approach to Sum-of-pairs Multiple Sequence Alignment, *J. Comput. Biol.*, 2, No. 3, 459-472.

Gusfield D., Bakasubramanian K., and Naor D. 1992. Parametric Optimization of Sequence Alignment, *Proc. 3rd ACM-SIAM Annual Symposium on Discrete Algorithms,* 432-439.

Hargrave P. A. 1991. *Current Opinion on Structural Biology,* 1, 575-581.

Hsu M. 1994. A Study on the Shortest-Path Algorithm for Route Navigation, *A Master's Thesis, Department of Information Science, University of Tokyo.*

Huang X., Pevzner P. A., and Miller W. 1994. Parametric Recomputing in Alignment Graphs, *Proc. 5th Annual Symposium on Combinatorial Pattern Matching,* 87-101, LNCS 807, Springer-Verlag, New York.

T. Ikeda, 1995. Applications of the A* Algorithm to Better Routes Finding and Multiple Sequence Alignment, *A Master's Thesis, Department of Information Science, University of Tokyo.*

Ikeda T., Hsu M., Imai H., Nishimura S., Shimoura H., Hashimoto T., Temmoku K., and Mitoh K. 1994. A Fast Algorithm for Finding Better Routes by AI Search Techniques, *Proc. IEEE International Confrence on Vehicle Navigation and Information System*, 90-99.

Ikeda T. and Imai H. 1994. Fast A* Algorithms for Multiple Sequence Alignment, *Proc. Genome Informatics Workshop V*, 90-99.

Imai H. and Ikeda T., 1995. k-group Multiple Alignment Based on A* Search, *Proc. Genome Informatics Workshop VI,* 9-18.

Naor D. and Brutlag D., 1993. On suboptimal alignments of biological sequences, *Proc. 4th Symposium on Combinatorial Pattern Matching,* 179-196. LNCS 684, Springer-Verlag, New York.

Saqi M. A. and Sternberg M. J. 1991. A simple method to generate non-trivial alternate alignments of protein sequences, *J. Mol. Biol.,* 219, 727-732.

Saqi M. A., Bates P. A., and Sternberg M. J. E. 1992. Towards an automatic method of predicting protein structure by homology: an evaluation of suboptimal sequence alignments, *Protein Engineering,* 5, 305-311.

Shibayama G. and Imai H. 1993. Finding K-best Alignment of Multiple Sequences, *Proc. Genome Informatics Workshop IV,* 120-129.

Shibuya T., Ikeda T., Imai H., Nishimura S., Shimoura H., and Tenmoku K. 1995. Finding a Realistic Detour by AI Search Techniques, *Proc. 2nd Intelligent Tranportation Systems,* 4, 2037-2044.

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

Shibuya T. and Imai H. 1997. New Flexible Approaches for Multiple Sequence Alignment, *Proc. 1st Annual International Conference on Computational Molecular Biology*, 267-276.

Shibuya T. and Imai H. 1996. Parametric Alignment of Multiple Biological Sequences, *Proc. Genome Informatics Workshop 1996,* 41-50.

Shibuya T., Imai H., Nishimura S., Shimoura H., and Tenmoku K. 1996. Detour Queries in Geographical Databases for Navigation and Related Algorithm Animations, *Proc. International Symposium on Cooperative Database Systems for Advanced Applications.*

Shirai Y. and Tsuji J. 1982. *Artificial Intelligence,* Iwanami Course: Information Science, 22, Iwanami, Tokyo, in Japanese.

Spouge J. L. 1989. Speeding Up Dynamic Programming Algorithms for Finding Optimal Lattice Paths, *SIAM J. Appl. Math.,* 49, 1552-1566.

Vingron M. and Waterman M. S. 1994. Sequence Alignment and Penalty Choices: Review of Concepts, Case Studies and Implications, *J. Mol. Biol.,* 235, 1-12.

Waterman M. S. 1995. Introduction to Computational Biology: Maps, Sequences and Genomes, Chapman & Hall.

Waterman M. S. 1994. Parametric and Ensemble Sequence Alignment Algorithms, *Bull. Math. Biol.,* 56, 743-767.

Waterman M. S. and Eggert M. 1987. A New Algorithm for Best Subsequence Alignments with Application to tRNA-rRNA Comparisons, *J. Mol. Biol.,* 197, 723-728.

Waterman M. S., Eggert M., and Lander E. 1992. Parametric Sequence Comparisons, *Proc. Natural Academy of Science, USA,* 89, 6090-6093.

Zimmer R. and Lengauer T. 1997. Fast and Numerically Stable Parametric Alignment of Biosequences, *Proc. 1st Annual International Conference on Computational Molecular Biology*, 344-353.

Zuker M. 1991. Suboptimal sequence alignment in molecular biology. Alignment with error analysis, *J. Mol. Biol.,* 221, 403-420.

Table 1: PAM-250 score matrix

C	12																			
S	0	2																		
T	−2	1	3																	
P	−3	1	0	6																
A	−2	1	1	1	2															
G	−3	1	0	−1	1	5														
N	−4	1	0	−1	0	0	2													
D	−5	0	0	−1	0	1	2	4												
E	−5	0	0	−1	0	0	1	3	4											
Q	−5	−1	−1	0	0	−1	1	2	2	4										
H	−3	−1	−1	0	−1	−2	2	1	1	3	6									
R	−4	0	−1	0	−2	−3	0	−1	−1	1	2	6								
K	−5	0	0	−1	−1	−2	1	0	0	1	0	3	5							
M	−5	−2	−1	−2	−1	−3	−2	−3	−2	−1	−2	0	0	6						
I	−2	−1	0	−2	−1	−3	−2	−2	−2	−2	−2	−2	−2	2	5					
L	−6	−3	−2	−3	−2	−4	−3	−4	−3	−2	−2	−3	−3	4	2	6				
V	−2	−1	0	−1	0	−1	−2	−2	−2	−2	−2	−2	−2	2	4	2	4			
F	−4	−3	−3	−5	−4	−5	−4	−6	−5	−5	−2	−4	−5	0	1	2	−1	9		
Y	0	−3	−3	−5	−3	−5	−2	−4	−4	−4	0	−4	−4	−2	−1	−1	−2	7	10	
W	−8	−2	−5	−6	−6	−7	−4	−7	−7	−5	−3	2	−3	−4	−5	−2	−6	0	0	17
	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W

Table 2: Sequences of EF-TU and EF-1α to be aligned and their scores of pairwise sequence alignments. We use the top d sequences in this table in the experiments.

Sequences			Pairwise Scores						
Species	Protein	Length	Met	Tha	Thc	Sul	Ent	Pla	Sty
Halobacterium marismortui (Hal)	EF-TU	421	1329	1314	1221	1109	1099	1000	971
Methanococcus vannielii (Met)	EF-TU	428		1336	1247	1150	1176	1087	1045
Thermoplasma acidophilum (Tha)	EF-1α	424			1311	1261	1233	1063	1072
Thermococcus celer (Thc)	EF-1α	428				1132	1130	1049	991
Sulfolobus acidocaldarius (Sul)	EF-1α	435					1192	1131	1099
Entamoeba histolytica (Ent)	EF-1α	430						1584	1551
Plasmodium falciparum (Pla)	EF-1α	443							1636
Stylonychia lemnae (Sty)	EF-1α	446							

Table 3: Searching time (sec) by the A* algorithm in the experiment on the d sequences of EF-TU and EF-1α. In case $d = 8$, the enhanced A* utilizing the optimal score is used. Note that only DP is used in case $d = 2$.

	$d=2$	$d=3$	$d=4$	$d=5$	$d=6$	$d=7$	$d=8$
best score	1329	3970	7709	12314	18101	24912	33129
Pre-process DP	0.32	1.00	4.30	7.23	11.1	16.0	20.5
Search (optimal)	-	0.18	0.52	3.35	19.6	426	5427
Search ($\Delta = 10$)	-	0.18	0.60	3.63	20.9	439	5686
Search ($\Delta = 20$)	-	0.22	0.73	4.17	23.1	462	6735
Search ($\Delta = 30$)	-	0.27	0.93	5.00	26.9	498	8027
Search ($\Delta = 40$)	-	0.33	1.23	6.22	31.5	552	9623

Table 4: Size of E_Δ and Eppstein's heap structure in the experiments on d sequences of EF-TU and EF-1α. The heap size in the table does not include the number of nodes made in step 4 of Eppstein algorithm, which is same as $|E_\Delta|$.

Δ		$d=2$	$d=3$	$d=4$	$d=5$	$d=6$	$d=7$	$d=8$
10	$\lvert E_{10} \rvert$	503	485	513	553	534	579	540
	heap size	437	277	411	503	454	674	404
20	$\lvert E_{20} \rvert$	1101	595	609	689	691	799	784
	heap size	7184	1010	1266	1701	1750	2604	2672
30	$\lvert E_{30} \rvert$	1447	946	817	901	864	1246	1316
	heap size	12983	4949	3417	3997	3594	7552	8539
40	$\lvert E_{40} \rvert$	2011	2528	1170	1249	1156	1973	2254
	heap size	17934	25861	8648	10036	7785	18407	23973

Table 5: Enumerating time (sec) when $\Delta = 30$ in the experiment on the d sequences EF-TU and EF-1α. (a) is the case enumerating all the suboptimal alignments, and (b) is the case enumerating alignments in class D_0 (optimal) and D_1. The time of constructing the Eppstein's heap structure is included in the time below.

		$d=2$	$d=3$	$d=4$	$d=5$	$d=6$	$d=7$	$d=8$
(a)	#alignments	38047513	8804702	327522816	85923864	20689104	49633652	13857237
	time (sec)	152.87	54.98	1830.18	510.63	124.55	292.50	109.00
(b)	#alignments	6968	1695	1117	2176	1659	41791	60589
	time (sec)	0.23	0.13	0.32	0.95	2.10	9.83	29.62

Table 6: Globin sequences to be aligned and their scores of pairwise sequence alignments.

globin		Length	Apl	Bus	Ct7	Ct3
Lumbricus terrestris - AIII	(Lum)	157	29	15	35	41
Aplysia limacina	(Apl)	146		126	177	140
Busycon canaliculatum	(Bus)	147			111	64
Chironomus thummi thummi - VIIA	(Ct7)	145				191
Chironomus thummi thummi - IIIa	(Ct3)	151				

Table 7: The best score, searching time (sec) by simple A* algorithm and the size of E_Δ in the experiment on the d globin sequences.

	$d = 2$	$d = 3$	$d = 4$	$d = 5$
best score	29	103	354	543
Pre-Process DP	0.05	0.27	0.52	0.83
Search (optimal)	-	0.73	7.40	837
Search ($\Delta = 10$)	-	0.85	8.13	865
Search ($\Delta = 20$)	-	0.93	9.43	909
Search ($\Delta = 30$)	-	1.22	11.22	964
Search ($\Delta = 40$)	-	1.63	14.13	1080
$\|E_{10}\|$	357	508	380	554
$\|E_{20}\|$	776	1184	866	1159
$\|E_{30}\|$	1149	2403	1575	2448
$\|E_{40}\|$	1544	3839	2669	4569

Table 8: The result of the experiment on parametric gap penalty using EF-1α sequences.

$d = 2$	Gap penalty	-16	-5	-3	-2.5	-2				
	#Solutions	4	12	24	192	576				
	#Max	-	8	16	8	32				
	#Min	-	4	8	16	8				
$d = 3$	Gap penalty	-16	-3.5	-3	-2.75	-2.5	-2.2	-2		
	#Solutions	8	16	24	32	72	48	256		
	#Max	-	8	16	16	16	32	96		
	#Min	-	8	8	16	16	16	32		
$d = 4$	Gap penalty	-16	-8	-3.83	-3.5	-2.5	-2.33	-2.25	-2	
	#Solutions	16	32	32	32	32	48	160	4608	
	#Max	-	16	16	16	16	32	128	384	
	#Min	-	16	16	16	16	16	32	128	
$d = 5$	Gap penalty	-16	-7.5	-4	-3.38	-3.17	-3	-2.88	-2.75	-2.5
	#Solutions	2	4	4	4	4	4	12	8	24
	#Max	-	2	2	2	2	2	4	4	4
	#Min	-	2	2	2	2	2	2	4	4
$d = 6$	Gap penalty	-16	-6.5	-4.5	-4	-3.5				
	#Solutions	4	16	8	8	4				
	#Max	-	4	4	4	-				
	#Min	-	4	4	4	-				

Table 9: TNF-α sequences used in the experiment

Species	Protein	Length
Homo sapiens (HSp)	tumor necrosis factor α (TNF-α) precursor	233
Mus musculus (MM)	tumor necrosis factor α (TNF-α) precursor	235
Sus scrofa (SS)	tumor necrosis factor α (TNF-α) precursor	232
Ovis orientalis aries (OOAp)	tumor necrosis factor α (TNF-α) precursor	234
Bos primigenius taurus (BPT)	tumor necrosis factor α (TNF-α) inhibitor	233
Equus caballus (EC)	tumor necrosis factor α (TNF-α) precursor	234
Oryctolagus cuniculus (OC)	tumor necrosis factor α (TNF-α) precursor	234
Rattus norvegicus (RN)	tumor necrosis factor α (TNF-α) precursor	235
Homo sapiens (HSi)	tumor necrosis factor α (TNF-α) inhibitor	233
Ovis orientalis aries (OOAi)	tumor necrosis factor α (TNF-α) inhibitor	233

Table 10: Scores of pairwise alignments of TNF-α sequences

	MM	SS	OOAp	BPT	EC	OC	RN	HSi	OOAi
HSp	955	990	924	935	1013	949	943	1041	932
MM		919	872	872	945	952	1124	952	866
SS			980	990	982	906	912	974	988
OOAp				1059	916	840	856	915	1145
BPT					933	865	854	926	1067
EC						962	931	1013	906
OC							946	961	845
RN								940	850
HSi									922
OOAi									

Table 11: The result of the experiment on parametric gap penalty using TNF-α.

$d = 2$	Gap penalty	-16	-2.5					
	#Solutions	6	18					
	# Max	-	12					
	# Min	-	6					
$d = 4$	Gap penalty	-16	-6.67	-3.75	-2.88	-2.67	-2.5	
	#Solutions	12	24	24	24	24	36	
	# Max	-	12	12	12	12	24	
	# Min	-	12	12	12	12	12	
$d = 6$	Gap penalty	-16	-4.95	-3.3	-2.75	-2.5		
	#Solutions	6	12	18	24	36		
	# Max	-	6	12	12	24		
	# Min	-	6	6	12	12		
$d = 8$	Gap penalty	-16	-5.57	-4.5	-3.36	-2.92	-2.75	-2.5
	#Solutions	1	2	2	3	4	4	2
	# Max	-	1	1	2	2	2	-
	# Min	-	1	1	1	2	2	-
$d = 10$	Gap penalty	-16	-5.28	-4.5				
	#Solutions	1	2	1				
	# Max	-	1	-				
	# Min	-	1	-				

Table 12: Sequences of Rhodopsin Superfamily used in the experiments

Species		Protein	Length
Halobacterium sp.	(Hal)	Halorhodopsin precursor	39
Homo Sapiens	(HSg)	Green-sensitive opsin	48
Homo Sapiens	(HSr)	Red-sensitive opsin	48
Gallus gallus	(GGd)	Rhodopsin	55
Homo Sapiens	(HSb)	Blue-sensitive opsin	54
Homo Sapiens	(HSd)	Rhodopsin	55
Drosophila melanogaster	(DM3)	Opsin RH3	53
Drosophila melanogaster	(DM4)	Opsin RH4	53

Table 13: Scores of pairwise sequences of Rhodopsin Superfamily

	HSg	HSr	GGd	HSb	HSd	DM3	DM4
Hal	-35	-32	-79	-61	-83	-31	-26
HSg		256	108	100	111	2	-10
HSr			107	100	110	7	-5
GGd				147	275	30	18
HSb					150	34	35
HSd						33	21
DM3							277

Table 14: The result of the experiment on parametric score matrix using rhodopsin sequences. `*` denotes the smallest score for `K-K` which induces the optimal alignment whose `K`'s are aligned desirably.

$d = 2$	Score of `K-K`	5	53*	128		
	#Solutions	1	6	5		
	#Max	-	1	-		
	#Min	-	5	-		
$d = 3$	Score of `K-K`	5	55.5	128		
	#Solutions	2	7	5		
	#Max	-	2	-		
	#Min	-	5	-		
$d = 4$	Score of `K-K`	5	38	58.5*	128	
	#Solutions	4	36	48	12	
	#Max	-	32	12	-	
	#Min	-	4	32	-	
$d = 5$	Score of `K-K`	5	24	89*	124	128
	#Solutions	24	32	20	28	16
	#Max	-	8	12	16	-
	#Min	-	24	8	12	-
$d = 6$	Score of `K-K`	5	20	109.5*	128	
	#Solutions	24	28	6	2	
	#Max	-	4	2	-	
	#Min	-	24	4	-	
$d = 7$	Score of `K-K`	5	16	33.33	64	
	#Solutions	4	8	6	2	
	#Max	-	4	2	-	
	#Min	-	4	4	-	
$d = 8$	Score of `K-K`	5	16.5	32		
	#Solutions	8	16	8		
	#Max	-	8	-		
	#Min	-	8	-		

Table 15: The result of the linear parametric analysis between PAM-250 and PAM-320 using 6 TNF-α sequences

$d = 6$	p	0	0.5	1
	#Solutions	6	12	9
	#Max	-	6	3
	#Min	-	6	6

Table 16: The result of the experiment on parametric weight matrix using EF-1α sequences.

Hal	Weight	1	1.33	3	3.17	4.5	4.75	5	6.57	16		
	#Solutions	4	16	16	12	8	8	48	16	8		
	#Max	-	4	8	4	4	4	8	8	-		
	#Min	-	4	4	8	4	4	4	8	-		
Met	Weight	1	2.25	3	4.2	4.4	4.5	12	14.5	16		
	#Solutions	4	8	12	16	16	16	16	16	24		
	#Max	-	4	8	8	8	8	8	8	16		
	#Min	-	4	4	8	8	8	8	8	8		
Tha	Weight	1	2.33	3	5.5	7.67	8	16				
	#Solutions	4	8	8	8	8	16	4				
	#Max	-	4	4	4	4	4	-				
	#Min	-	4	4	4	4	4	-				
Thc	Weight	1	1.8	3	4	5	5.33	6	10.33	12	14.44	16
	#Solutions	4	8	8	12	12	8	8	8	8	8	12
	#Max	-	4	4	2	4	4	4	4	4	4	8
	#Min	-	4	4	4	2	4	4	4	4	4	4
Sul	Weight	1	2	3	3.75	5	6	6.43	8	10.25	14	16
	#Solutions	4	12	8	8	12	16	16	16	16	16	8
	#Max	-	4	4	4	8	8	8	8	8	8	-
	#Min	-	4	4	4	4	8	8	8	8	8	-
Ent	Weight	1	1.33	1.5	3	3.66	5	6	10.33	13	16	
	#Solutions	4	8	8	8	8	8	8	12	24	16	
	#Max	-	4	4	4	4	4	4	8	16	-	
	#Min	-	4	4	4	4	4	4	4	8	-	

Table 17: EF-2 (translation elongation factor eEF-2) sequences to be aligned and their pairwise scores

Sequences			Pairwise Scores			
Species	Protein	Length	CGS	DM	DD	SC
Homo sapiens (HS)	eEF-2	858	4216	3409	2463	2941
Cricetinae gen. sp. (CGS)	eEF-2	858		3392	2456	2931
Drosophila melanogaster (DM)	eEF-2	844			2416	2959
Dictyostelium discoideum (DD)	eEF-2	830				2446
Saccharomyces cerevisiae (SC)	eEF-2	842				

Table 18: The result of the experiment on parametric weight matrix using EF-2.

HS	Weight	1	1.33	2	3	3.14	6	9	16
	#Solutions	16	32	32	16	16	16	24	24
	#Max	-	16	8	8	8	8	16	-
	#Min	-	16	16	8	8	8	8	-

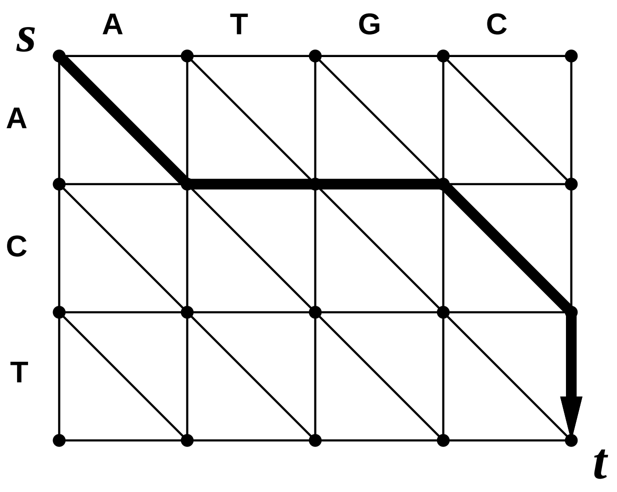


Figure 1: The graph for the alignment of two sequences `ATGC` and `ACT`. The s-t path in the bald line represents the alignment of `ATGC-` and `A--CT`

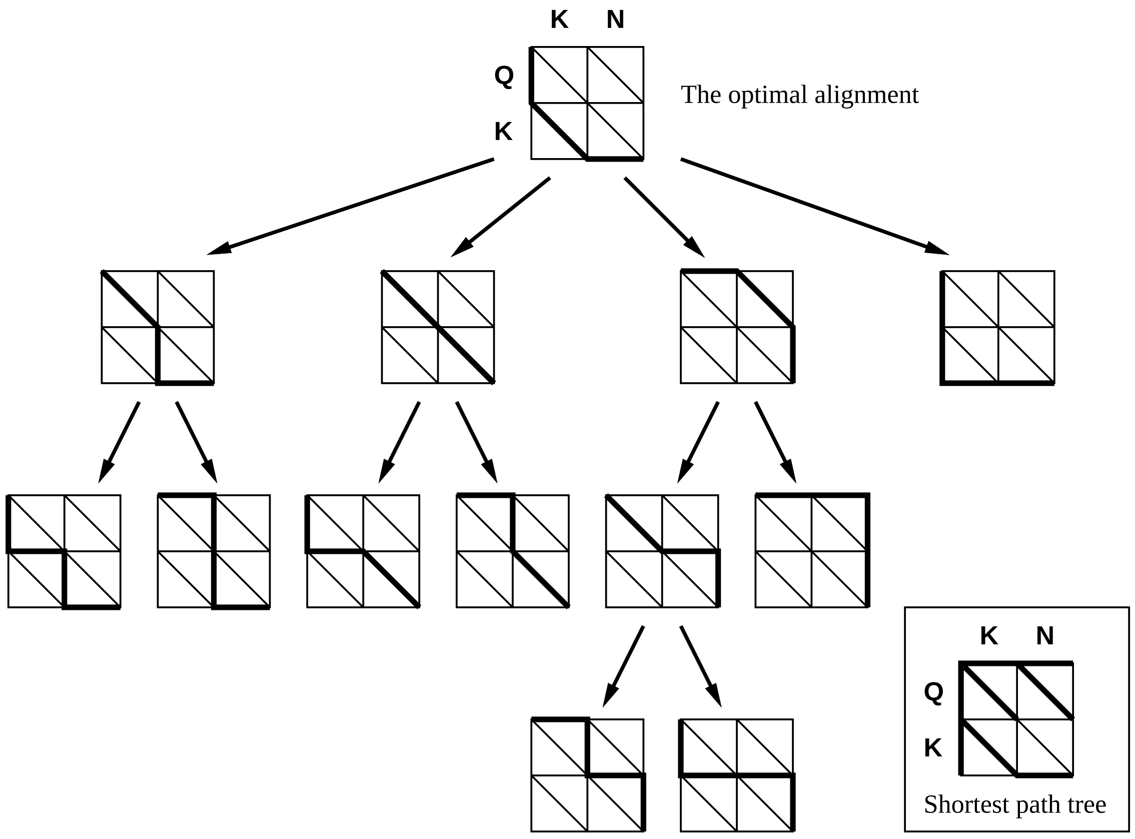


Figure 2: The path heap of the alignment graph of two sequences `KN` and `QK`

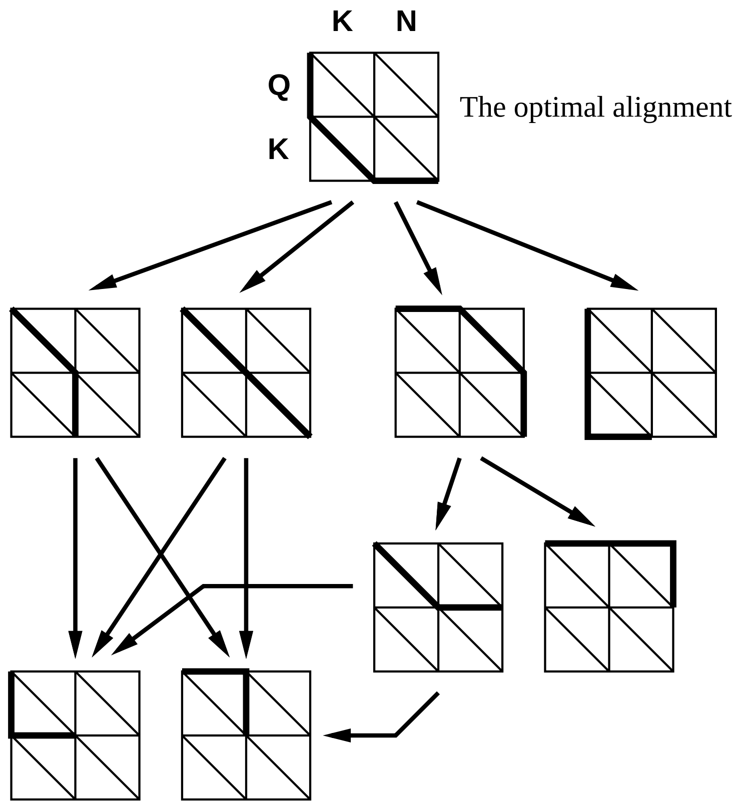


Figure 3: The compact path heap of the alignment graph of two sequences `KN` and `QK`. Some of the nodes are shared, and the number of nodes in it is reduced.

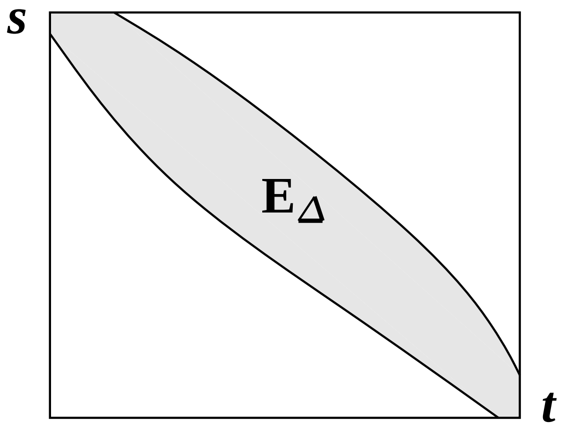


Figure 4: E_Δ is a set of vertices which are used by the s-t paths whose lengths are at most Δ longer than the shortest path

```
REAFSQAIWRATFAQVPESRSLFKR==
ADFLV-ALF-EKFPDSANFFADFKGKS
KNG-S-LLFGLLFKTYPDTKKHFKHFD
LAAVF-TAYPDIQARFPQFAGK-DVAS
GSGVE-ILY-FFLNKFPGNFPMFKKLG
```

(a) The optimal alignment

```
REA | FSQ | AIWRATFAQVPESRSLFKR==
ADF | LV- | ALF-EKFPDSANFFADFKGKS
KNG | -S- | LLFGLLFKTYPDTKKHFKHFD
LAA | -VF | TAYPDIQARFPQFAGK-DVAS
GSG | -VE | ILY-FFLNKFPGNFPMFKKLG
```

(b) A suboptimal alignment

```
REAFSQAIWRATFAQVPESRS | LF | KR==
ADFLV-ALF-EKFPDSANFFA | DF | KGKS
KNG-S-LLFGLLFKTYPDTKK | HF | KHFD
LAAVF-TAYPDIQARFPQFAG | -K | DVAS
GSGVE-ILY-FFLNKFPGNFP | MF | KKLG
```

(c) Another suboptimal alignment

```
REA | FSQ | AIWRATFAQVPESRS | LF | KR==
ADF | LV- | ALF-EKFPDSANFFA | DF | KGKS
KNG | -S- | LLFGLLFKTYPDTKK | HF | KHFD
LAA | -VF | TAYPDIQARFPQFAG | -K | DVAS
GSG | -VE | ILY-FFLNKFPGNFP | MF | KKLG
```

(d) Unnecessary alignment to check

Figure 5: Examples of suboptimal alignments of multiple protein sequences

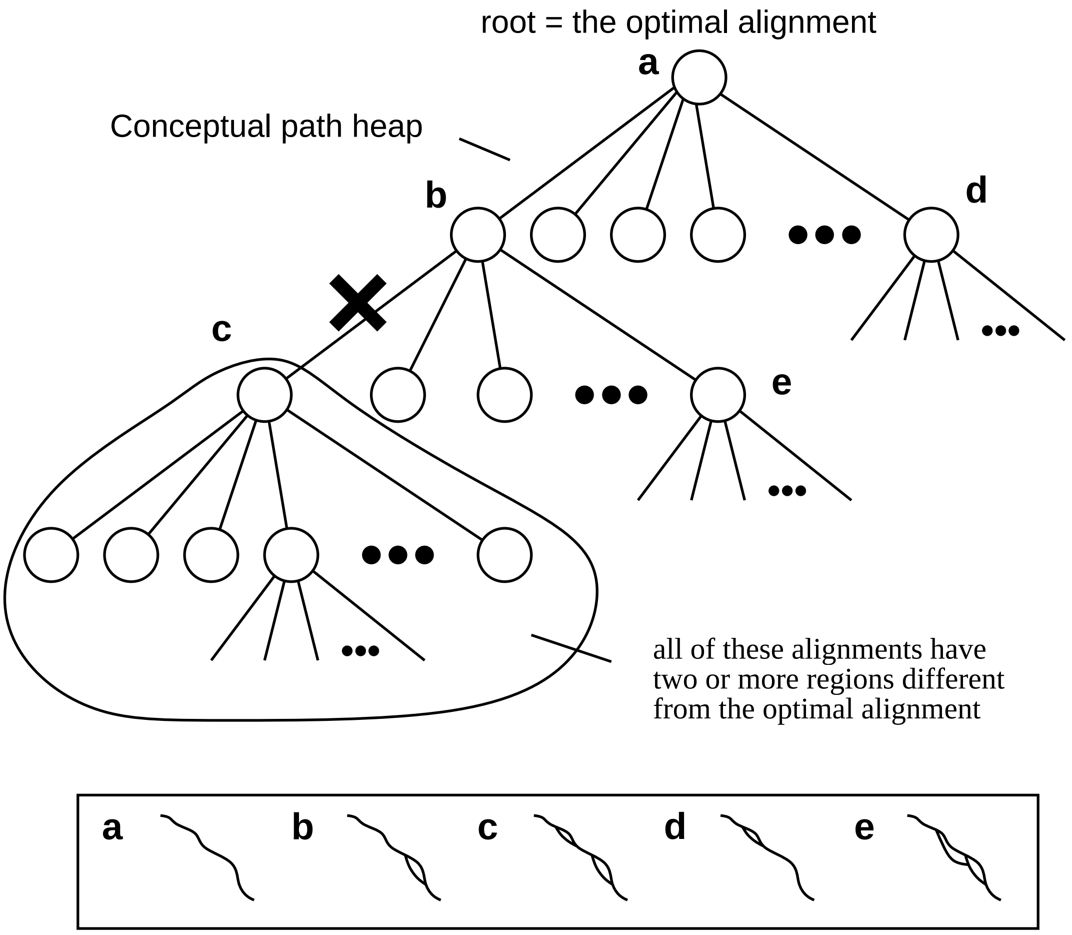
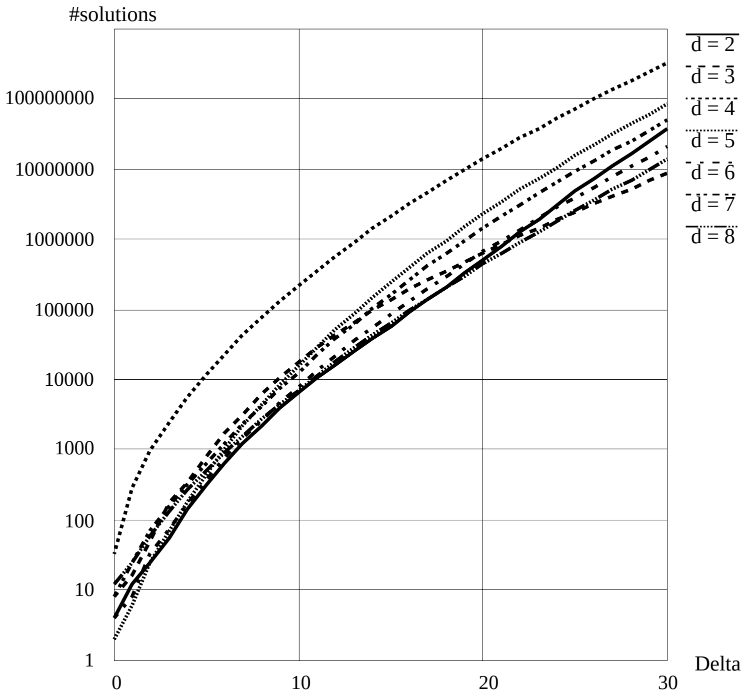

root = the optimal alignment
a
Conceptual path heap
b
d
c
e
all of these alignments have two or more regions different from the optimal alignment
a
b
c
d
e

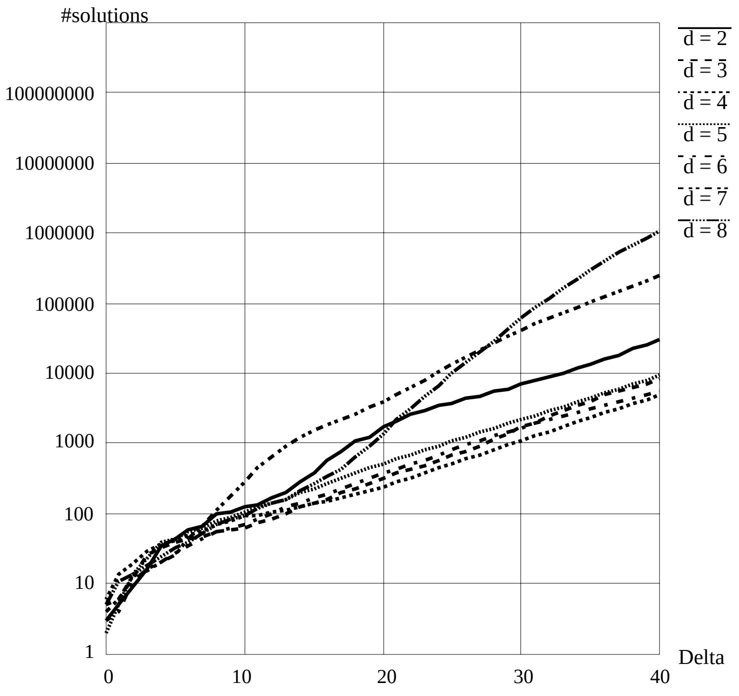
Figure 6: An example of a conceptual path heap. We can easily construct a heap which does not contain unnecessary alignments such as **c** and its all descendants.

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

    81                                                                           160
Hal STDTYDFTIVDCPGHRDFVKNMITGASQADNAVLVVAA-D---D-GV-QP-QTQEHVFLARTLGIGELIVAVNKMD-L-V
Met PTAKYEVTIVDCPGHRDFIKNMITGASQADAAVLVVNVDDA--KSGI-QP-QTREHVFLIRTLGVRQLAVAVNKMD-T-V
Tha ETDKYYFTLIDAPGHRDFVKNMITGTSQADAAILVISARDG--E-GV-ME-QTREHAFLARTLGVPQMVVAINKMDATSP
Thc ETPHRYITIIDAPGHRDFVKNMITGASQADAAVLVVAV-T---D-GV-MP-QTKEHAFLARTLGINNILVAVNKMD-M-V
Sul ETRKYFFTVIDAPGHRDFVKNMITGASQADAAILVVSAKKGEYEAGMSAEGQTREHIILSKTMGINQVIVAINKMDLADT
Ent ETSKYYFTIIDAPGHRDFIKNMITGTSQADVAILIVAAGTGEFEAGISKNGQTREHILLSYTLGVKQMIVGVNKMD-A-I
Pla ETPRYFFTVIDAPGHKDFIKNMITGTSQADVALLVVPADVGGFDGAFSKEGQTKEHVLLAFTLGVKQIVVGVNKMD-T-V
Sty ETAKSVFTIIDAPGHRDFIKNMITGTSQADAAILIIASGQGEFEAGISKEGQTREHALLAFTMGVKQMIVAVNKMDDKSV

    161                                                                          240
Hal DYGESEYKQVVEEV-KDLLTQVRFDSENAKFIPVSAFEGDNIAEESEHTGWYDGEILLEALNELPAPEPPTDAPLRLPIQ
Met NFSEADYNELKKMIGDQLLKMIGFNPEQINFVPVASLHGDNVFKKSERNPWYKGPTIAEVIDGFQPPEKPTNLPLRLPIQ
Tha PYSEKRYNEVKADA-EKLLRSIGFK-D-ISFVPISGYKGDNVTKPSPNMPWYKGPTLLQALDAFKVPEKPINKPLRIPVE
Thc NYDEKKFKAVAEQV-KKLLMMLGYK-N-FPIIPISAWEGDNVVKKSDKMPWYNGPTLIEALDQMPEPPKPTDKPLRIPIQ
Sul PYDEKRFKEIVDTV-SKFMKSFGFDMNKVKFVPVVAPDGDNVTHKSTKMPWYNGPTLEELLDQLEIPPKPVDKPLRIPIQ
Ent QYKQERYEEIKKEI-SAFLKKTGYNPDKIPFVPISGFQGDNMIEPSTNMPWYKGPTLIGALDSVTPPERPVDKPLRLPLQ
Pla KYSEDRYEEIKKEV-KDYLKKVGYQADKVDFIPISGFEGDNLIEKSDKTPWYKGRTLIEALDTMQPPKRPYDKPLRIPLQ
Sty NWDQGRFIEIKKEL-SDYLKKIWLQPRQDPFIPISGWHGDNMLEKSPNMPWFTGSTLIDALDALDQPKRPKDKPLRLPLQ

    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
Sty DVYKIGGIGTVPVGRVETGLLKPGMVLTFAPMN-I----T-TECKSVEMHHESLTEAEPGDNVGFTVKNLSVKDLRRGYV

    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
Sty ASDSKNDPAKD-TTN-FLAQVIVLNHPGQIQKGYAPVLDCHTAHIACKFDEIESKVDRRSGKVLE-EEPKFIKSGEAALV

    401                                                  456
Hal TVRPQKPLSIEPSSEIPELGSFAIRDMGQTIAAGKV--L---G---V-NE-----R
Met KLIPTKPMVIESVKEIPQLGRFAIRDMGMTVAAGMA--I---Q---VTAKN----K
Tha KVIPDKPLVIEKVSEIPQLGRFAVLDMGQTVAAGQC--I---D---L-EK-----R
Thc VLRPTKPMVIEPVKEIPQMGRFAIRDMGQTVAAGMV--I---S---I-QKA----E
Sul KFKPIKELVAEKFREFPALGRFAMRDMGKTVGVGVI--I---D---VKPRKVE-VK
Ent KIVPTKPLCVEEFAKFPPLGRFAVRDMKQTVAVGVV--K---A---V-T------P
Pla SLEPKKPMVVETFTEYPPLGRFAIRDMRQTIAVGIINQLKRKNLGAVTAKAPA-KK
Sty RMVPQKPMCVEAFNQYPPLGRFAVRDMKQTVAVGVIKEVVKKEQKGMVTKAAQKKK
```

Figure 7: The optimal alignment of the 8 EF-TU and EF-1α sequences



(a) enumerating all the alignments



(b) enumerating alignments in classes D_0 and D_1

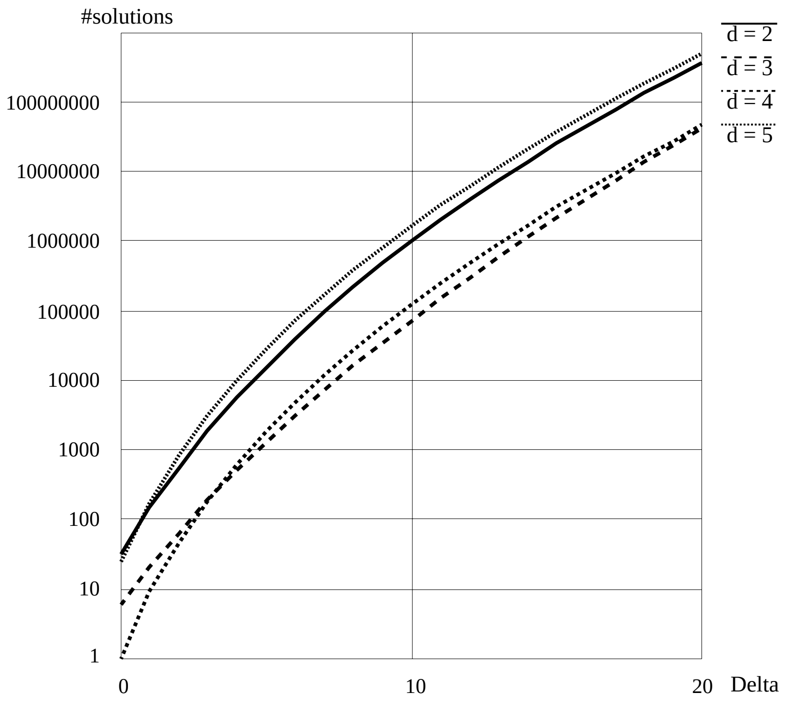
Figure 8: Number of the suboptimal alignments of d sequences of EF-TU and EF-1α whose scores are at most Δ worse than the optimal. (a) is the case enumerating all the alignments. ($0 \leq \Delta \leq 30$) (b) is the case enumerating alignments in classes D_0 and D_1. ($0 \leq \Delta \leq 40$)

```
    1 ****                                                                        80
Hal MSDEQ-HQNLAIIGHVDHGKSTLVGRLLYETGSVPEHVIEQHKEEAEEKGKGGFEFAYVMDNLAEERERGVTIDIAHQEF
Met MAKTKPILNVAFIGHVDAGKSTTVGRLLLDGGAIDPQLIVRLRKEAEEKGKAGFEFAYVMDGLKEERERGVTIDVAHKKF
Tha MASQKPHLNLITIGHVDHGKSTLVGRLLYEHGEIPAHIIEEYRKEAEQKGKATFEFAWVMDRFKEERERGVTIDLAHRKF
Thc MAKEKPHINIVFIGHVDHGKSTTIGRLLFDTANIPENIIKKFE-EMGEKGK-SFKFAWVMDRLKEERERGITIDVAHTKF
Sul MS-QKPHLNLIVIGHVDHGKSTLIGRLLMDRGFIDEKTVKEAEEAAKKLGKDSEKYAFLMDRLKEERERGVTINLSFMRF
Ent MPKEKTHINIVVIGHVDSGKSTTTGHLIYKCGGIDQRTIEKFEKESAEMGKGSFKYAWVLDNLKAERERGITIDISLWKF
Pla MGKEKTHINLVVIGHVDSGKSTTTGHIIYKLGGIDRRTIEKFEKESAEMGKGSFKYAWVLDKLKAERERGITIDIALWKF
Sty MPKEKNHLNLVVIGHVDSGKSTSTGHLIYKCGGIDKRTIEKFEKEAAEMGKGSFKYAWVLDKLKAERERGITIDIALWNF

    81                                    **                                     **
Hal STDTYDFTIVDCPGHRDFVKNMITGASQADNAVLVVAAD----D-GV-QP-QTQEHVFLARTLGIGELIVAVNKMD--LV
Met PTAKYEVTIVDCPGHRDFIKNMITGASQADAAVLVVNVDDA--KSGI-QP-QTREHVFLIRTLGVRQLAVAVNKMD--TV
Tha ETDKYYFTLIDAPGHRDFVKNMITGTSQADAAILVISARDG--E-GV-ME-QTREHAFLARTLGVPQMVVAINKMDATSP
Thc ETPHRYITIIDAPGHRDFVKNMITGASQADAAVLVVAVT----D-GV-MP-QTKEHAFLARTLGINNILVAVNKMD--MV
Sul ETRKYFFTVIDAPGHRDFVKNMITGASQADAAILVVSAKKGEYEAGMSAEGQTREHIILSKTMGINQVIVAINKMDLADT
Ent ETSKYYFTIIDAPGHRDFIKNMITGTSQADVAILIVAAGTGEFEAGISKNGQTREHILLSYTLGVKQMIVGVNKMD--AI
Pla ETPRYFFTVIDAPGHKDFIKNMITGTSQADVALLVVPADVGGFDGAFSKEGQTKEHVLLAFTLGVKQIVVGVNKMD--TV
Sty ETAKSVFTIIDAPGHRDFIKNMITGTSQADAAILIIASGQGEFEAGISKEGQTREHALLAFTMGVKQMIVAVNKMDDKSV

    161                                                                          240
Hal DYGESEYKQVVEEV-KDLLTQVRFDSENAKFIPVSAFEGDNIAEESEHTGWYDGEILLEALNELPAPEPPTDAPLRLPIQ
Met NFSEADYNELKKMIGDQLLKMIGFNPEQINFVPVASLHGDNVFKKSERNPWYKGPTIAEVIDGFQPPEKPTNLPLRLPIQ
Tha PYSEKRYNEVKADA-EKLLRSIGFK-D-ISFVPISGYKGDNVTKPSPNMPWYKGPTLLQALDAFKVPEKPINKPLRIPVE
Thc NYDEKKFKAVAEQV-KKLLMMLGYK-N-FPIIPISAWEGDNVVKKSDKMPWYNGPTLIEALDQMPEPPKPTDKPLRIPIQ
Sul PYDEKRFKEIVDTV-SKFMKSFGFDMNKVKFVPVVAPDGDNVTHKSTKMPWYNGPTLEELLDQLEIPPKPVDKPLRIPIQ
Ent QYKQERYEEIKKEI-SAFLKKTGYNPDKIPFVPISGFQGDNMIEPSTNMPWYKGPTLIGALDSVTPPERPVDKPLRLPLQ
Pla KYSEDRYEEIKKEV-KDYLKKVGYQADKVDFIPISGFEGDNLIEKSDKTPWYKGRTLIEALDTMQPPKRPYDKPLRIPLQ
Sty NWDQGRFIEIKKEL-SDYLKKIWLQPRQDPFIPISGWHGDNMLEKSPNMPWFTGSTLIDALDALDQPKRPKDKPLRLPLQ

    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
Sty DVYKIGGIGTVPVGRVETGLLKPGMVLTFAPMN-I----T-TECKSVEMHHESLTEAEPGDNVGFTVKNLSVKDLRRGYV

    321                                                             **           400
Hal CGPADDPPSVA--ET-FQAQIVVMQHPSVITEGYTPVFHAHTAQVACTVESIDKKIDPSSGEVAEE-NPDFIQNGDAAVV
Met LGHTTNPPTVA--TD-FTAQIVVLQHPSVLTDGYTPVFHTHTAQIACTFAEIQKKLNPATGEVLEE-NPDFLKAGDAAIV
Tha CGHLDTPPTVV--KA-FTAQIIVLNHPSVIAPGYKPVFHVHTAQVACRIDEIVKTLNPKDGTTLKE-KPDFIKNGDVAIV
Thc AGHTNNPPTVVRPKDTFKAQIIVLNHPTAITVGYTPVLHAHTLQVAVRFEQLLAKLDPRTGNIVEE-NPQFIKTGDSAIV
Sul AGSVQNPPTVA--DE-FTAQVIVIWHPTAVGVGYTPVLHVHTASIACRVSEITSRIDPKTGKEAEK-NPQFIKAGDSAIV
Ent ASDAKNQPAVG-CED-FTAQVIVMNHPGQIRKGYTPVLDCHTSHIACKFEELLSKIDRRTGKSMEGGEPEYIKNGDSALV
Pla ASDTKNEPAKG-CSK-FTAQVIILNHPGEIKNGYTPLLDCHTSHISCKFLNIDSKIDKRSGKVVEE-NPKAIKSGDSALV
Sty ASDSKNDPAKD-TTN-FLAQVIVLNHPGQIQKGYAPVLDCHTAHIACKFDEIESKVDRRSGKVLEE-EPKFIKSGEAALV

    401                                              *******
Hal TVRPQKPLSIEPSSEIPELGSFAIRDMGQTIAAGKV--L------GV-NE-----R
Met KLIPTKPMVIESVKEIPQLGRFAIRDMGMTVAAGMA--I------QVTAKN----K
Tha KVIPDKPLVIEKVSEIPQLGRFAVLDMGQTVAAGQC--I------DL-EK-----R
Thc VLRPTKPMVIEPVKEIPQMGRFAIRDMGQTVAAGMV--I------SI-QKA----E
Sul KFKPIKELVAEKFREFPALGRFAMRDMGKTVGVGVI--I------DVKPRKVE-VK
Ent KIVPTKPLCVEEFAKFPPLGRFAVRDMKQTVAVGVV--K------AV-TP======
Pla SLEPKKPMVVETFTEYPPLGRFAIRDMRQTIAVGIINQLKRKNLGAVTAKAPA-KK
Sty RMVPQKPMCVEAFNQYPPLGRFAVRDMKQTVAVGVIKEVVKKEQKGMVTKAAQKKK
```

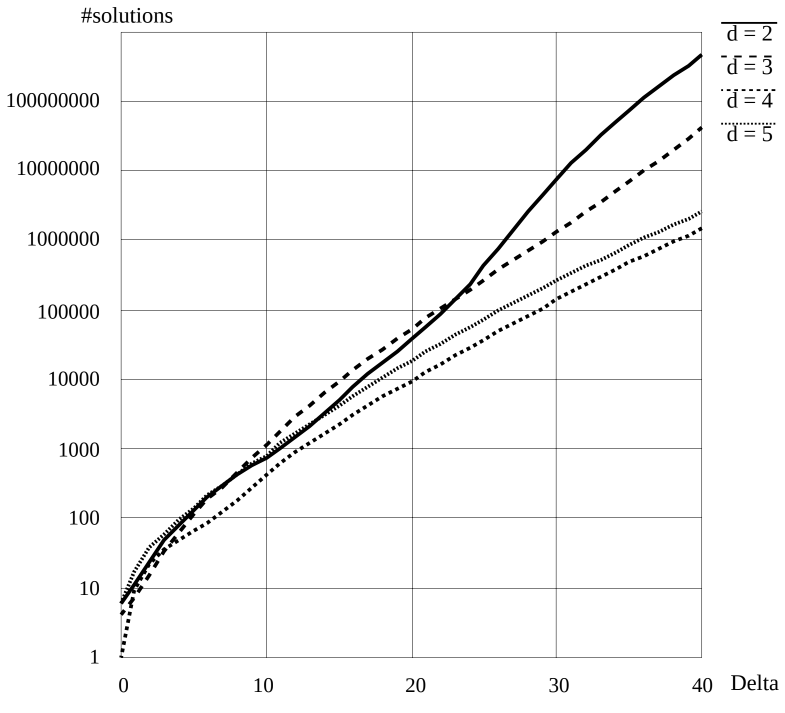
Figure 9: An example of suboptimal alignments of the 8 EF-TU and EF-1α sequences

```
    1                                                                                83
Lum KKQCGVLEGLKVKSEWGRAYGSGHDREAFSQAIWRATFAQVPESRSLFKRVHGDH-TS--DPA-FIAHAERVLGGLDIAISTL
Apl S-LSAAEADL-AGKSWAPVFAN-KN--ANGADFLVALFEKFPDSANFFADFKGKSVADIKASPKLRDVSSRIFTRLNEFVNNA
Bus G-LDGAQKTA-LKESWKVLGADGPTMMKNGSLLFGLLFKTYPDTKKHFKHFDDATFAAMDTTGVGKAHGVAVFSGLGSMICSI
Ct7 APLSADQASL-VKSTWAQV----RN--S-EVEILAAVFTAYPDIQARFPQFAGKDVASIKDTGAFATHAGRIVGFVSEIIALI
Ct3 V-ATPAMPSM-TDAQVAAVKGDWEKIKGSGVEILYFFLNKFPGNFPMFKKL-GNDLAAAKGTAEFKDQADKIIAFLQGVIEKL

    84                                                                             165
Lum DQP--A-TLKEELDHLQVQHEGRKIPDNYFDAFKTAILHVVAAQLGERCYSNNEEI-HDAIACDGFARVLPQVLERGIKGHH
Apl ANA--G-KMSAMLSQFAKEHVGFGVGSAQFENVRSMFPGFVAS-VA----APPAGA-DAAWT-KLFGLII-DAL---KAAGA
Bus DDD--D-CVBGLAKKLSRNHLARGVSAADFKLLEAVFKZFLDE--A----TQRKAT-DAQKD-AD-GALL-TML---IKAHV
Ct7 GNESNAPAVQTLVGQLAASHKARGISQAQFNEFRAGLVSYVSS-NV----AWNAAA-ESAWT-AGLDNIF-GLL---FAA-L
Ct3 GSDM-G-GAKALLNQLGTSHKAMGITKDQFDQFRQALTELLGN-LG---FGGNIGAWNATVD-LMFHVIF-NAL---DGTPV
```

Figure 10: One of the optimal alignments of the 5 globin sequences



(a) enumerating all the alignments



(b) enumerating alignments in classes D_0 and D_1

Figure 11: Number of the suboptimal alignments of d globin sequences whose scores are at most Δ worse than the optimal alignment. (a) is the case enumerating all the alignments. $(0 \leq \Delta \leq 20)$ (b) is the case enumerating alignments in classes D_0 and D_1. $(0 \leq \Delta \leq 40)$

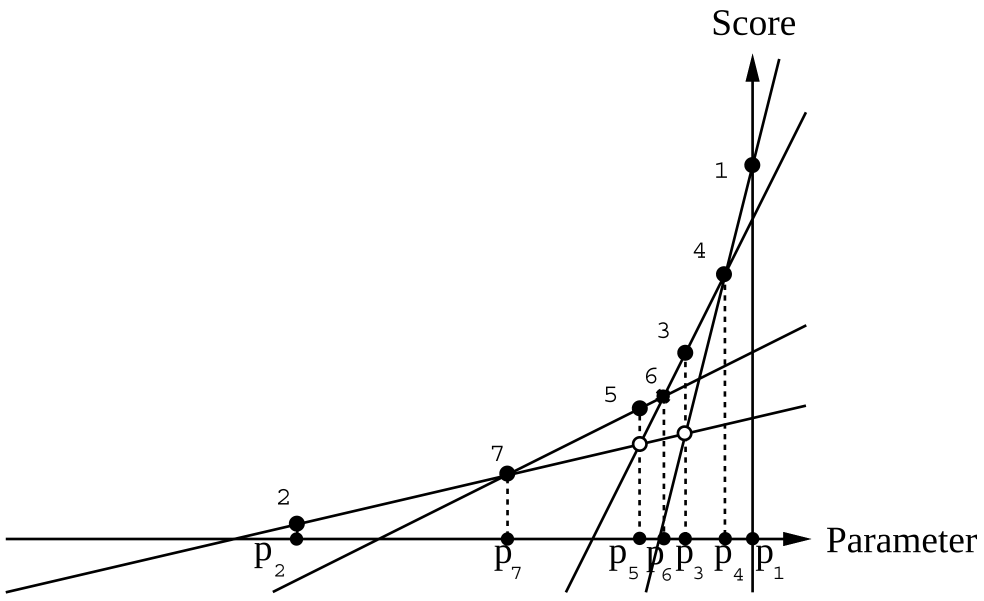


Figure 12: An example of division of 1-parameter space. In this case, there are 4 regions between p_1 and p_2.

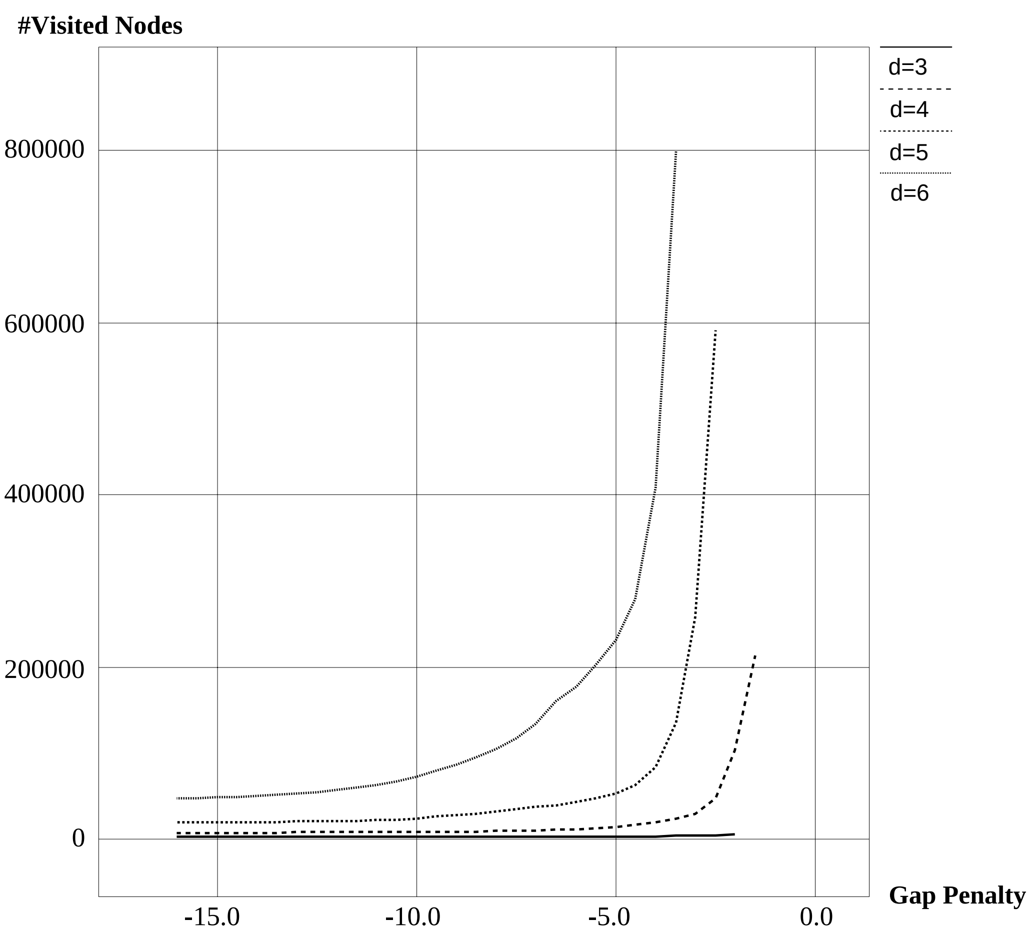


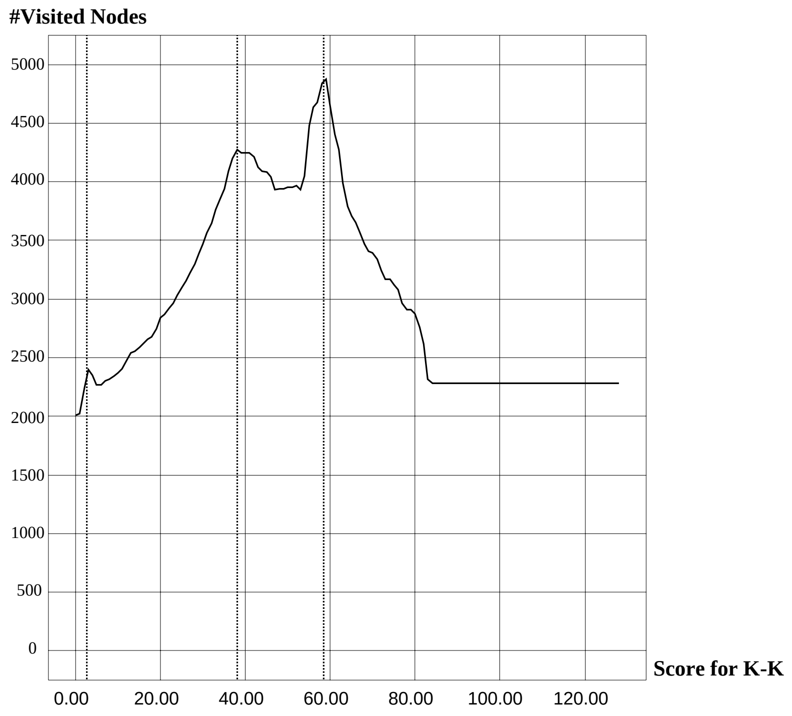
Figure 13: Number of visited nodes by A* algorithm for various gap penalties.

Figure 14: The optimal alignment of 5 rhodopsin sequences based on PAM-250

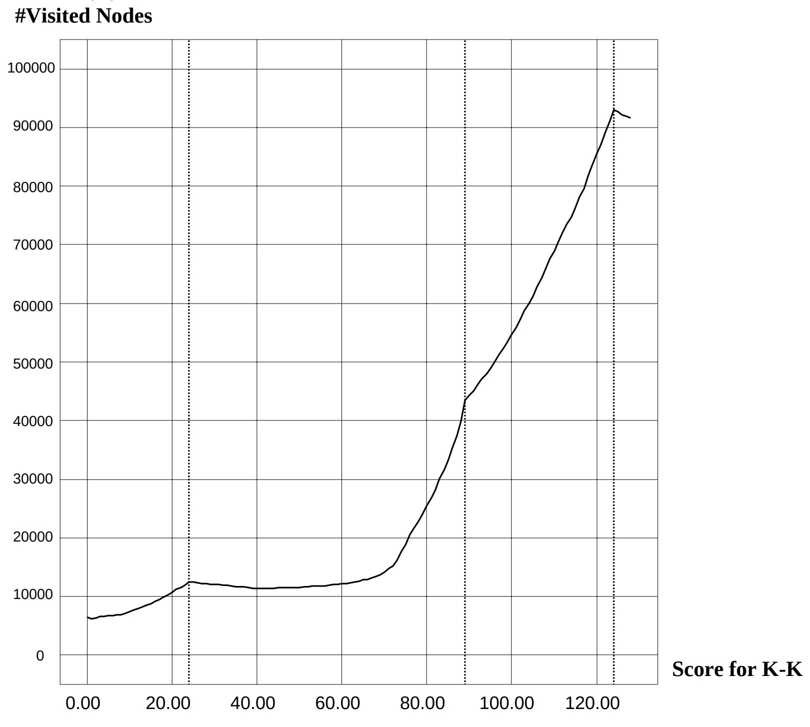
```
     1                 *       *                          55
Hal  ==GLALVQSVGVTSWAYS-VLDVF-AK-YVF---AF-I-LLR-WV-ANN---ER=
HSg  NPGYPFHPLMAALPAFFAKSATIYNPVIYVFMNRQFRNCILQ-LF-GKK----V=
HSr  NPGYAFHPLMAALPAYFAKSATIYNPVIYVFMNRQFRNCILQ-LF-GKK----V=
GGd  NQGSDFGPIFMTIPAFFAKSSAIYNPVIYIVMNKQFRNCMITTLCCGKNPLGDED
HSb  NRNHGLDLRLVTIPSFFSKSACIYNPIIYCFMNKQFQACIMK-MVCGKAMTDESD
```

Figure 15: The optimal alignment of 5 rhodopsin sequences when the score for K-K is between 89 and 124

```
     1                    *                                 60
Hal  GLALVQSVGVTSWAYSVLDVFAK--YVF-A--FILL-R--W---VAN---NER=======
HSg  N----PGYPFHPLMAALPAFFAKSATIYNPVIYVFMNRQFRNC-ILQLF-GKK-V=====
HSr  N----PGYAFHPLMAALPAYFAKSATIYNPVIYVFMNRQFRNC-ILQLF-GKK-V=====
GGd  N----QGSDFGPIFMTIPAFFAKSSAIYNPVIYIVMNKQFRNCMITTLCCGKNPLGDE-D
HSb  N----RNHGLDLRLVTIPSFFSKSACIYNPIIYCFMNKQFQAC-IMKMVCGKA-MTDESD
```



(a) Case of the alignment of 4 rhodopsin sequences



(a) Case of the alignment of 5 rhodopsin sequences

Figure 16: Number of visited nodes by A* algorithm for various scores for K-K. The dotted lines denote the boundaries of the regions where the optimal solution is same.