PAPER

Near-Optimal Block Alignments*

Kuo-Tsung TSENG[†], **Chang-Biau YANG**[†,††a)], ***Nonmembers***, **Kuo-Si HUANG**[†], ***Student Member***, ***and*** **Yung-Hsing PENG**[†], ***Nonmember***

SUMMARY The optimal alignment of two given biosequences is mathematically optimal, but it may not be a biologically optimal one. To investigate more possible alignments with biological meaning, one can relax the scoring functions to get near-optimal alignments. Though the near optimal alignments increase the possibility of finding the correct alignment, they may confuse the biologists because the size of candidates is large. In this paper, we present the filter scheme for the near-optimal alignments. An easy method for tracing the near-optimal alignments and an algorithm for filtering those alignments are proposed. The time complexity of our algorithm is $O(dmn)$ in the worst case, where d is the maximum distance between the near-optimal alignments and the optimal alignment, and m and n are the lengths of the input sequences, respectively.

key words: *computational biology, longest common subsequence, biosequence alignment, near-optimal alignments*

1. Introduction

Biosequences comparison [1]–[5] could be regarded as the sequence alignment problem, which is a well studied problem in the algorithm area [6]–[10]. With proper measuring schemes, it is not difficult to find the optimal alignment of given sequences. However, there is no completely suitable measuring scheme in biosequences. Scientists have presented many scoring functions to measure the similarity of biosequences [11]–[13]. Most of them failed. There always exist some biosequences with lower scores but higher similarities (judged by biologists or experiments) with any kind of scoring function.

Naor and Brutlag showed that the alignment with optimal score is not always the most biologically meaningful one [14], so they presented the near-optimal alignment in order to provide more possible alignments for biologists to choose. It is natural that the possibility of finding the correct alignment will be increased if we provide more alignments, which, however, may confuse the biologists because of their large cardinality. Thus, some biologically filtering criteria are needed to help us to choose the correct alignment.

In this paper, we present an easy method to trace the near-optimal alignments of given biosequences and propose a novel algorithm to filter the output with some biologically meaningful criteria. We use the criterion: the most conserved alignment proposed by Tseng et al. [15] as the example in our algorithm and name it the *near-optimal block alignment*. Our program URL is http://bio.cse.nsysu.edu.tw/NBA/. It is easy to implement and its computational time is small and might have chances to find the same result as the affine gap penalty does. The criteria to determine meaningful alignments could be open to discussion. The time complexity will not be increased if the filter can be done in linear time.

The rest of this paper is organized as follows. In Sect. 2, we shall give an easy method to trace the near-optimal alignments. Next, we shall illustrate the proposed algorithm to filter out the desired alignment by the most conserved criterion in Sect. 3. Finally, some discussions and conclusions will be given in Sect. 4.

2. Tracings in the Alignment Lattice

In this section, we shall demonstrate the idea of tracings in the alignment lattice, and show how they help us to find the optimal and near-optimal alignments. Let $S1$ and $S2$ be two input sequences, where $|S1| = m$ and $|S2| = n$. We first use an example to explain our idea. Suppose two sequences $S1 = abdcd$ and $S2 = bacddb$ are given to be aligned with the score matrix shown in Table 1. We then have the alignment lattice AL of sequences $S1$ and $S2$ shown in Fig. 1 after performing the traditional alignment scheme [6]–[10].

The bold lines in Fig. 1 represent the correct alignments in the corresponding positions. The numbers beside lines are the costs of alignments. It is well known that the optimal alignment can be obtained if we trace back the alignment lattice AL from the lower right corner to the upper left corner [8]. In our example, there are two optimal alignments $\frac{\texttt{abdcd--}}{\texttt{-bacddb}}$ and $\frac{\texttt{abdc-d-}}{\texttt{-bacddb}}$.

Table 1 The score matrix of {a,b,c,d}.

	-	a	b	c	d
-	$-\infty$	-1	-1	-1	-1
a	-1	4	1	0	2
b	-1	1	3	0	-2
c	-1	0	0	2	1
d	-1	2	-2	1	1

Manuscript received November 9, 2006.
Manuscript revised August 16, 2007.

[†] The authors are with the Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan.
[††] Correspondence author.

[*] This research work was partially supported by the National Science Council of Taiwan under contract NSC-95-2221-E-110-084.

a) E-mail: cbyang@cse.nsysu.edu.tw

Program URL: http://bio.cse.nsysu.edu.tw/NBA/
DOI: 10.1093/ietisy/e91–d.3.789

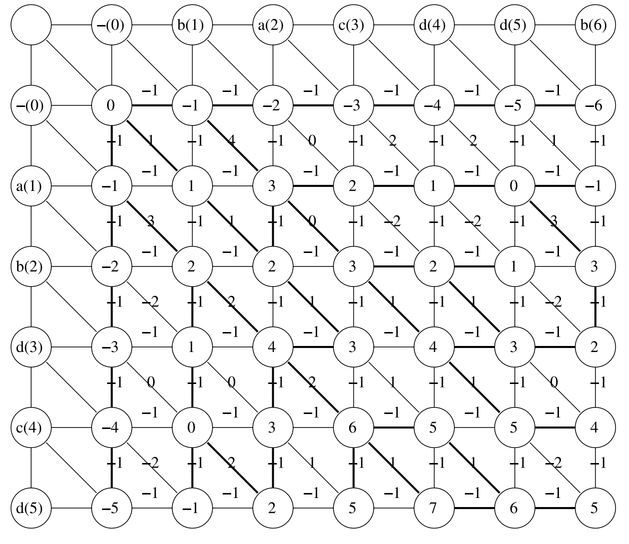

Fig. 1 The alignment lattice AL of sequences abdcd and bacddb with the score matrix shown in Table 1.

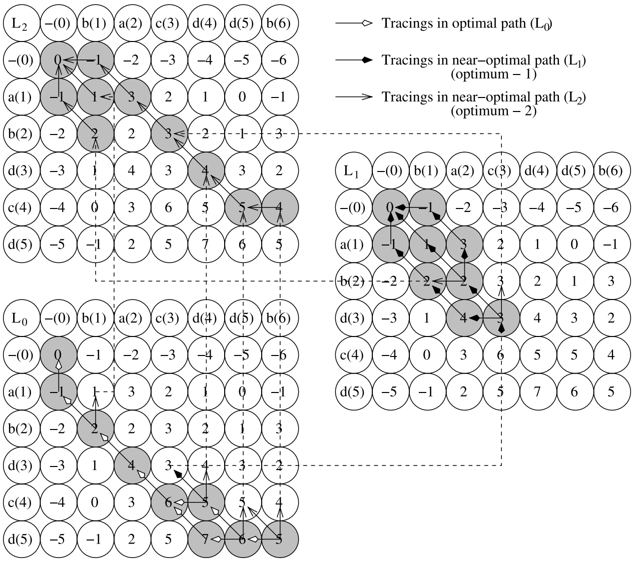
Tracings in optimal path (L0)
Tracings in near–optimal path (L1) (optimum – 1)
Tracings in near–optimal path (L2) (optimum – 2)
L2
L1
L0
–(0) b(1) a(2) c(3) d(4) d(5) b(6)
–(0) 0 –1 –2 –3 –4 –5 –6
a(1) –1 1 3 2 1 0 –1
b(2) –2 2 2 3 2 1 3
d(3) –3 1 4 3 4 3 2
c(4) –4 0 3 6 5 5 4
d(5) –5 –1 2 5 7 6 5

Fig. 2 Tracings in optimal and near-optimal alignments of sequences abdcd and bacddb.

To find the optimal alignment of two given sequences is easy, but how to find those alignments which are within d score compared to the optimal alignment needs some tricks. The optimal alignment means that we choose the correct alignment, for example bold lines in Fig. 1, in each position. What would it be if we choose the wrong alignment somewhere? The score of somewhere-wrong-alignment is worse than the optimal alignment, but how bad will it be?

Let $P = (i, j, U)$ denote the alignment from position (i, j) to the U direction, where $0 \le i \le m$, $0 \le j \le n$, and $U \in \{H, V, D\}$. The U direction of $\{H, V, D\}$ means the *Horizontal*, *Vertical*, or *Diagonal* direction. We define the δ function to calculate the effect when the alignment P, with respect to $AL(i, j)$, is chosen as follows.

$$\delta(P) = AL(i, j) + \begin{cases} -AL(i, j-1) - ScoreMatrix(-, S2_j) & \text{if } U = H, \\ -AL(i-1, j) - ScoreMatrix(S1_i, -) & \text{if } U = V, \\ -AL(i-1, j-1) - ScoreMatrix(S1_i, S2_j) & \text{if } U = D, \end{cases}$$

where $S1_i$ and $S2_j$ represent the ith and jth characters of $S1$ and $S2$, respectively.

Traditionally, $AL(i, j)$ is undefined when $i < 0$ or $j < 0$, so that $\delta(P) = \infty$ if an undefined value is encountered. We use examples to illustrate the effect measurement of $\delta(P)$. For example, suppose $P = (5, 6, D)$ which is an incorrect alignment in Fig. 1. Clearly, $\delta(P) = 2$ here. If P is chosen and afterward we choose the correct alignments, i.e. bold lines, from position (4, 5) till position (0, 0). The result score would be *optimal score* $- \delta(P) = 5 - 2 = 3$. As another example, we follow the bold lines from position (5, 6) till position (4, 4), $P = (4, 4, D)$ is chosen, and afterward we follow the bold lines from position (3, 3) till position (0, 0). Here, $\delta(P) = 1$. It will construct an alignment with score $= 5 - \delta(P) = 5 - 1 = 4$. Similarly, $\delta(4, 4, H) = 0$ and $\delta(4, 4, V) = 2$.

It is easy to prove that we choose all the ways of correct alignments from the lower right corner to P, and then choose all the ways of correct alignments from P to the upper left corner, we will construct an alignment with score $\delta(P)$ less than the optimal alignment. In other words, $\delta(P) = 0$ if and only if the alignment P is the correct alignment in the corresponding position, i.e. the bold lines in Fig. 1. With this fact, we are able to find the near-optimal alignments within d score less than the optimal score. Figure 2 shows the near-optimal alignment example of the sequences given above with $d = 2$. The possible partial alignments for constructing the near-optimal alignments are called *tracings*. Hollow arrows in layer 0 (L_0) will construct the optimal alignments, solid arrows will construct the near-optimal alignments with score *optimum* – 1 in layer 1 (L_1) and simple arrows in layer 2 (L_2) have similar meaning but with score *optimum* – 2.

Our method to mark tracings is shown as follows. These tracings are recorded from the lower right corner back to the upper left corner. This method can be regarded as a traditional back tracing technique if d is set to 0. It will help us to process those possible partial alignments that may be used to construct the near-optimal alignments if d is greater than 0. The function $EnQ(x, Y)$ is used to add element x into queue Y, and the function $x = DeQ(Y)$ is used to remove the first element of queue Y and to store it in x. Elements in a queue are in the form of $[k](i, j)$, which means the position (i, j) of layer k. $TR_k(i, j, U) = \alpha$ represents an possible alignment coming from layer α ($0 \le \alpha \le k \le d$) in the U ($U \in \{H, V, D\}$) direction to position (i, j) of layer k. Note that $TR_k(i, j, U) = -1$ means that there is no possible alignment coming from the U direction, For example, $TR_2(4, 5, V) = 0$ represents an alignment going to position (4, 5) of layer 2 in the V direction from layer 0. (Since it comes from the V direction, we know the position of it is (5, 5).) Q is a temporary queue in the tracing marking method and queue R will be used in our near-optimal block align-

Table 2 All near-optimal alignments of sequences abdcd and bacddb when $d = 2$, $\tau = 0$ and $\psi = 2$.

Layer	Alignment	A	ω
0	$\dfrac{\mathtt{a \vert bdc \vert - \vert d \vert -}}{\mathtt{- \vert bac \vert d \vert d \vert b}}$	$\{\tau_1^-, \tau_3^+, \tau_1^-, \tau_1^+, \tau_1^-\}$	$1^2 + 3^2 + 1^2 + 1^2 + 1^2 = 13$
0	$\dfrac{\mathtt{a \vert bdcd \vert --}}{\mathtt{- \vert bacd \vert db}}$	$\{\tau_1^-, \tau_4^+, \tau_2^-\}$	$1^2 + 4^2 + 2^2 = 21$
1	$\dfrac{\mathtt{- \vert a \vert b \vert dcd \vert -}}{\mathtt{b \vert a \vert - \vert cdd \vert b}}$	$\{\tau_1^-, \tau_1^+, \tau_1^-, \tau_3^+, \tau_1^-\}$	$1^2 + 1^2 + 1^2 + 3^2 + 1^2 = 13$
1	$\dfrac{\mathtt{abdcd \vert -}}{\mathtt{bacdd \vert b}}$	$\{\tau_5^+, \tau_1^-\}$	$5^2 + 1^2 = 26$
1	$\dfrac{\mathtt{a \vert bd \vert - \vert cd \vert -}}{\mathtt{- \vert ba \vert c \vert dd \vert b}}$	$\{\tau_1^-, \tau_2^+, \tau_1^-, \tau_2^+, \tau_1^-\}$	$1^2 + 2^2 + 1^2 + 2^2 + 1^2 = 11$
2	$\dfrac{\mathtt{- \vert a \vert b \vert dc \vert -d}}{\mathtt{b \vert a \vert c \vert dd \vert b-}}$	$\{\tau_1^-, \tau_1^+, \tau_1^=, \tau_2^+, \tau_2^-\}$	$1^2 + 1^2 + 1^2 + 2^2 + 2^2 = 11$
2	$\dfrac{\mathtt{- \vert a \vert b \vert dc \vert d}}{\mathtt{b \vert a \vert c \vert dd \vert b}}$	$\{\tau_1^-, \tau_1^+, \tau_1^=, \tau_2^+, \tau_1^-\}$	$1^2 + 1^2 + 1^2 + 2^2 + 1^2 = 8$
2	$\dfrac{\mathtt{- \vert a \vert b \vert dc \vert d-}}{\mathtt{b \vert a \vert c \vert dd \vert -b}}$	$\{\tau_1^-, \tau_1^+, \tau_1^=, \tau_2^+, \tau_2^-\}$	$1^2 + 1^2 + 1^2 + 2^2 + 2^2 = 11$
2	$\dfrac{\mathtt{- \vert a \vert b \vert d \vert c \vert d \vert -}}{\mathtt{b \vert a \vert c \vert d \vert - \vert d \vert b}}$	$\{\tau_1^-, \tau_1^+, \tau_1^=, \tau_1^+, \tau_1^-, \tau_1^+, \tau_1^-\}$	$1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 = 7$
2	$\dfrac{\mathtt{a \vert b \vert dc \vert - \vert d \vert -}}{\mathtt{b \vert - \vert ac \vert d \vert d \vert b}}$	$\{\tau_1^+, \tau_1^-, \tau_2^+, \tau_1^-, \tau_1^+, \tau_1^-\}$	$1^2 + 1^2 + 2^2 + 1^2 + 1^2 + 1^2 = 9$
2	$\dfrac{\mathtt{a \vert b \vert dcd \vert --}}{\mathtt{b \vert - \vert acd \vert db}}$	$\{\tau_1^+, \tau_1^-, \tau_3^+, \tau_2^-\}$	$1^2 + 1^2 + 3^2 + 2^2 = 15$
2	$\dfrac{\mathtt{- \vert a \vert b \vert d \vert cd \vert -}}{\mathtt{b \vert a \vert c \vert - \vert dd \vert b}}$	$\{\tau_1^-, \tau_1^+, \tau_1^=, \tau_1^-, \tau_2^+, \tau_1^-\}$	$1^2 + 1^2 + 1^2 + 1^2 + 2^2 + 1^2 = 9$
2	$\dfrac{\mathtt{a \vert b \vert - \vert dcd \vert -}}{\mathtt{- \vert b \vert a \vert cdd \vert b}}$	$\{\tau_1^-, \tau_1^+, \tau_1^-, \tau_3^+, \tau_1^-\}$	$1^2 + 1^2 + 1^2 + 3^2 + 1^2 = 13$

ment algorithm, which will be demonstrated in Sect. 3. Actually, the tracing marking method and near-optimal block alignment algorithm can be done together, so that we can use one queue Q only. It is for clarity that we explain our idea in this way.

Method: Tracing Marking

```
Input: Alignment lattice $AL$ with threshold $d$.
Output: Tracing queue $R$ and tracings (possible alignments) $TR$ that construct near-optimal alignments
        within $d$ score compared to the optimal alignment.
Step 1: Initialization: $TR_k(i, j, U) = -1$, where $0 \leq k \leq d$, $0 \leq i \leq m$, $0 \leq j \leq n$ and $U \in \{H, V, D\}$. $Q = \emptyset$,
        $R = \emptyset$.
Step 2: $EnQ([0](m, n), Q)$, $EnQ([0](m, n), R)$.
Step 3: If $Q \neq \emptyset$, then $B = DeQ(Q)$; otherwise, stop.
Step 4:
    Let the content of $B$ be $[k](i, j)$
        $\alpha = \delta((i, j, H)) + k$,
    and $\beta = \delta((i, j, V)) + k$,  then
        $\gamma = \delta((i, j, D)) + k$,
    $\begin{cases} EnQ([\alpha](i, j-1), Q), \\ EnQ([\alpha](i, j-1), R), & \text{if } \alpha \leq d, \\ TR_\alpha(i, j-1, H) = k, \end{cases}$
    $\begin{cases} EnQ([\beta](i-1, j), Q), \\ EnQ([\beta](i-1, j), R), & \text{if } \beta \leq d, \\ TR_\beta(i-1, j, V) = k, \end{cases}$
    $\begin{cases} EnQ([\gamma](i-1, j-1), Q), \\ EnQ([\gamma](i-1, j-1), R), & \text{if } \gamma \leq d. \\ TR_\gamma(i-1, j-1, D) = k, \end{cases}$
Step 5: Go to Step 3.
```

In the above tracing marking method, we do our tracing starting from the lower right corner, so Step 2 adds $[0](m, n)$ on layer 0 as the first (source) element of our queue. At Step 4, we process the extracted element B and then calculate the effect of each direction. Since element B is at layer k, B will go to layer α if $P = (i, j, H)$ is chosen. If $\alpha > d$, we ignore it. Otherwise we add the next position into our queue and record that it comes from layer k. For example in Fig. 2, element $[0](4, 4).\{\alpha, \beta, \gamma\} = \{0, 2, 1\}$ represents that it goes to layers 0, 2 and 1 in the H, V and D directions, respectively.

3. An Algorithm for Near-Optimal Block Alignment

In Sect. 2, we gave the method to trace back all near-optimal alignments. Actually, there are numerous near-optimal alignments even when d is small. All near-optimal alignments of Fig. 2 are listed in Table 2. As we can see, there are 2 alignments in layer 0 (Position (5, 5) in layer 0 branches two ways), 3 alignments in layer 1 (Position (3, 3) branches two ways and one of them branches three ways again at position (2, 2), but only two ways go to layer 1, so $1 + 2 = 3$.) and 8 alignments in layer 2. It is not so useful if we just list all of the near-optimal alignments. Some filtering schemes should be invoked to help us to choose the most meaningful alignment. The filtering scheme could be various in many aspects. Here we use the *most conserved alignment* which was defined by Tseng et al. [15] as our filtering scheme. The idea of the near-optimal block alignment is similar to finding motifs between two sequences. When two biosequences are aligned, the common parts of them are more meaningful. Those parts may be some functional genes or help us to se-

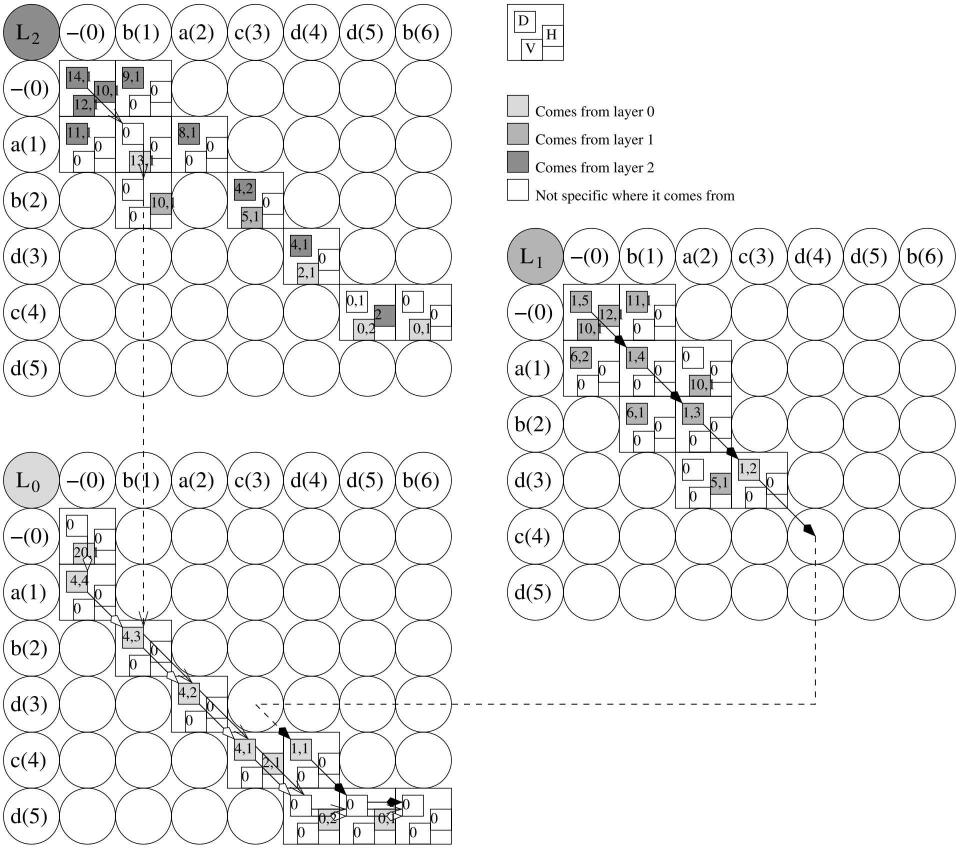

Fig. 3 The final result of sequences abdcd and bacddb after Algorithm NBA is performed.

lect the better templates when predicting the 3D structure of proteins based on the homology modeling technique [16]. Sometimes we need to focus our attention on their different parts to cast the junk of biosequences. Concluding the above, we have to divide the sequences into either common/meaningful or different/junk parts, and we call these parts as *blocks* in this paper. The longer blocks are the better since the longer common/different parts are more significant than the shorter ones.

In this section, we shall propose an algorithm to solve the near-optimal block alignment problem. Given $\tau \in R$, a τ_i^+-*block*, $\tau_i^=$-*block*, or τ_i^--*block* is a maximum area with score continuously greater than, equal to, or less than the threshold τ, respectively, where i represents the length of that block. τ is a threshold used to judge if an alignment of two characters is similar enough or not. For example, suppose $\tau = 0$, the alignment $\frac{\texttt{-abdcd-}}{\texttt{bacdd-b}}$ can be divided into $\tau_1^-|\tau_1^+|\tau_1^=|\tau_2^+|\tau_2^-$, which is $\begin{array}{c|c|c|cc|cc} -1 & 4 & 0 & 1 & 1 & -1 & -1 \\ - & a & b & d & c & d & - \\ \hline b & a & c & d & d & - & b \end{array}$, where the score of each character pair is shown upon it. As another example, suppose $\tau = 2$. The same alignment is now divided into $\tau_1^-|\tau_1^+|\tau_5^-$, which is $\begin{array}{c|c|ccccc} -1 & 4 & 0 & 1 & 1 & -1 & -1 \\ - & a & b & d & c & d & - \\ \hline b & a & c & d & d & - & b \end{array}$. Note that the way to divide an alignment into $\tau - blocks$ is unique. For example, it is invalid if we divide the above alignment into $\tau_1^-|\tau_1^+|\tau_2^-|\tau_3^-$ with $\tau = 2$, since a $\tau - block$ is a maximum continuous area, and then $\tau_2^-|\tau_3^-$ should be merged into τ_5^-.

After two sequences have been aligned, the alignment (A) could be regarded as a list of $\tau - blocks$. Tseng et al. [15] defined ω in their paper to judge if an alignment is conversed. The formal definition of ω is given as follows.

$$A = \{\tau_{a_1}^{t_1}, \tau_{a_2}^{t_2}, \cdots, \tau_{a_l}^{t_l}\},$$

where l is the number of blocks in A, $t_i \in \{+, =, -\}$, $1 \le i \le l$. And,

$$\omega(A) = \textstyle\sum_{1 \le i \le l}(a_i)^{\psi},$$

where ψ is a parameter which is 2 in this paper.

The near-optimal block alignment will be A if $\omega(A)$ is maximum. An example is illustrated in Table 2.

Clearly, the alignment with larger ω means the alignment with longer blocks. As we mentioned before, the longer blocks are the better. The near-optimal block alignment problem is to find the alignment with the maximum ω in all near-optimal alignments, which is $\frac{\texttt{abdcd|-}}{\texttt{bacdd|b}}$ in our example when $d = 2$, $\tau = 0$ and $\psi = 2$.

Let us take sequences abdcd and bacddb as our example in Fig. 3. There are three little squares inside each square. Each little square represents the accumulated ω

from the lower right corner (position (6, 5)) of layer 0 to this position in the respective direction. If there are two numbers in the little square, then the left number represents ω and the other denotes the current block length at that position. The current block length is 0 if it is not shown. The circle positions means impossible alignments, so we will not show them.

Before presenting our algorithm, we first explain the meanings of variables used in the algorithm. The alignment lattice AL is of size $(m+1) \times (n+1)$, where m and n are the lengths of the two given sequences, respectively. In our algorithm, $C(i, j, U)$ $(U \in \{H, V, D\})$ denotes the added score (edge weight) from the prior horizontal, vertical, or diagonal position to position (i, j), and $\omega_k(i, j, U)$ $(U \in \{H, V, D\})$ denotes the maximum $\sum_{1 \leq i \leq k} (a_i)^{\psi}$ from position (m, n) of layer 0 across the prior horizontal, vertical, or diagonal positions to position (i, j) of layer k. The last, $L_k(i, j, U)$ $(U \in \{H, V, D\})$ denotes the current block length of position (i, j) of layer k that comes from various directions. Our algorithm is given as follows.

```
**Algorithm: Near-optimal Block Alignments (NBA)**
**Input:** Alignment lattice $AL$, tracings of possible alignments $TR$ and tracing queue $R$.
**Output:** Maximum $\omega$ among all near-optimal alignments.
**Step 1:** Initialization: $\omega_k(i, j, U)=0$ and $L_k(i, j, U)=0$,
    where $0 \leq k \leq d$, $0 \leq i \leq m$, $0 \leq j \leq n$, and $U \in \{H, V, D\}$.
**Step 2:** $C(i, j, U) = \begin{cases} ScoreMatrix(-, S2_j) & \text{if } U = H, \\ ScoreMatrix(S1_i, -) & \text{if } U = V, \\ ScoreMatrix(S1_i, S2_j) & \text{if } U = D, \end{cases}$
    where $S1_i$ and $S2_j$ represent the $i$th and $j$th characters of $S1$ and $S2$, respectively and $0 \leq i \leq m$, $0 \leq j \leq n$, and $U \in \{H, V, D\}$.
    $C(i, j, U) = \infty$ if an undefined value is encountered.
**Step 3:** If $R \neq \emptyset$, then $B = DeQ(R)$; otherwise go to Step 6.
**Step 4:** Let $[k](i, j)$ be the content of $B$.
    $\Delta = TR_k(i, j, U)$,
    $\omega_k(i, j, U) = \begin{cases} Choose(k, i, j, \Delta, U) & \text{if } \Delta \geq 0, \\ 0 & \text{if } \Delta = -1, \end{cases}$
    where $U \in \{H, V, D\}$.
    $\omega_k(i, j, U) = 0$ if an undefined value is encountered.
**Step 5:** Go to Step 3.
**Step 6:** Output $max(\omega_k(0, 0, U) + (L_k(0, 0, U))^{\psi})$, where $0 \leq k \leq d$, $U \in \{H, V, D\}$.
```

For example in Fig. 2, suppose element [2](3, 4) has to be processed now. It is clear that there is no incoming edge from direction H, so $\omega_2(3, 4, H) = 0$. And there is only one way to go to the position (3,4) of layer 2 in V direction, we will leave the $\omega_2(3, 4, V)$ out of discussion. If we want to decide the value of $\omega_2(3, 4, D)$, we have to look over all the incoming edges of position (4,5) of layer 2. (Position (4,5) is the prior position of position (3,4) in direction D.) In this case, it has three incoming edges from H, V and D directions. (Directions V and D come in from layer 0, and direction H comes from layer 2.) Though we have known the values of ω and L of those prior positions, we need to check if the current block can be extended or not when we choose the edge of some direction. All three incoming edges get negative scores, but we get positive score in D direction. (Threshold τ is 0 in our example.) It means that a new block starts, and then we have to reset current block length to 1 and to calculate the current ω by adding the power ψ (ψ is an predefined parameter, which is 2 in this paper.) of prior block length. $\omega_2(3, 4, D)\{H, V, D\} = \{4, 4, 1\}$ in this case, since we have to choose the maximum, $\omega_2(3, 4, D) = 4$. Note that if there are more than one maximum, we should choose the one with the longest current block length.

Since it is complicated to decide the correct value of $\omega_k(i, j, U)$, we use the function $Choose(k, i, j, \Delta, U)$ to choose the value. We show function $Choose(k, i, j, \Delta, U)$ and the meanings of its arguments as follows.

```
**Function: Choose(k, i, j, Δ, U)**
**Input:** $k$, $i$ and $j$, where $k$ is the index of the layer, $i, j$ mean the coordinates, $\Delta$ is the incoming layer and $U \in \{H, V, D\}$ means the direction that it came from.
**Output:** The correct value of $\omega_k(i, j, U)$, and the value of $L_k(i, j, U)$ which is updated to a correct one.
**Step 1:** $(x, y) = \begin{cases} (i, j+1) & \text{if } U = H. \\ (i+1, j) & \text{if } U = V. \\ (i+1, j+1) & \text{if } U = D. \end{cases}$
**Step 2:** Check if the phase is changed from $(x, y, U')$ to $(i, j, U)$, where $(x, y, U')$ represent the outgoing edge of direction $U'$ at position $(x, y)$.
    A phase is said to be *changed* if and only if one of the following conditions holds.
    $\begin{cases} C(x, y, U') < \tau \;\&\; C(i, j, U) \geq \tau, \\ C(x, y, U') > \tau \;\&\; C(i, j, U) \leq \tau, \\ C(x, y, U') = \tau \;\&\; C(i, j, U) \neq \tau, \end{cases}$
    where $U' \in \{H, V, D\}$.
    A changed phase means a new block, and we have to reset the length of current block to 1.
**Step 3:**
    Compute the following:
    $TempL[U'] =$
    $\begin{cases} 1 & \text{if phase is changed} \\ & \text{from } (x, y, U') \text{ to } (i, j, U), \\ L_{\Delta}(x, y, U') + 1 & \text{otherwise}, \end{cases}$
    where $U' \in \{H, V, D\}$.

    $Temp\omega[U'] =$
    $\begin{cases} \omega_{\Delta}(x, y, U') + (L_{\Delta}(x, y, U'))^{\psi} + (TempL[U'])^{\psi} \\ \text{(if phase is changed from } (x, y, U') \text{ to } (i, j, U)\text{),} \\ \\ \omega_{\Delta}(x, y, U') + (TempL[U'])^{\psi} \\ \text{(otherwise),} \end{cases}$
    where $U' \in \{H, V, D\}$.
**Step 4:** Without loss of generality, assume that $Temp\omega[Z]$ is not less than the other two. Then:
    $L_k(i, j, U) = TempL[Z]$

    $OK =$
```

Table 3 An example of protein sequences which affine gap penalty and NBA (layer 14) result in the same alignment.

Protein sequence S1 (PDB ID= 1hgb C)	VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSH GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKL LSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR
Protein sequence S2 (PDB ID= 1a9w E)	VHFTAEEKAAVTSLWSKMNVEEAGGEALGRLLVVYPWTQRFFDSFGNLSS PSAILGNPKVKAHGKKVLTSFGDAIKNMDNLKPAFAKLSELHCDKLHVDP ENFKLLGNVMVIILATHFGKEFTPEVQAAWQKLVSAVAIALAHKYH
Parameters	Scoring function: PAM250 ; Gap Penalty: -9 (for origin, affine gap penalty and NBA) Gap Opening Penalty: -18 (for affine gap penalty only) τ: -8.5 (for NBA only)
Alignment by affine gap penalty and NBA	$\frac{\texttt{V-LSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFD------LSHGSAQVKGHGKKVADALTNAVAH}}{\texttt{VHFTAEEKAAVTSLWSKM--NVEEAGGEALGRLLVVYPWTQRFFDSFGNLSSPSAILGNPKVKAHGKKVLTSFGDAIKN}}$ $\frac{\texttt{VDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR}}{\texttt{MDNLKPAFAKLSELHCDKLHVDPENFKLLGNVMVIILATHFGKEFTPEVQAAWQKLVSAVAIALAHKYH}}$
Original optimal alignment	$\frac{\texttt{V-LSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF-DLS-H----GSAQVKGHGKKVADALTNAVAH}}{\texttt{VHFTAEEKAAVTSLWSKM--NVEEAGGEALGRLLVVYPWTQRFFDSFGNLSSPSAILGNPKVKAHGKKVLTSFGDAIKN}}$ $\frac{\texttt{VDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR}}{\texttt{MDNLKPAFAKLSELHCDKLHVDPENFKLLGNVMVIILATHFGKEFTPEVQAAWQKLVSAVAIALAHKYH}}$

$$
\begin{cases}
\omega_\Delta(x,y,Z) + (L_\Delta(x,y,Z))^\psi \\
\text{(if phase is changed from } (x,y,Z) \text{ to } (i,j,U)), \\
\\
\omega_\Delta(x,y,Z) \\
\text{(otherwise).}
\end{cases}
$$

Notice that if there are more than one maximum in *Tempω*{H,V,D}, we should find the most benefit one, i.e. the one with the longest current block length, as our Z.

Step 5: Return(OK).

Figure 3 shows the full result after NBA algorithm is performed. In this example, the maximum ω is 21 of layer 0, with the alignment $\frac{\texttt{a|bdcd|--}}{\texttt{-|bacd|db}}$; 26 of layer 1, with the alignment $\frac{\texttt{abdcd|-}}{\texttt{bacdd|b}}$; 15 of layer 2, with the alignment $\frac{\texttt{a|b|dcd|--}}{\texttt{b|-|acd|db}}$. As we can see, there is a positive block with length 5 in layer 1. It means that the block may be more meaningful if it is in biosequences.

It is clear that the time complexity of Algorithm NBA is $O(dmn)$. We may reduce the time complexity to $O(|R|)$ which is much less than $O(dmn)$ if we skip the initialization in Step 1.

4. Discussions and Conclusions

We compare our algorithm (NBA) with the affine gap penalty alignment since it is widely believed that affine gap is more appropriate for alignments with biological meaning. There is no doubt about that the result of affine gap penalty alignment belongs to near-optimal alignments when the original scoring function is applied. (Since the threshold d in the "near" definition can be adjusted, any alignment falls in near-optimal alignments.) The question now is: Can NBA find the alignment with the fewest gap blocks in alignments of the same scores? By the definition of ω, it is clear that finding the maximum ω implies finding the fewest (or longest, not always but usually) $\tau - blocks$ since the sum of $\tau - block$ lengths is almost fixed. Therefore, if gap blocks can be regarded as $\tau^- - blocks$, NBA might have chances to find the same alignment as affine gap does.

For example, there are two real protein sequences $S1$ and $S2$ in Table 3 to be aligned. Since the worst matched score in PAM250 is -8 and we hope that the gap blocks can be regarded as $\tau^- - blocks$ in NBA, gap penalty is set to -9 (less than -8) and then τ is set to -8.5 (between the worst matched score and the gap penalty). After NBA is executed, we have exactly the same alignment in layer 14 as affine gap penalty does if gap opening penalty is -18. This result implies with careful parameter setting, our NBA may do what affine gap penalty does. By the way, the computational time for our example in Table 3 is 0.06 seconds in a Linux server with 2 GB RAM and an AMD Athlon(tm) 64 X2 Dual Core Processor 3800+ CPU.

The difference between NBA and affine gap penalty is the block. NBA tries to maximize the $\tau^+ - blocks$ and minimize the $\tau^- - blocks$, which holds when the resulting alignment is close enough to optimum, but the latter focuses on the gapped blocks only. They may have the same result if gapped blocks can be seen as $\tau^- - blocks$. Usually the nucleic acid sequences are easy to do so by modifying the original scoring function, but protein sequences are not since some mismatches are worse than the gaps.

In this paper, we present a method to mark the tracings of all near-optimal alignments within d score compared to the optimal alignment. And then, we propose an algorithm to solve the near-optimal block alignment problem. Both the method and the algorithm can be implemented easily and efficiently. The filtering scheme can be replaced by any one mentioned by Tseng et al. [15] or other criteria easily.

The time complexity will remain the same if the criteria can be done in linear time.

The real biological sequence alignment is hard to find because we do not really know the correct scoring function of nature. The scoring functions presented by scientists may be close to the correct one, though. Thus we need to check all the near-optimal alignments to find the real one. It is too time consuming to check by human power. Our algorithm is a good choice to speed up our understanding of mysterious phenomena.

For now it is necessary to design different algorithms to filter the near-optimal alignments with different criteria. In the future, we would like to parameterize the problem and design the algorithm to solve it.

Acknowledgement

This research work was partially supported by the National Science Council of Taiwan under contract NSC-95-2221-E-110-084.

References

[1] O. Gotoh, "Optimal sequence alignment allowing for long gaps," Bulletin of Mathematical Biology, vol.52, pp.359–373, 1990.

[2] O. Gotoh, "An improved algorithm for matching biological sequences," J. Molecular Biology, vol.162, pp.705–708, 1982.

[3] W. Pearson and W. Miller, "Dynamic programming algorithms for biological sequence comparison," Methods in Enzymology, vol.210, pp.575–601, 1992.

[4] D.F. Feng, M.S. Johnson, and R.F. Doolittle, "Aligning amino acid sequences: Comparison of commonly used method's," J. Molecular Evolution, vol.21, pp.112–125, 1985.

[5] S. Altschul and B.W. Erickson, "Optimal sequence alignment using affine gap costs," J. Molecular Biology, vol.48, pp.603–616, 1986.

[6] A. Apostolico and C. Guerra, "The longest common subsequence problem revisited," Algorithmica, no.2, pp.315–336, 1987.

[7] L. Bergroth, H. Hakonen, and T. Raita, "A survey of longest common subsequence algorithms," Seventh International Symposium on String Processing Information Retrieval, pp.39–48, A Coruña, Spain, 2000.

[8] D.S. Hirschberg, "Algorithms for the longest common subsequence problem," J. ACM, vol.24, no.4, pp.664–675, 1977.

[9] J.W. Hunt and T.G. Szymanski, "A fast algorithm for computing longest common subsequences," Commun. ACM, vol.20, no.5, pp.350–353, 1977.

[10] C.B. Yang and R.C.T. Lee, "Systolic algorithms for the longest common subsequence problem," J. Chinese Institute of Engineers, vol.10, no.6, pp.691–699, 1987.

[11] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman, "Basic local alignment search tool," J. Molecular Biology, vol.215, pp.403–410, 1990.

[12] M.O. Dayhoff, Atlas of Protein Sequence and Structure, National Biomedical Research Foundation, Washington, DC, 1978.

[13] R.M. Schwartz and M.O. Dayhoff, Matrices for detecting distant relationships, National Biomedical Research Foundation, Washington, DC, 1979.

[14] D. Naor and D.L. Brutlag, "On near-optimal alignments of biological sequences," J. Computing Biology, vol.4, pp.349–366, 1994.

[15] K.T. Tseng, C.B. Yang, and K.S. Huang, "The better alignment among output alignments," Proc. 2005 International Conference on Mathematics and Engineering Techniques in Medecine and Biological Sciences, pp.31–37, Las Vegas, Nevada, USA, 2005.

[16] Y.Y. Chen, C.B. Yang, and K.T. Tseng, "Prediction of protein structures based on curve alignment," Proc. 20th Workshop on Combinatorial Mathematics and Computation Theory, pp.33–44, Chiayi, Taiwan, 2003.

Kuo-Tsung Tseng received his BS degree in Applied Mathematics from National Sun Yat-sen University, Kaohsiung, Taiwan, in 1997. He is now a PhD candidate in Computer Science and Engineering at National Sun Yat-sen University, Kaohsiung, Taiwan. His current research interests are computer algorithms, bioinformatics and wireless networks.

Chang-Biau Yang received the BS degree in electronic engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1982, and the MS degree in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 1984. Then, he received the PhD degree in computer science from National Tsing Hua University in 1988. He is currently a professor in the Department of Computer Science and Engineering, National Sun Yat-sen University. His research interests include computer algorithms, interconnection networks, and bioinformatics.

Kuo-Si Huang received his BS degree in Applied Mathematics from National Sun Yat-sen University, Kaohsiung, Taiwan, in 1997, and the PhD degree in Computer Science and Engineering at the same university in 2007. His current research interests are computer algorithms and bioinformatics.

Yung-Hsing Peng received his BS degree in Computer Science and Engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, in 2003, and then in 2004 he received his MS degree in the same university. He is now a PhD candidate in Computer Science and Engineering at National Sun Yat-sen University, Kaohsiung, Taiwan. His current research interests are computer algorithms, pattern matching and bioinformatics.