Enumerating Suboptimal Alignments of Multiple Biological Sequences Efficiently

Tetsuo SHIBUYA, Hiroshi IMAI
Department of Information Science, Faculty of Science, University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan

The multiple sequence alignment problem is very applicable and important in various fields in molecular biology. Because the optimal alignment that maximizes the score is not always biologically most significant, providing many suboptimal alignments as alternatives for the optimal one is very useful. As for the alignment of two sequences, this suboptimal problem is well-studied[6,9,12], but for the alignment of multiple sequences, it has been considered impossible to investigate such suboptimal alignments because of the enormous size of the problem. The optimal multiple alignment can be obtained with A* algorithm[4,5], and an efficient algorithm for the k shortest paths problem on general graphs is discovered recently[1]. We extend these algorithms for computation of set of all aligned groups of residues in optimal and suboptimal alignments, and for enumeration of suboptimal alignments. The suboptimal alignments are numerous. Thus we discuss what kind of suboptimal alignment is unnecessary to enumerate, and propose an efficient technique to enumerate only necessary alignments. The practicality of these algorithms are demonstrated through experiments. Moreover, the property of suboptimal alignments of multiple sequences are also examined through experiments.

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. This method needs $O(n^d)$ 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[4,5].

A suboptimal alignment is an alignment whose score is close to the optimal one. The optimal alignment based on the scoring criterion is not always the biologically most significant alignment, and such candidates as suboptimal alignments for the best alignment are often required. In fact, in case of aligning two sequences, the suboptimal alignments problem are well-studied[6,9,12] and used for many applications such as predicting protein structure and so on[7,8,11].

```
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 1: Examples of suboptimal alignments of multiple protein sequences

In multiple alignment problem, with these methods for only two sequences, we can see suboptimal alignments of each pair of sequences[12], 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[6,12]. But such enumeration becomes easier because a new efficient algorithm for the k shortest paths problem is proposed by Eppstein[1,10]. This algorithm enumerates the k shortest paths in $O(k + n + m)$ time and space after having constructed the shortest path tree from the source or the destination for any graph with non-negative m edges and n vertices.

In this paper, we first discuss the method to obtain E_Δ, which represents all aligned groups of residues in optimal and suboptimal alignments which are at most Δ worse than the optimal, by extending the A* algorithm, and based on this extended A* algorithm and the Eppstein algorithm, we go on to discuss the methods for the enumeration problem. Figure 1 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). Thus, if we check suboptimal alignments one by one, examining (d) may be of no use: we can reconstruct (d) from (b) and (c). Thus we propose a new enumeration algorithm which does not enumerate such unnecessary alignment as (d) based on the algorithms above. We further show the efficiency and practicality of these algorithms and the property of these suboptimal alignments through experiments on groups of protein sequences. In the experiments, we will show that both the number of the alignments and the enumeration time are drastically reduced by ignoring such unnecessary alignments.

2 Computation of a Subgraph Related to Suboptimal Alignments

E_Δ is a set of vertices which are used by the s-t path whose length is at most Δ longer than the shortest path. In this section, we first introduce A* algorithm[4,5], and then extends it to compute E_Δ efficiently.

2.1 A* Algorithm for Multiple Sequence Alignment

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. In multiple alignment problem, the sum of all the scores for alignments of pairwise sequences is generally used as the score. Thus the score of the alignment equals the length of the corresponding path, defining length of each edge as the sum of the lengths of the corresponding edges in the graphs of pairwise alignments. This longest path problem can be easily transformed to the shortest path problem[4,5].

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 t can be used. Ikeda and Imai[4] show the following estimator is very useful 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.

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

$$l'(u, v) = l(u, v) + h(v) - h(u) \tag{1}$$

Note that the time and space used for the DP is negligible, if d is large. A vertex in the graph for 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[4,5] and the computing time is also reduced: we can ignore w such that $L^*(s,v)+l(v,w)>L^+(s,t)$, when we examine the descendant vertices of v. This is called the enhanced A* algorithm.

2.2 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. This problem is well-studied[5,6,9,12], and computing this set E_Δ with A* algorithm is not so complicated. For any path p from s to t, the modified path length by the expression (1) 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. Hence, first we modify the edge lengths, and then we can obtain E_Δ with the Dijkstra method as follows[5].

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:
$$\delta(u,v)=l(u,v)+L^*(s,u)-L^*(s,v) \qquad (2)$$
Then 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 $L^*(s,v)+l(v,w)>L^*(s,t)+\Delta$.

3 Enumeration of Suboptimal Alignments

In this section, we first introduce Eppstein algorithm[1] briefly, and then extend it for the alignment problem. Moreover, we propose a new enumeration method to avoid unnecessary alignments in enumeration.

3.1 Eppstein Algorithm

Eppstein[1] 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[1] also proposed an easier algorithm of $O(m+n\log n+k)$ time.

In the algorithm, we use $\delta(u,v)$ for the edge (u,v) as in equation (2). 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 sidetracks on the path, and we define $sidetrack(p)$ as the nearest sidetrack from s within them.

Let $(tail(p), head(p))$ be $sidetrack(p)$. Then we can suppose a heap, in which the parent of a path p is a path which is same as p from $head(p)$ to t, but go along the shortest path from s to $head(p)$ instead of using $sidetrack(p)$. We define this parent of p as $parent(p)$ and we call p a child of $parent(p)$. 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 $\delta(sidetrack(p))$ longer than $parent(p)$.

The basic concept of the Eppstein algorithm is to modify this path heap to 4-heap. From this heap, we can obtain the k shortest paths in $O(k)$ time[2], or $O(k\log k)$ time in sorted form. The following is the outline of this 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), 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 $\delta(u,v)$, where n represents sidetrack (u,v).
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. For more details, see the paper[1].

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[1]: 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 Extending Eppstein Algorithm to Reduce Memory Space

In this subsection, we discuss our approach for enumeration of 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_Δ after computing E_Δ 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 of Eppstein algorithm, we do not have to compute E_Δ additionally. First, we must take the step 1 and 2 in the subsection 2.2, using upper bounding technique. These procedures cannot be skipped. After these procedures, we implement the Eppstein algorithm as follows:

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 constructed heaps yet.

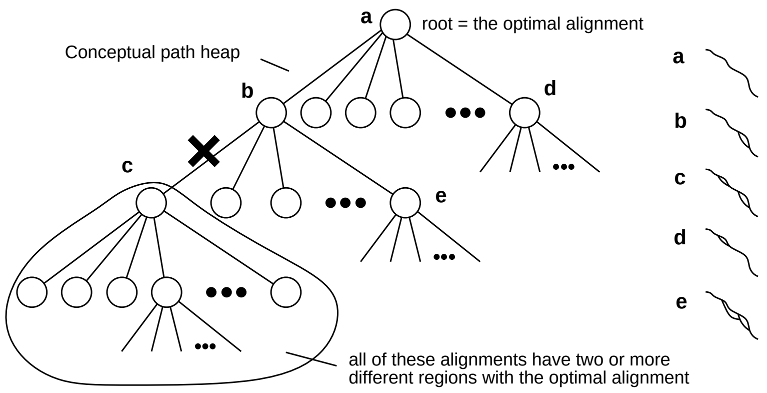
With this method, when we finish enumerating all the suboptimal alignments, the set of vertices for which we constructed the Eppstein heap is also E_Δ. Thus we do not have to compute E_Δ additionally. Notice that this technique can be used in general graphs other than the graphs for alignments.

3.3 Avoiding Unnecessary Alignments

As we will show in the next section, the suboptimal alignments of multiple sequences are numerous. In examining the suboptimal alignments one by one, we should select alignments worth taking time to see. For this requirement, we introduce a notion of alignment class D_i as follows:

Definition 1 *D_i is a class of alignments which have i regions, which are sets of consecutive columns in the alignment, different from the alignment in D_0. There is only one alignment in D_0, and the alignment in D_0 is one of the optimal alignments which is arbitrary chosen.*

In Figure 1, the optimal alignment (a) is in D_0 (and none of the others are in this class), suboptimal alignments (b) and (c) are in D_1, and suboptimal



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

Figure 2: 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.

alignment (d) is in D_2. We can easily construct alignments in D_i ($i \geq 2$) and their scores from alignments in D_0 and D_1. Thus alignments in D_i ($i \geq 2$) are unnecessary to enumerate in many cases.

The technique to avoid such alignments in D_i ($i \geq 2$) in enumeration of suboptimal alignments is rather simple: 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 children (defined in subsection 3.1). Note that it is possible to construct a heap structure for enumerating only alignments in class D_1. Accordingly we can efficiently enumerate only the alignments in class D_1. Moreover, notice that we can extend this algorithm for enumerating alignments in class D_i ($i < d$) for any d.

3.4 Extracting Knowledge from Eppstein Heap

As mentioned by Eppstein[1], the Eppstein heap has a good feature: some of numerical values for each suboptimal solutions 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 indels, the score computed with another score table different from the table used in computing the optimal solution, the length of the alignment, and so on.

4 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 with 128 megabyte memory.

4.1 Case with High Similarity

We first did experiments on a group of 8 sequences with high similarity in Table 1. 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 2). These are higher than in the experiment in the next subsection.

Table 1: 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							

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 2, 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 3, Table 3 and Table 4 show the results of enumerating the suboptimal alignments. Figure 3(a) shows that there are enormous number of suboptimal alignments, and the number increases exponentially as Δ increases. However, in Figure 3(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 4): it seems difficult to

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

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 4).

Observing Figure 3(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.

Table 4: 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

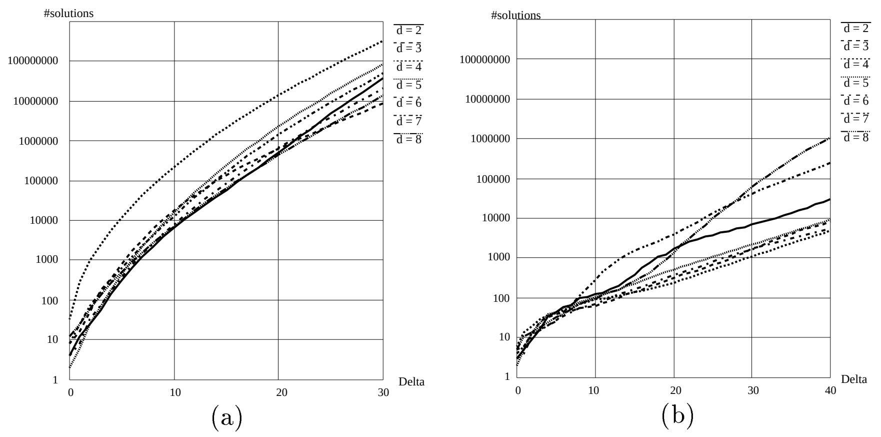


Figure 3: Number of the suboptimal alignments of d sequences of EF-TU and EF-1α whose scores are at most Δ worse than the optimal. (a) is the case enumerating all the alignments. $(0 \leq \Delta \leq 30)$ (b) is the case enumerating alignments in classes D_0 and D_1. $(0 \leq \Delta \leq 40)$

According to Table 3, $|E_\Delta|$ and the size of Eppstein heap for this size of Δ is not so large. Thus, the enumeration time in Table 4(b) is small, though it includes times for constructing the heap. In Figure 3(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 3. 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 3(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.

4.2 Case with Low Similarity

We next did experiments on 5 globin sequences as in Table 5. According to Table 5, 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 4 and Table 6 show the result of the experiments. According to Table 6(a), 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 4, the number of alignments in this experiment is also drastically reduced as in the experiments in the previous subsection by ignoring alignments in class D_i $(i \geq 2)$: the number of alignments in D_1 is only

Table 5: 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 6: (a) Searching time (sec) by simple A* algorithm, (b) the best score and the size of E_Δ in the experiment on the d globin sequences.

(a)

	$d=2$	$d=3$	$d=4$	$d=5$
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

(b)

	$d=2$	$d=3$	$d=4$	$d=5$
best score	29	103	354	543
$\|E_{10}\|$	357	508	380	554
$\|E_{20}\|$	776	1184	866	1159
$\|E_{30}\|$	1149	2403	1575	2448
$\|E_{40}\|$	1544	3839	2669	4569

0.004% of that of all the suboptimal alignments in case $d = 5$ and $\Delta = 20$.

As mentioned by Zuker[12], 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.

5 Concluding Remarks

We investigated suboptimal multiple sequence alignments problem. Based on A* algorithm and Eppstein algorithm, we showed that the suboptimal multiple alignments can be enumerated in practical time. We also proposed a new efficient enumeration method. This method enumerates only the alignments which has only one region different from the optimal solution. The suboptimal alignments enumerated in this way are far fewer than by normal enumeration.

Our technique used with some approximate methods is also useful in many cases. Our algorithm can also be applied to many other optimization problems in molecular biology, such as tree-based alignment problem, gene finding problem, and so on. These remain as future works.

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,

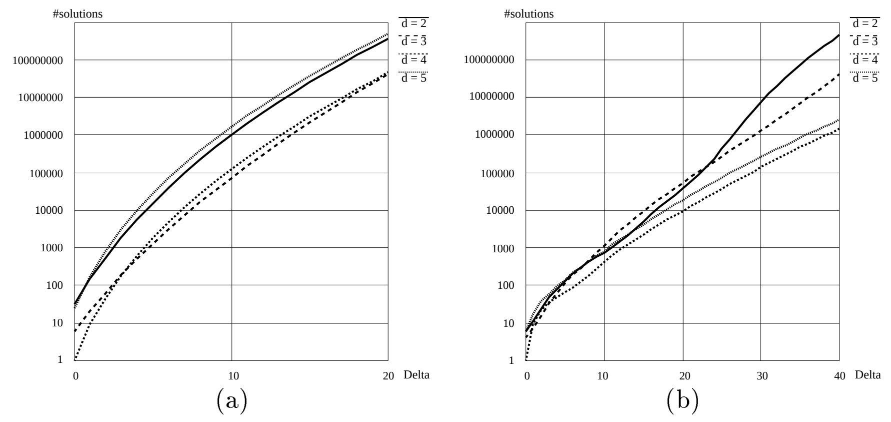


Figure 4: Number of the suboptimal alignments of d globin sequences whose scores are at most Δ worse than the optimal alignment. (a) is the case enumerating all the alignments. $(0 \le \Delta \le 20)$ (b) is the case enumerating alignments in classes D_0 and D_1. $(0 \le \Delta \le 40)$

Sports and Culture of Japan.

References

1. D. Eppstein, *Proc. IEEE Foundation of Computer Science*, 35: pp.154-165, 1994.
2. G. N. Frederickson, *Information and Computation,* 104: pp.197-214, 1993.
3. J. Gracy, et al., *Protein Eng.* 6: pp.821-829, 1993.
4. T. Ikeda, and H. Imai, *Proc. Genome Informatics Workshop V*, pp.90-99, 1994.
5. T. Ikeda, *Master's Thesis,* Dept. of Info. Sci., Univ. of Tokyo, 1995.
6. D. Naor and D. Brutlag, *Proc. 4th Symp. Combinatorial Pattern Matching,* pp.179-196. Springer-Verlag LNCS 684, 1993.
7. M. A. Saqi and M. J. Sternberg, *J. Mol. Biol.* 219: pp.727-732, 1991.
8. M. A. Saqi, et al., *Protein Eng.* 5: pp.305-311, 1992.
9. G. Shibayama, and H. Imai, *Proc. Genome Informatics Workshop IV,* pp.120-129, 1993.
10. T. Shibuya, et al., *Proc. 2nd Intelligent Transportation Systems,* pp.2031-2036, 1995.
11. M. S. Waterman and M. Eggert, *J. Mol. Biol.* 197: pp.723-728, 1987.
12. M. Zuker, *J. Mol. Biol.* 221: pp.403-420, 1991.