Finding Winner Alignments with Multiple Scoring Matrices [*]

Kuo-Tsung Tseng, Chang-Biau Yang, Yung-Hsing Peng and Chiou-Ting Tseng
Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan

Abstract

How to align two given sequences properly is a fundamental problem in bioinformatics. In the sequence alignment problem, the most essential thing that directly affects the resulting alignment is the scoring matrix. There are a variety of scoring matrices used for alignment, and each of them has its own purpose in biosequence alignment. It seems unlikely that an alignment is optimal for each scoring matrix. But, there may be one that meets the most matrices with acceptable scores.

In this paper, we present an efficient algorithm for finding the winner alignment when multiple scoring matrices are applied. We then discuss the variants of the comparing function. By simply reordering the steps in the comparing function, we could obtain another winner alignment under different criterion. Our algorithm solves the problem in $O(kmn)$ time where k is the number of scoring matrices involved, and m and n are the lengths of the two input sequences. A more efficient algorithm with $O(k(|\Sigma|+1)^2+mn)$ time can find the winner alignment under the summing scheme, where $|\Sigma|$ denotes the alphabet size of the input sequences.

1 Introduction

It is widely believed that in most cases, proteins with similar structures have similar functions, and similar biosequences share similar structures. In other words, a protein without known function could be understood if we could find a similar protein whose function has been known already. The issue now is the method of determining the similarity of two given biosequences. Thus, for many years, the comparison of biosequences [1, 6–8, 12, 14] has drawn a lot of attention. Actually, sequence comparison is usually regarded as the sequence alignment problem, which is a well studied problem in the algorithm area [3, 4, 9, 10, 16].

The essential in the sequence alignment problem is the scoring matrix. It is easy to find an optimal alignment if a proper measuring scheme is given. A variety of scoring matrices [2, 5, 13] were proposed, but none of them is completely successful. It seems that for any kind of scoring matrices, there always exist some biosequences with lower scores but higher similarities (judged by biologists or experiments).

All the scoring matrices PAMs, Blosums and Gonnets have their own purposes in biosequence alignment. It seems unlikely that they would share the same optimal alignment. How far should we step backward to have an acceptable common alignment in all scoring matrices applied?

In 1994, Naor and Brutlag presented the idea of the near optimal alignment and they showed that the alignment with optimal score is not always the most biologically meaningful one [11]. Their result implies that an alignment of acceptable scores, i.e. near-optimal alignment, in all scoring matrices is meaningful. Thus, such an alignment should be focused further.

In this paper, a useful feature [15] of the alignment lattice is proposed to calculate the losing score lattice. With these losing score lattices of several scoring matrices, it is not difficult to find the *winner alignment* when k scoring matrices is applied. Conceptually, a *winner alignment* refers to the best alignment (under some user-specified mechanisms) which fits the most scoring matrices . This idea is particularly useful when we have no idea which scoring matrix is the most believable.

Our method proposed in this paper for finding the winner alignment can be easily implemented. Its time complexity is $O(kmn)$ where $m,\ n$ are lengths of the two input sequences, respectively and k is the number of scoring matrices that we applied. It can be reduced to $O(mn)$ when the alphabet size of the input sequences and k are much smaller than $m,\ n$ under the summing scheme.

The rest of this paper is organized as follows. In Section 2, we give the useful feature to calculate the losing score lattice. Next, we will propose our method for finding the winner alignment in Section 3. Then in Section 4, we will give the analysis of comparing

[*]Correspondence e-mail: cbyang@cse.nsysu.edu.tw (Chang-Biau Yang).

Table 1: Three scoring matrices of {a,b,c,d}.

SM_1	-	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

SM_2	-	a	b	c	d
-	$-\infty$	-1	-1	-1	-1
a	-1	4	-2	0	2
b	-1	-2	3	0	3
c	-1	0	0	2	-2
d	-1	2	3	-2	5

SM_3	-	a	b	c	d
-	$-\infty$	-1	-1	-1	-1
a	-1	1	2	0	2
b	-1	2	1	0	1
c	-1	0	0	1	3
d	-1	2	1	3	1

function, and further propose an easier way to obtain the winner alignment under the summing scheme. Finally, some discussions and conclusions will be given in Section 5.

2 Preliminaries

In this section, we demonstrate how we calculate the losing score lattice with a simple example. Let S and T be the two input sequences, where $|S| = m$ and $|T| = n$. $S =$ `abdcd` and $T =$ `bacddb` are given to be aligned with the scoring matrix SM_1 shown in Table 1. We then have the alignment lattice AL_1 of sequences S and T, as shown in Figure 1, after the traditional alignment scheme is performed. [3,4,9,10,16].

The bold lines in Figure 1 represent the correct alignments of getting optimal scores. The numbers beside lines are the costs of alignments. To obtain the optimal alignment, one can trace from the lower right corner back to the upper left corner in AL_1 [9]. In our example, there are two optimal alignments $\frac{\texttt{abdcd--}}{\texttt{-bacddb}}$ and $\frac{\texttt{abdc-d-}}{\texttt{-bacddb}}$.

It is easy to find the optimal alignment of two given sequences, but to find the winner alignment we need an efficient way to calculate the difference between the optimal alignment and any other alignment. To do so, a special lattice of the same size as the alignment lattice,

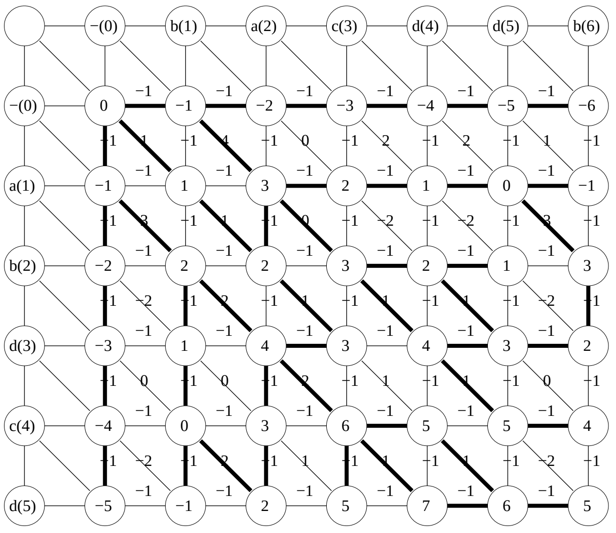

Figure 1: The alignment lattice AL_1 of sequences `abdcd` and `bacddb` with the scoring matrix SM_1 shown in Table 1.

the *Losing Score Lattice*, is used in our method.

For an alignment lattice computed with S and T, let $P = (i, j, U)$ denote the operation which aligns S_i and T_j via direction U, where $0 \le i \le m$, $0 \le j \le n$, and $U \in \{h, d, v\}$. Here h, d and v represent the *horizontal*, *diagonal*, and *vertical* directions, respectively. In addition, we define the δ function to calculate the effect when the operation P, with respect to the scoring matrix SM and alignment lattice AL, is chosen as follows.

$$\delta(P) = AL(i,j) + \begin{cases} -AL(i,j-1) - SM(-,T_j) & \text{if } U = h, \\ -AL(i-1,j-1) - SM(S_i,T_j) & \text{if } U = d, \\ -AL(i-1,j) - SM(S_i,-) & \text{if } U = v, \end{cases}$$

where S_i and T_j represent the ith and jth characters of S and T, 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 some examples to illustrate the usage of $\delta(P)$. Take Figure 1 for example, $P = (5,6,v)$ yields a nonoptimal alignment. Clearly, in this case we have $\delta(P) = \delta(5,6,v) = AL(5,6) + AL(5-1,6) - SM(d,-) = 5-4-(-1) = 2$. Suppose P is chosen and afterward we follow the correct alignments, i.e. bold lines, from position (4, 6) till position (0, 0). Then, the score of the obtained nonoptimal alignment would be *optimal score* $- \delta(P) = 5-2 = 3$. As another example, we first follow the bold lines from position (5, 6) till position (4, 4), then $P = (4,4,d)$ is chosen together with bold lines from position (3, 3) till position (0, 0). In this case, we have $\delta(P) = \delta(4,4,d) = AL(4,4) + AL(4-1,4-1) - SM(c,d) =$

$5-3-1=1$. This operation yields an alignment with score $5-\delta(P)=5-1=4$. Similarly, it can be easily verified that $\delta(4,4,h)=0$ and $\delta(4,4,v)=2$, which result in the alignment score $5-0=5$ and $5-2=3$, respectively.

It is easy to prove that an optimal alignment can be constructed if and only if each operation P in this alignment satisfies $\delta(P)=0$. Note that the losing score $\delta(P)$ is accumulative. This means if a series of operations $\hat{P}=P_1P_2\cdots P_t$ are chosen, then the difference between the optimal alignment and $\hat{P}$ would be $\sum_{s=1}^{t}\delta(P_s)$.

Hence, the losing score lattice can be constructed by considering each $\delta(P)$ in AL. It is clear that the time complexity for constructing the losing score lattice is $O(3mn)=O(mn)$, which turns out to be $O(kmn)$ if k scoring matrices are applied. In this paper, 3 scoring matrices listed in Table 1 are used and Table 2 shows their losing score lattices (LSLs).

3 Finding the Winner Alignment

We propose an algorithm for finding the winner alignment in this section. Here we adopt a *Voting Lattice* (VL) of size $(m+1)\times(n+1)$. Each element at position (i,j) in VL is a k-dimensional vector that records the minimum losing scores of the k scoring matrices, which is obtained by tracing in LSL from position (m,n) back to position (i,j).

Algorithm: Finding the Winner Alignment (FWA)

Input: Losing score lattices LSL_z, where $1\le z\le k$.

Output: The winner alignment which fits the most scoring matrices.

Step 0:(Initialization) $VL(m,n)=\begin{vmatrix}0\\0\\0\end{vmatrix}$.

Step 1: In VL, it is clear that the value at position (m,j), $0\le j\le n-1$, can be obtained only from the horizontal direction and the value at position (i,n), $0\le i\le m-1$, can be obtained only from the vertical direction. Since $\delta(P)$ is accumulative, we have the following formula.

$$VL(m,j)=VL(m,j+1)+\begin{vmatrix}\delta_1(m,j+1,h)\\ \vdots\\ \delta_k(m,j+1,h)\end{vmatrix},$$
where $0\le j\le n-1$.

$$VL(i,n)=VL(i+1,n)+\begin{vmatrix}\delta_1(i+1,n,v)\\ \vdots\\ \delta_k(i+1,n,v)\end{vmatrix},$$
where $0\le i\le m-1$.

For example,
$$VL(5,2)=VL(5,2+1)+\begin{vmatrix}\delta_1(5,2+1,h)\\ \delta_2(5,2+1,h)\\ \delta_3(5,2+1,h)\end{vmatrix}$$
$$=\begin{vmatrix}3\\10\\8\end{vmatrix}+\begin{vmatrix}4\\4\\5\end{vmatrix}=\begin{vmatrix}7\\14\\13\end{vmatrix}.$$
$$VL(2,6)=VL(2+1,6)+\begin{vmatrix}\delta_1(2+1,6,v)\\ \delta_2(2+1,6,v)\\ \delta_3(2+1,6,v)\end{vmatrix}$$
$$=\begin{vmatrix}5\\5\\8\end{vmatrix}+\begin{vmatrix}0\\7\\5\end{vmatrix}=\begin{vmatrix}5\\12\\13\end{vmatrix}.$$

Step 2: $VL(i,j)=$
$$min\begin{cases}VL(i,j+1) & + \begin{vmatrix}\delta_1(i,j+1,h)\\ \vdots\\ \delta_k(i,j+1,h)\end{vmatrix},\\ VL(i+1,j+1) & + \begin{vmatrix}\delta_1(i+1,j+1,d)\\ \vdots\\ \delta_k(i+1,j+1,d)\end{vmatrix},\\ VL(i+1,j) & + \begin{vmatrix}\delta_1(i+1,j,v)\\ \vdots\\ \delta_k(i+1,j,v)\end{vmatrix},\end{cases}$$
where $0\le i\le m-1$, $0\le j\le n-1$.

To pick the smallest vector, the comparing function ($\prec$) for a pair of vectors $X=\begin{vmatrix}x_1\\ \vdots\\ x_k\end{vmatrix}$ and $Y=\begin{vmatrix}y_1\\ \vdots\\ y_k\end{vmatrix}$ is defined as follows, where the comparing function proceeds until $min(X,Y)$ is determined.

I.Voting Mechanism: If $x_i<y_i$, where $1\le i\le k$, in most of i's, then $min(X,Y)=X$, denoted as $X\prec Y$. If $min(X,Y)$ cannot be determined, go to II.

II.Summing Model: $X\prec Y$ if $\sum_{i=1}^{k}x_i<\sum_{i=1}^{k}y_i$. If $min(X,Y)$ cannot be determined, go to III.

III.Minimax Decision: $X\prec Y$ if $max_{1\le i\le k}\,x_i<max_{1\le i\le k}\,y_i$. If $min(X,Y)$ cannot be determined, go to IV.

Table 2: The losing score lattice LSL_z of sequences abdcd and bacddb with the scoring matrix SM_z shown in Table 1, where $1 \leq z \leq 3$. Numbers in ( ) represent the $\delta_z(i,j,h),\ \delta_z(i,j,d) and\ \delta_z(i,j,v)$ of SM_z at position (i,j).

LSL_1	-(0)	b(1)	a(2)	c(3)	d(4)	d(5)	b(6)
-(0)	0(∞,∞,∞)	-1(0,∞,∞)	-2(0,∞,∞)	-3(0,∞,∞)	-4(0,∞,∞)	-5(0,∞,∞)	-6(0,∞,∞)
a(1)	-1(∞,∞,0)	1(3,0,3)	3(3,0,6)	2(0,4,6)	1(0,2,6)	0(0,2,6)	-1(0,3,6)
b(2)	-2(∞,∞,0)	2(5,0,2)	2(1,0,0)	3(2,0,2)	2(0,2,2)	1(0,2,2)	3(3,0,5)
d(3)	-3(∞,∞,0)	1(5,5,0)	4(4,0,3)	3(0,0,1)	4(2,0,3)	3(0,0,3)	2(0,3,0)
c(4)	-4(∞,∞,0)	0(5,3,0)	3(4,2,0)	6(4,0,4)	5(0,1,2)	5(1,0,3)	4(0,1,3)
d(5)	-5(∞,∞,0)	-1(5,5,0)	2(4,0,0)	5(4,1,0)	7(3,0,3)	6(0,0,2)	5(0,2,2)

LSL_2	-(0)	b(1)	a(2)	c(3)	d(4)	d(5)	b(6)
-(0)	0(∞,∞,∞)	-1(0,∞,∞)	-2(0,∞,∞)	-3(0,∞,∞)	-4(0,∞,∞)	-5(0,∞,∞)	-6(0,∞,∞)
a(1)	-1(∞,∞,0)	-2(0,0,0)	3(6,0,6)	2(0,4,6)	1(0,2,6)	0(0,2,6)	-1(0,6,6)
b(2)	-2(∞,∞,0)	2(5,0,5)	2(1,6,0)	3(2,0,2)	5(3,0,5)	4(0,0,5)	3(0,0,5)
d(3)	-3(∞,∞,0)	1(5,0,0)	4(4,0,3)	3(0,3,1)	8(6,0,4)	10(3,0,7)	9(0,2,7)
c(4)	-4(∞,∞,0)	0(5,3,0)	3(4,2,0)	6(4,0,4)	7(2,6,0)	9(3,3,0)	10(2,0,2)
d(5)	-5(∞,∞,0)	-1(5,0,0)	2(4,0,0)	5(4,4,0)	11(7,0,5)	12(2,0,4)	12(1,0,3)

LSL_3	-(0)	b(1)	a(2)	c(3)	d(4)	d(5)	b(6)
-(0)	0(∞,∞,∞)	-1(0,∞,∞)	-2(0,∞,∞)	-3(0,∞,∞)	-4(0,∞,∞)	-5(0,∞,∞)	-6(0,∞,∞)
a(1)	-1(∞,∞,0)	2(4,0,4)	1(0,1,4)	0(0,2,4)	-1(0,0,4)	-2(0,0,4)	-3(0,0,4)
b(2)	-2(∞,∞,0)	1(4,1,0)	4(4,0,4)	3(0,2,4)	2(0,1,4)	1(0,1,4)	0(0,1,4)
d(3)	-3(∞,∞,0)	0(4,1,0)	3(4,0,0)	7(5,0,5)	6(0,2,5)	5(0,2,5)	4(0,2,5)
c(4)	-4(∞,∞,0)	-1(4,2,0)	2(4,2,0)	6(5,2,0)	10(5,0,5)	9(0,0,5)	8(0,3,5)
d(5)	-5(∞,∞,0)	-2(4,1,0)	1(4,0,0)	5(5,0,0)	9(5,2,0)	11(3,0,3)	10(0,0,3)

IV.Priority Judgment: $X \prec Y$ if $x_i < y_i$ and SM_i has higher priority than any other SM_j that $x_j > y_j$. If $min(X,Y)$ cannot be determined, go to V.

V.: Randomly choose either X or Y.

Step 3: Trace back from $VL(0,0)$ to $VL(m,n)$ to obtain the winner alignment.

The proof for our FWA algorithm is omitted and left to the full version of this paper. Here, we only demonstrate FWA with an example. The constructed VL with FWA is listed in Table 3. In Table 3, one can see that $VL(0,0) = \begin{vmatrix} 1 \\ 16 \\ 0 \end{vmatrix}$, which means that there exists at least one alignment which is an optimal alignment for SM_3, with score $optimum - 1$ in SM_1, and distance 16 from the optimum in SM_2. In this example, the winner alignment is $\frac{\texttt{abdcd-}}{\texttt{bacddb}}$, and its alignment scores for SM_1, SM_2, SM_3 are 4, -4 and 10, respectively.

4 Variants of the Comparing Function

It is interseting that if we change the steps order in our comparing function (most-fit scheme), we will get another winner alignment with different criterion. For example, to minimize the total losing scores in all score matrices, one can simply raise the summing model to the first step in the comparing function. As another example, to find an alignment which must be optimal for SM_1, and get the maximum score in SM_2, then SM_3, then $\cdots$, and finally SM_k, one can make the priority judgment step be primary. Note that when the priority judgment is primary, other comparing steps will become redundant, since there is no way that the priority judgment could possibly fail.

One thing should be noticed is that the first step of the comparing function ($\prec$) must be order preserving to ensure the correctness of our method. That is,

$$\forall \text{ X, Y, Z} \in \begin{vmatrix} v_1 \\ \vdots \\ v_k \end{vmatrix}, \text{ where } v_1, \cdots, v_k \in \{0\} \cup \mathcal{N},$$
$$\text{if } X \prec Y,\ (X+Z) \prec (Y+Z).$$

In our comparing functions, voting mechanism,

Table 3: The voting lattice VL using the most-fit/summing scheme for sequences `abdcd` and `bacddb` with the *LSLs* shown in Table 2. The values of these two models in the voting lattice are almost the same except those cells with tiny size fonts, which indicate the values for the summing scheme when they do not share the same value. The winner alignment of the most-fit scheme is shown by the arrows without boxes . The boxed-arrows show the way to win in the summing scheme.

VL	-(0)	b(1)	a(2)	c(3)	d(4)	d(5)	b(6)
-(0)	$\begin{matrix}1\\16\\0\end{matrix}$ $\left\|\begin{smallmatrix}2\\3\\5\end{smallmatrix}\right\|$	$\begin{matrix}2\\3\\5\end{matrix}$ $\boxed{\leftarrow}$	$\begin{matrix}9\\7\\5\end{matrix}$	$\begin{matrix}7\\10\\6\end{matrix}$	$\begin{matrix}11\\14\\10\end{matrix}$	$\begin{matrix}11\\18\\18\end{matrix}$	$\begin{matrix}16\\23\\21\end{matrix}$
a(1)	$\begin{matrix}0\\3\\8\end{matrix}$	$\nwarrow$ $\begin{matrix}1\\16\\0\end{matrix}$	$\begin{matrix}2\\3\\4\end{matrix}$ $\boxed{\nwarrow}$	$\begin{matrix}5\\3\\3\end{matrix}$	$\begin{matrix}5\\8\\6\end{matrix}$	$\begin{matrix}5\\12\\14\end{matrix}$	$\begin{matrix}10\\17\\17\end{matrix}$
b(2)	$\begin{matrix}5\\8\\11\end{matrix}$ $\left\|\begin{smallmatrix}11\\9\\3\end{smallmatrix}\right\|$	$\begin{matrix}0\\3\\7\end{matrix}$	$\nwarrow$ $\begin{matrix}1\\10\\0\end{matrix}$	$\begin{matrix}2\\3\\2\end{matrix}$ $\boxed{\nwarrow}$	$\begin{matrix}3\\3\\2\end{matrix}$	$\begin{matrix}6\\10\\5\end{matrix}$	$\begin{matrix}5\\12\\13\end{matrix}$
d(3)	$\begin{matrix}11\\14\\6\end{matrix}$	$\begin{matrix}6\\9\\2\end{matrix}$	$\begin{matrix}0\\3\\7\end{matrix}$	$\nwarrow$ $\begin{matrix}1\\7\\0\end{matrix}$	$\begin{matrix}2\\3\\0\end{matrix}$ $\boxed{\nwarrow}$	$\begin{matrix}3\\3\\0\end{matrix}$	$\begin{matrix}5\\5\\8\end{matrix}$
c(4)	$\begin{matrix}12\\19\\17\end{matrix}$ $\left\|\begin{smallmatrix}13\\16\\18\end{smallmatrix}\right\|$	$\begin{matrix}7\\14\\13\end{matrix}$ $\left\|\begin{smallmatrix}8\\11\\14\end{smallmatrix}\right\|$	$\begin{matrix}4\\7\\10\end{matrix}$	$\begin{matrix}0\\3\\5\end{matrix}$	$\nwarrow$ $\begin{matrix}0\\1\\0\end{matrix}$	$\begin{matrix}2\\0\\0\end{matrix}$ $\boxed{\nwarrow}$	$\begin{matrix}2\\3\\3\end{matrix}$
d(5)	$\begin{matrix}16\\23\\21\end{matrix}$	$\begin{matrix}11\\18\\17\end{matrix}$	$\begin{matrix}7\\14\\13\end{matrix}$	$\begin{matrix}3\\10\\8\end{matrix}$	$\begin{matrix}0\\3\\3\end{matrix}$	$\nwarrow$ $\begin{matrix}0\\1\\0\end{matrix}$	$\leftarrow$ $\begin{matrix}0\\0\\0\end{matrix}$ $\boxed{\nwarrow}$

summing model and priority judgment are order preserving, but minimax decision is not. Consider the following example.

If $X = \left|\begin{matrix}2\\2\\2\end{matrix}\right|$, $Y = \left|\begin{matrix}1\\3\\1\end{matrix}\right|$ and $Z = \left|\begin{matrix}2\\0\\2\end{matrix}\right|$,

we have $max(X) = 2 < max(Y) = 3$. Thus $X \prec Y$.

$X' = X+Z = \left|\begin{matrix}4\\2\\4\end{matrix}\right|$, $Y' = Y+Z = \left|\begin{matrix}3\\3\\3\end{matrix}\right|$.

$max(X') = 4 > max(Y') = 3$. Thus $X' \succ Y'$.

Though the minimax decision cannot be the first step of the comparing function, it has no problem to be used in any of the following steps. Once the order preserving property is ensured, each winner would always conform to the rule of the first step.

For the special case that only the summing model is applied, called the *summing scheme*, we propose an algorithm with $O(k(|\Sigma|+1)^2+mn)$ time to find the winner alignment, where $|\Sigma|$ denotes the alphabet size of the input sequences.

Lemma 1 *Let $SM' = SM_1 + SM_2 + \cdots + SM_k$, any optimal alignment for SM' is a winner alignment for the summing scheme.*

Proof: Suppose $R = \dfrac{s_1^1, s_2^1, \cdots, s_r^1}{s_1^2, s_2^2, \cdots, s_r^2}$, whose length is r, is an arbitrary alignment of two sequences S and T. Let Opt_i be the optimal score for aligning S and T with SM_i, and let M_i be the alignment score of R in SM_i, where $1 \le i \le k$. We then have $M_i = SM_i(s_1^1, s_1^2) + SM_i(s_2^1, s_2^2) + \cdots + SM_i(s_r^1, s_r^2)$.

According to the summing scheme, the total losing score of R is $\sum_{i=1}^k (Opt_i - M_i)$. Therefore, the total losing score for any winner alignment can be written as $min_R(\sum_{i=1}^k (Opt_i - M_i))$=$\sum_{i=1}^k Opt_i - max_R(\sum_{i=1}^k M_i)$.

This implies that any winner alignment must have its total alignment score, $(\sum_{i=1}^k M_i)$, be maximized. Manipulating the formula, we have $max_R(\sum_{i=1}^k M_i)$
$=max_R(\sum_{i=1}^k SM_i(s_1^1, s_1^2) + \sum_{i=1}^k SM_i(s_2^1, s_2^2) + \cdots$
$+ \sum_{i=1}^k SM_i(s_r^1, s_r^2))$
$=max_R(SM'(s_1^1, s_1^2) + SM'(s_2^1, s_2^2) + \cdots + SM'(s_r^1, s_r^2))$.

Therefore, the optimal alignment for the scoring matrix SM' is the demanded winner alignment. $\square$

Here we give the new scoring matrix $SM' = SM_1 + SM_2 + SM_3$ of our previous example in Table 4, and keep the losing scores in the lattice LSL', as shown in Table 5. In this example, one can see that the traditional alignment lattice suffices for determining the winner alignment. One can take the LSL' in Table 5 as

Table 4: A new scoring matrix SM' of $\{$a,b,c,d$\}$.

SM'	-	a	b	c	d
-	$-\infty$	-1	-1	-1	-1
a	-1	9	1	0	6
b	-1	1	7	0	2
c	-1	0	0	5	2
d	-1	6	2	2	7

a contrast with either Table 2 or Table 3.

The VL under the summing scheme is listed in Table 3, using tiny size fonts if the corresponding value is different from that of the most-fit scheme. One can see that $VL(0,0) = \begin{vmatrix} 2 \\ 3 \\ 5 \end{vmatrix}$, which shows that there exists at least one alignment which is with score $opt_1 - 2$ in SM_1, $opt_2 - 3$ in SM_2, and $opt_3 - 5$ in SM_3. In our example, the winner alignment is $\frac{\texttt{-abdcd}}{\texttt{bacddb}}$, and its scores in SM_1, SM_2 and SM_3 are 3, 9 and 5, respectively.

5 Conclusion

In this paper, we present a simple algorithm to find the winner alignment when multiple scoring matrices are involved. For the most-fit scheme, the time complexity of our algorithm is $O(kmn)$, where m and n are the lengths of the two input sequences, and k is the number of scoring matrices applied. For the summing scheme, we also show that the required time can be reduced to $O(k(|\Sigma|+1)^2 + mn)$, which would be $O(mn)$ for the most cases. These two algorithms are both time-efficient and code-efficient. In addition, by changing the order of our comparing function steps, one can obtain another result with different criterion, which shows the flexibility of our scheme.

In the future, we will consider other meaningful mechanisms which can be added into the comparing function. It would be interesting to study their effects on both time and space complexity.

References

[1] S. Altschul and B. W. Erickson, “Optimal sequence alignment using affine gap costs,” *Journal of Molecular Biology*, Vol. 48, pp. 603–616, 1986.

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, “Basic local alignment search tool,” *Journal of Molecular Biology*, Vol. 215, pp. 403–410, 1990.

[3] A. Apostolico and C. Guerra, “The longest common subsequence problem revisited,” *Algorithmica*, No. 2, pp. 315–336, 1987.

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

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

[6] D. F. Feng, M. S. Johnson, and R. F. Doolittle, “Aligning amino acid sequences: comparison of commonly used method s,” *Journal of Molecular Evolution*, Vol. 21, pp. 112–125, 1985.

[7] O. Gotoh, “An improved algorithm for matching biological sequences,” *Journal of Molecular Biology*, Vol. 162, pp. 705–708, 1982.

[8] O. Gotoh, “Optimal sequence alignment allowing for long gaps,” *Bulletin of Mathematical Biology*, Vol. 52, pp. 359–373, 1990.

[9] D. S. Hirschberg, “Algorithms for the longest common subsequence problem,” *Journal of the ACM*, Vol. 24, No. 4, pp. 664–675, 1977.

[10] J. W. Hunt and T. G. Szymanski, “A fast algorithm for computing longest common subsequences,” *Communications of the ACM*, Vol. 20, No. 5, pp. 350–353, 1977.

[11] D. Naor and D. L. Brutlag, “On near-optimal alignments of biological sequences,” *Journal of Computing Biology*, Vol. 4, pp. 349–366, 1994.

[12] W. Pearson and W. Miller, “Dynamic programming algorithms for biological sequence comparison,” *Methods in Enzymology*, Vol. 210, pp. 575–601, 1992.

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

[14] K. T. Tseng, C. B. Yang, and K. S. Huang, “The better alignment among output alignments,” *Journal of Computers*, Vol. 3, pp. 51–62, 2007.

Table 5: The losing score lattice LSL' of sequences abdcd and bacddb with the scoring matrix SM' shown in Table 4. Numbers in ( ) represent the $\delta'(i,j,h),\ \delta'(i,j,d), and\ \delta'(i,j,v)$ of SM' at position (i,j).

LSL'	-(0)	b(1)	a(2)	c(3)	d(4)	d(5)	b(6)
-(0)	0(∞,∞,∞)	←-3(0,∞,∞)	-6(0,∞,∞)	-9(0,∞,∞)	-12(0,∞,∞)	-15(0,∞,∞)	-18(0,∞,∞)
a(1)	-3(∞,∞,0)	1(7,0,7)	↖6(8,0,15)	3(0,9,15)	0(0,3,15)	-3(0,3,15)	-6(0,8,15)
b(2)	-6(∞,∞,0)	4(13,0,6)	3(2,1,0)	↖6(6,0,6)	5(2,0,8)	2(0,0,8)	4(5,0,13)
d(3)	-9(∞,∞,0)	1(13,5,0)	10(12,0,10)	7(0,2,4)	↖13(9,0,11)	12(2,0,13)	9(0,5,8)
c(4)	-12(∞,∞,0)	-2(13,7,0)	7(12,6,0)	15(11,0,11)	12(0,3,2)	↖15(6,0,6)	12(0,0,6)
d(5)	-15(∞,∞,0)	-5(13,5,0)	4(12,0,0)	12(11,3,0)	22(13,0,13)	19(0,0,7)	↖17(1,0,8)

[15] K. T. Tseng, C. B. Yang, K. S. Huang, and Y. H. Peng, "Near-optimal block alignments," *IEICE TRANSACTIONS on Information and Systems*, Vol. E91-D, No. 3, pp. 789–795, 2008.

[16] C. B. Yang and R. C. T. Lee, "Systolic algorithms for the longest common subsequence problem," *Journal of the Chinese Institute of Engineers*, Vol. 10, No. 6, pp. 691–699, 1987.