Available online at www.sciencedirect.com

ScienceDirect

Information Processing Letters 105 (2008) 188–193

Information Processing Letters

www.elsevier.com/locate/ipl

# Efficient algorithms for finding interleaving relationship between sequences ☆

Kuo-Si Huang, Chang-Biau Yang *, Kuo-Tsung Tseng, Hsing-Yen Ann, Yung-Hsing Peng

*Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung 80424, Taiwan*

Received 14 February 2007; received in revised form 25 July 2007; accepted 23 August 2007

Available online 12 September 2007

Communicated by L. Boasson

**Abstract**

The longest common subsequence and sequence alignment problems have been studied extensively and they can be regarded as the relationship measurement between sequences. However, most of them treat sequences evenly or consider only two sequences. Recently, with the rise of whole-genome duplication research, the doubly conserved synteny relationship among three sequences should be considered. It is a brand new model to find a merging way for understanding the interleaving relationship of sequences. Here, we define the merged LCS problem for measuring the interleaving relationship among three sequences. An $O(n^3)$ algorithm is first proposed for solving the problem, where $n$ is the sequence length. We further discuss the variant version of this problem with the block information. For the blocked merged LCS problem, we propose an algorithm with time complexity $O(n^2m)$, where $m$ is the number of blocks. An improved $O(n^2 + nm^2)$ algorithm is further proposed for the same blocked problem.
© 2007 Published by Elsevier B.V.

*Keywords:* Design of algorithms; Bioinformatics; Dynamic programming; Longest common subsequence; Merged sequence

## 1. Introduction

A *subsequence* $P$ of a sequence $S$, over a finite symbol set $\Sigma$, is a sequence obtained from $S$ by deleting zero or more (not necessarily contiguous) symbols. The *longest common subsequence* (LCS) problem is defined as follows: Given $k$ ($k \geqslant 2$) sequences, find a longest sequence $P$ such that $P$ is a common subsequence of these $k$ sequences. If the number of input sequences, $k$, is not fixed, the problem is NP-complete even over binary alphabets [14]. In general, many researches and applications focused on the LCS problem of only two input sequences [5,19]. The LCS problem is a famous and classical problem in computer science and molecular biology [6]. The common subsequences can be viewed as the identical parts of the input sequences, that can help us reconstruct an alignment of these sequences. There are several variant LCS problems for specific applications such as with fragments, constraint sequences, or increasing subsequences [2,4,16,20].

Let us consider a special relationship of sequences for playing the poker card. The riffle shuffle, which is a shuffle trick when we play poker cards, merges two

☆ This research work was partially supported by the National Science Council of Taiwan under NSC-94-2213-E-110-051.
* Corresponding author.
*E-mail address:* cbyang@cse.nsysu.edu.tw (C.-B. Yang).

0020-0190/$ – see front matter © 2007 Published by Elsevier B.V.
doi:10.1016/j.ipl.2007.08.028

halves of a given deck of cards into one interleaving deck. If only one shuffle is performed, it is easy to find the mapping relation between the interleaved deck and the two original halves. We can observe that the card order of each original half must be conserved unbrokenly in the interleaved deck. It is interesting that the riffle shuffle phenomenon exists in the genomes of organisms, but it is more complex than that in poker cards. The complexity rises due to the existence of identical cards, the missing cards of the deck and the original halves, and the clones of some cards, that are corresponding to identical symbols, gene loss, and gene duplications in genomes, respectively. The term *synteny* means that the order of specific genes in the chromosome is conserved over different organisms [10,9].

Recently, the interleaving relationship of sequences is shown in the genomic comparison of two yeast species *Kluyveromyces waltii* and *Saccharomyces cerevisiae* [10]. Kellis et al. tried to find the doubly conserved synteny (DCS) blocks of the two species, in which each region of *K. waltii* is associated with two regions of *S. cerevisiae*, as the support for the whole genome duplication (WGD) or two rounds of gene duplication (2R) hypothesis [7,10,15]. In addition, Jaillon et al. proposed the genomic comparison of human and *Tetraodon*, a teleost fish, chromosomes [9]. The synteny blocks and the WGD are also available in the genomic comparison of human and *Tetraodon*. For finding the DCS blocks, we can use local alignment [18] to identify the relationship of similar regions, or try to locate known genes on chromosomes. But this approach is a working heuristic and it may not be computationally optimal.

In this paper, we define the merged LCS problem, denoted as $LCS(T, E(A, B))$, to measure the interleaving relationship among sequences $T$, $A$ and $B$. The merged sequence $E(A, B)$ is composed of $A$ and $B$ interlacedly. That is, $E(A, B)$ can be separated into two disjoint subsequences of $A$ and $B$. For understanding the relationship among sequences $T$, $A$ and $B$, we can first merge $A$ and $B$ to a specific interleaving sequence $E(A, B)$; then find the relationship, such as LCS or alignment, of $T$ and $E(A, B)$. This problem is not so trivial because there are many various sequences of $E(A, B)$ based on different interleaving combinations of $A$ and $B$. Note that in this paper, $LCS(A, B)$ may be used to represent the length value of $LCS(A, B)$ or the string of $LCS(A, B)$ alternatively if there is no ambiguity of its meaning.

## 2. The merged LCS problem

Given a target sequence $T = t_1t_2\cdots t_{|T|}$, and a pair of merging sequences $A = a_1a_2\cdots a_{|A|}$ and $B = b_1b_2\cdots b_{|B|}$, where $|T|$, $|A|$ and $|B|$ denote the lengths of $T$, $A$ and $B$, respectively. We regard sequence lengths as $O(n)$ for simplicity. A merged sequence $E(A, B) = e_1e_2e_3\cdots e_{|E(A,B)|}$ is obtained by merging two subsequences of $A$ and $B$, where $e_i \in \{a_j \mid 1 \leqslant j \leqslant |A|\} \cup \{b_k \mid 1 \leqslant k \leqslant |B|\}$ and $1 \leqslant i \leqslant |E(A, B)|$; that is, there exists one subsequence $S$ of $E(A, B)$ such that $S$ is a subsequence of $A$ and $E(A, B) - S$ is a subsequence of $B$. The merged LCS (MLCS) problem is to find the LCS of $T$ and $E(A, B)$, denoted by

$$LCS\big(T, E(A, B)\big) = LCS\big(t_1t_2\cdots t_{|T|}, E(a_1a_2\cdots a_{|A|}, b_1b_2\cdots b_{|B|})\big).$$

Note that based on the definition, the interleaving sequence $E(A, B)$ may not be unique.

For example, consider sequences $T =$ `atacgcgctt`, $A =$ `cgatacc`, and $B =$ `aattcgc`. $E_1(A, B) =$ `cgataaacgc`, $E_2(A, B) =$ `aattcgacgctacc`, and $E_3(A, B) =$ `cgaaatactcgc` are some merged sequences of $A$ and $B$. We can find that $LCS(T, E_1(A, B)) =$ `atacgc`, $LCS(T, E_2(A, B)) =$ `aacgcgct`, $LCS(T, E_3(A, B)) =$ `ataccgc`. In this example,

$$LCS\big(T, E(A, B)\big) = LCS\big(T, E_2(A, B)\big)$$

with length 8.

We may design a simple heuristic for solving the MLCS problem. One may first find $LCS(T, A)$ and $LCS(T - LCS(T, A), B)$, and then merge them into a solution candidate. Another candidate is the merged sequence of $LCS(T, B)$ and $LCS(T - LCS(T, B), A)$. However, this heuristic may miss the optimal solution. One can easily examine one counter example with sequences $T_c =$ `aaaggccctt`, $A_c =$ `ggaaacc`, and $B_c =$ `aattccc`.

Here, we propose a dynamic programming algorithm for solving the MLCS problem as follows. For abbreviating the expressions, let

$$L(i, j, k) = LCS\big(t_1t_2\cdots t_i, E(a_1a_2\cdots a_j, b_1b_2\cdots b_k)\big).$$

**Algorithm MergedLCS**

*Input*: A target sequence $T$, and two merging sequences $A$ and $B$.
*Output*: $LCS(T, E(A, B)) = L(|T|, |A|, |B|)$.
*Step* 1. Initialize $L(0, j, k) = 0$ and compute $L(i, j, 0)$ and $L(i, 0, k)$ and by the dynamic programming approach for $LCS(T, A)$ and $LCS(T, B)$, for $0 \leqslant i \leqslant |T|$, $0 \leqslant j \leqslant |A|$ and $0 \leqslant k \leqslant |B|$.

*Step* 2. For $1 \leqslant i \leqslant |T|$, $1 \leqslant j \leqslant |A|$ and $1 \leqslant k \leqslant |B|$, calculate $L(i, j, k)$ with the following dynamic programming formula.

$$L(i,j,k) = \max \begin{cases} L(i-1, j-1, k) + 1 & \text{if } t_i = a_j, \\ L(i-1, j, k-1) + 1 & \text{if } t_i = b_k, \\ \max \begin{cases} L(i-1, j, k) \\ L(i, j-1, k) \\ L(i, j, k-1) \end{cases} & \text{if } t_i \neq a_j \text{ and } t_i \neq b_k. \end{cases} \tag{1}$$

*Step* 3. Return $LCS(T, E(A, B)) = L(|T|, |A|, |B|)$.

**Theorem 1.** *Algorithm MergedLCS solves the MLCS problem in $O(|T||A||B|)$ time and $O(|A||B|)$ space.*

**Proof.** We shall prove the correctness of Algorithm MergedLCS by induction. One can easily verify the correctness of the boundary cases in Step 1 and the base case $L(1, 1, 1)$ in Step 2. For considering $L(i', j', k')$, suppose that $L(i, j, k)$ is optimal for all $i \leqslant i'$, $j \leqslant j'$, $k \leqslant k'$ but $(i, j, k) \neq (i', j', k')$. If there exists a match for computing $L(i', j', k')$, one can obtain $L(i', j', k')$ from either $L(i' - 1, j' - 1, k') + 1$ or $L(i' - 1, j', k' - 1) + 1$ based on its matched situation, either $t_{i'} = a_{j'}$ or $t_{i'} = b_{k'}$, respectively. The value of $L(i', j', k')$ is at most $\max\{L(i' - 1, j' - 1, k'), L(i' - 1, j', k' - 1)\} + 1$. With the existence of this match, $L(i', j', k')$ reaches the possible maximal value, and hence $L(i', j', k')$ is optimal. If there is no match for computing $L(i', j', k')$, that is $t_{i'} \neq a_{j'}$ and $t_{i'} \neq b_{k'}$, we can try to remove one of symbols, $t_{i'}$, $a_{j'}$, or $b_{k'}$ for computing $L(i', j', k')$. Hence the optimal value of $L(i', j', k')$ must be the maximal value among $L(i' - 1, j', k')$, $L(i', j' - 1, k')$, and $L(i', j', k' - 1)$. Since there is no match, the value will not be increased and $L(i', j', k')$ is also optimal. Therefore, we can conclude that the dynamic programming formula is correct.

The time complexity is clearly $O(|T||A||B|)$ based on the dynamic programming formula. We can reduce the space complexity to $O(|A||B|)$ with the same concept of divide and conquer used in the LCS algorithm [6]. In order to reduce the space complexity, let

$$L_H\big([i, i'], [j, j'], [k, k']\big) = LCS\big(t_i t_{i+1} \cdots t_{i'-1} t_{i'}, E(a_j a_{j+1} \cdots a_{j'-1} a_{j'}, b_k b_{k+1} \cdots b_{k'-1} b_{k'})\big).$$

The MLCS can also be computed by the recurrence formula

$$L_H\big([i, i'], [j, j'], [k, k']\big) = \max_{j \leqslant p \leqslant j', k \leqslant q \leqslant k'} \big\{L_H\big(\big[i, \lfloor (i + i')/2 \rfloor\big], [j, p], [k, q]\big) + L_H\big(\big[\lfloor (i + i')/2 \rfloor + 1, i'\big], [p + 1, j'], [q + 1, k']\big)\big\}. \qquad \square$$

Since we assume that the lengths of sequences $T$, $A$ and $B$ are $O(n)$. it is clear that Algorithm MergedLCS can solve the MLCS problem in $O(n^3)$ time and $O(n^2)$ space.

## 3. The merged LCS problem with block constraint

In real applications, the MLCS problem may be accomplished with some constraints, such as the block constraint. The block constraint is based on the known sequence segments of genes or the specific DNA segments that we are interested in, such as the coding sequences (CDS) and mRNA segments in a GenBank file [3]. Fortunately, the required time and space for solving the MLCS problem will be reduced by adopting the block constraint.

The MLCS problem with block constraint, abbreviated as the blocked merged LCS (BMLCS) problem, is described as follows. Given a target sequence $T = t_1 t_2 \cdots t_{|T|}$, and a pair of merging block sequences $A = A_1 A_2 \cdots A_\alpha = a_1 a_2 \cdots a_{|A|}$ and $B = B_1 B_2 \cdots B_\beta = b_1 b_2 \ldots b_{|B|}$, where $\alpha$, $\beta$ are the numbers of the blocks of $A$ and $B$, respectively. A blocked merged sequence $E^b(A, B) = E_1 E_2 E_3 \cdots E_\varepsilon = e_1 e_2 e_3 \cdots e_{|E^b(A,B)|}$ is composed of the blocks in $A$ and $B$, where $E_i^b \in \{A_j \mid 1 \leqslant j \leqslant \alpha\} \cup \{B_k \mid 1 \leqslant k \leqslant \beta\}$ and $1 \leqslant i \leqslant \varepsilon$, $\varepsilon$ is the number of blocks in $E^b(A, B)$. That is, there exists one block subsequence $S$ of $E^b(A, B)$ such that $S$ is a block subsequence of $A$, and $E^b(A, B) - S$ is a block subsequence of $B$. Here, one block subsequence of a block sequence $A$ can be obtained by deleting zero or more blocks from $A$. The BMLCS problem is to find the block constrained LCS of $T$ and $E^b(A, B)$, denoted by

$$bLCS\big(T, E^b(A, B)\big) = bLCS\big(t_1 t_2 \cdots t_{|T|}, E^b(A_1 A_2 \cdots A_\alpha, B_1 B_2 \cdots B_\beta)\big).$$

By adding some block constraints into the example in Section 2, consider sequences $T =$ `atacgcgctt`, $A = A_1 A_2 =$ `cgat acc`, and $B = B_1 B_2 B_3 =$ `aat tc gc`. In detail, blocks $A_1 =$ `cgat`, $A_2 =$ `acc`, $B_1 =$ `aat`, $B_2 =$ `tc`, and $B_3 =$ `gc`. With the block constraint, let $E_4^b(A, B) = A_1 B_1 B_2 A_2 =$ `cgat aat tc acc`, $E_5^b(A, B) = B_1 A_1 A_2 B_2 =$ `aat cgat acc tc`, and $E_6^b(A, B) = B_1 B_2 A_1 B_3 A_2 =$ `aat tc cgat gc acc` are some blocked merged sequences of $A$ and $B$. We can compute that $bLCS(T, E_4^b(A, B)) =$ `ataccc`, $bLCS(T, E_5^b(A, B)) =$ `atcgcct`, and $bLCS(T, E_6^b(A, B)) =$ `ataccc`. In this block constrained case, $bLCS(T,$

$E^b(A, B)) = bLCS(T, E^b_5(A, B))$ with length 7. Note that the solution previously found for $LCS(T, E_2(A, B))$ does not survive the above block constraints.

Similar to the merged sequence, the blocked merged sequence of $A$ and $B$ may not be unique based on the definition. Here, we do not specify any block constraint on the target sequence $T$; so we do not expect that each block of $E^b(A, B)$ should align to any specific block of $T$. The reason is that if the target sequence is just sequenced, its annotation and block information may not be available yet. In the BMLCS problem, the absence of block information of any one merging sequence, $A$ or $B$, is allowed because we can regard each symbol as one block. If there exist block constraints on $T$, we can easily extend and modify Algorithm MergedLCS to solve the problem by regarding each block as a specific symbol, under the assumption that each block can only align to at most one block of another sequence. If several blocks in $T$ can be aligned to several blocks in $A$ and $B$, it turns out to be a harder problem, which is not discussed here. By combining Algorithm MergedLCS and the block constraint, we can use the dynamic programming formula in Fig. 1 to solve the BMLCS problem.

All of the three cases in Fig. 1 lead to an important characteristic, the end symbol of a block (EOB). The symbol index of $A$ or $B$ in $L(i, j, k)$ can be changed validly only when the symbol index is EOB. If the index is not at the end of a block, we must move this index until it reaches an EOB. There are $O(|T|)$ and $O(\alpha+\beta)$ positions that should be considered for $T$ and the EOB's of $E^b(A, B)$, respectively. When a specific position of $T$ and one EOB are fixed, $O(|A| + |B|)$ positions are needed to be compared in the worst case. We conclude that the time complexity is $O(|T|(|A| + |B|)(\alpha + \beta))$, abbreviated as $O(n^2m)$ in general with $m = \max\{\alpha, \beta\}$, which is less than that of Algorithm MergedLCS.

Let us consider a different viewpoint of the BMLCS problem. If we perform some preprocessing rather than the straightforward computation, we can design a more efficient algorithm. For example, to deal with sequences $T$, $A_1$ and $B_3$, we can spend $O(|T||A_1|)$ and $O(|T||B_3|)$ time to compute $LCS(T, A_1)$ and $LCS(T, B_3)$ respectively, and store the necessary information $S$-table [8,12,13] in advance. In brief, the $S$-table stores the leftmost indexes of various LCS lengths. If we want to compute $LCS(T, A_1B_3)$ or $LCS(T, B_3A_1)$, we can reuse the $S$-table. Let $T[i_1, i_2]$ represent one substring $t_{i_1}t_{i_1+1}\cdots t_{i_2-1}t_{i_2}$ of $T$, and $T[i_1, i_2]$ is empty if $i_1 > i_2$. The $S$-table of sequences $T$ and $X$, denoted as $S^t(T, X)$, stores the LCS length of $X$ and each suffix $T[i, |T|]$ of $T$. The symbol $\oplus$ denotes the merging operation of a vector and an $S$-table. Lemma 2 describes the $S$-table and a basic dynamic programming operation which utilizes a precomputed $S$-table to speed up LCS computations. This technique will later be applied to speed up our algorithm.

**Lemma 2.** *(See [1,8,11–13,17].) Given a sequence $T$ and a composite sequence $C = C_1C_2$, $LCS(T[1, j], C_1)$, $1 \leqslant j \leqslant n$, and $S^t(T, C_2)$ can be obtained in $O(|T||C_1|)$ and $O(|T||C_2|)$ time, respectively. If $LCS(T[1, j], C_1)$ and $S^t(T, C_2)$ are given, $LCS(T[1, j], C_1C_2)$ can be obtained by merging $LCS(T[1, j], C_1) \oplus S^t(T, C_2)$ in $O(|T|)$ time.*

Let

$$L^b(i, j, k) = LCS\big(t_1t_2\cdots t_i, E^b(A_1A_2\cdots A_j, B_1\cdots B_k)\big),$$

where $0 \leqslant i \leqslant |T|$, $0 \leqslant j \leqslant \alpha$, $0 \leqslant k \leqslant \beta$. The vector $V^b(j, k) = \langle L^b(1, j, k), L^b(2, j, k), \ldots, L^b(|T|, j, k)\rangle$ stores the LCS of $L^b(i, j, k)$ for each prefix $T[1, i]$ of $T$ and $E^b(A_1A_2\cdots A_j, B_1B_2\cdots B_k)$. For a specific position $x$,

$$L^b(x, j, k) = \max_{0\leqslant i\leqslant x}\big\{L^b(i, j-1, k) + LCS\big(T[i+1, x], A_j\big), L^b(i, j, k-1) + LCS\big(T[i+1, x], B_k\big)\big\}.$$

$$L(i,j,k)=\max\begin{cases}\max\begin{cases}L(i-1,j-1,k)+1 & \text{if } t_i=a_j,\\ L(i,j-1,k) & \text{if } t_i\neq a_j,\\ L(i-1,j,k-1)+1 & \text{if } t_i=b_k,\\ L(i,j,k-1) & \text{if } t_i\neq b_k,\\ L(i-1,j,k) & \text{if } t_i\neq a_j \text{ and } t_i\neq b_k,\end{cases} & \text{if } a_j=\text{EOB and } b_k=\text{EOB},\\ \max\begin{cases}L(i-1,j-1,k)+1 & \text{if } t_i=a_j,\\ L(i,j-1,k) & \text{if } t_i\neq a_j,\\ L(i-1,j,k) & \text{if } t_i\neq a_j,\end{cases} & \text{if } a_j\neq\text{EOB and } b_k=\text{EOB},\\ \max\begin{cases}L(i-1,j,k-1)+1 & \text{if } t_i=b_k,\\ L(i,j,k-1) & \text{if } t_i\neq b_k,\\ L(i-1,j,k) & \text{if } t_i\neq b_k,\end{cases} & \text{if } a_j=\text{EOB and } b_k\neq\text{EOB}.\end{cases} \tag{2}$$

Fig. 1. The dynamic programming formula for solving the BMLCS problem, where EOB means the end of a block.

Algorithm BMergedLCS+ is our improved version for solving the BMLCS problem by using the $S$-table.

**Algorithm BMergedLCS+**

```
Input:   A target sequence $T$, and two merging block sequences $A$ and $B$.
Output:  $bLCS(T, E^b(A, B)) = L^b(|T|, \alpha, \beta)$.
Step 1.  Compute and store $S^t(T, A_j)$ and $S^t(T, B_k)$ from $LCS(T, A_j)$ and $LCS(T, B_k)$, respectively, where $1 \leqslant j \leqslant \alpha$, $1 \leqslant k \leqslant \beta$.
Step 2.  Initialize $L^b(i, 0, 0) = 0$, for $1 \leqslant i \leqslant |T|$.
Step 3.  Compute $V^b(j,k) = \max\{V^b(j-1,k) \oplus S^t(T, A_j), V^b(j,k-1) \oplus S^t(T, B_k)\}$, for $1 \leqslant j \leqslant \alpha$, $1 \leqslant k \leqslant \beta$.
Step 4.  Return $bLCS(T, E^b(A, B)) = L^b(|T|, \alpha, \beta)$.
```

**Theorem 3.** *Algorithm BMergedLCS+ solves the BMLCS problem in $O(n^2 + nm^2)$ time and space.*

**Proof.** The correctness is clear because the recurrence formula in Step 3 checks all combinations of $E^b(A, B)$, and $bLCS(T, E^b(A, B))$ is obtained by merging small blocks progressively based on Lemma 2. In Step 1, it requires $O(|T|(|A| + |B|))$ time and space to compute and store the $S$-table. In Step 2, the initialization for $L^b(i, 0, 0)$ requires $O(|T|)$ time. In Step 3, the length of vector $V^b(j,k)$ is $|T|$. Based on Lemma 2, we can obtain the merged vectors $V^b(j-1,k) \oplus S(T, A_j)$ and $V^b(j,k-1) \oplus S(T, B_k)$ in $O(|T|)$ time. The vector $V^b(j,k)$ can be computed in $O(|T|)$ time by simply comparing $V^b(j-1,k) \oplus S(T, A_j)$ and $V^b(j,k-1) \oplus S(T, B_k)$ for each position in $T$. The merging operation $\oplus$ is invoked for each $j$ and $k$, where $1 \leqslant j \leqslant \alpha \leqslant m$ and $1 \leqslant k \leqslant \beta \leqslant m$. So, vector $V^b(j,k)$ can be computed in $O(n\alpha\beta)$ time, which is denoted as $O(nm^2)$ in general. Here, we can store all vectors $V^b(j,k)$ for tracing the LCS path [8,17]; this also requires $O(nm^2)$ storage. Thus, both time and space complexities of this algorithm are $O(n^2 + nm^2)$. □

## 4. Conclusion

In this paper, we introduce the merged LCS problem and its variant with the block constraint for finding interleaving relationship between sequences. The merged LCS problem can be used as one measurement of doubly conserved synteny block and thus it is a real problem in the molecular biology research. We first propose an algorithm for the merged LCS problem, whose time and space complexities are $O(n^3)$ and $O(n^2)$, respectively, where $n$ is the sequence length. For reducing time and space requirements of the algorithm, we consider the block information of real data and define the blocked merged LCS problem. For the blocked merged LCS problem, we propose two algorithms with time complexities $O(n^2 m)$ and $O(n^2 + nm^2)$, where $m$ is the number of blocks and $m \leqslant n$.

There are still some problems worth further study. Our algorithms here focus on the LCS problem. For protein sequences or considering the effect of gaps, one may extend the merged LCS problem to the merged alignment problem. Furthermore, one may extend the problem to the multiple merged LCS or multiple merged alignment problems for merging several sequences to be an algorithm for detecting 3R or 4R blocks of whole genome duplication.

## References

[1] A. Aggarwal, M.M. Klawe, S. Moran, P. Shor, R. Wilber, Geometric applications of a matrix-searching algorithm, Algorithmica 2 (1) (1987) 195–208.

[2] B.S. Baker, R. Giancarlo, Sparse dynamic programming for longest common subsequence from fragments, Journal of Algorithms 42 (2) (2002) 231–254.

[3] D.A. Benson, I. Karsch-Mizrachi, D.J. Lipman, J. Ostell, D.L. Wheeler, GenBank, Nucleic Acids Research 34 (2006) D16–D20.

[4] F.Y.L. Chin, A.D. Santis, A.L. Ferrara, N.L. Ho, S.K. Kim, A simple algorithm for the constrained sequence problems, Information Processing Letters 90 (4) (2004) 175–179.

[5] D. Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge Press, New York, 1997.

[6] D.S. Hirschberg, A linear space algorithm for computing maximal common subsequence, Communications of the ACM 18 (6) (1975) 341–343.

[7] K. Hokamp, A. McLysaght, K.H. Wolfe, The 2R hypothesis and the human genome sequence, Journal of Structural and Functional Genomics 3 (1–4) (2003) 95–110.

[8] K.-S. Huang, C.-B. Yang, K.-T. Tseng, Y.-H. Peng, H.-Y. Ann, Dynamic programming algorithms for the mosaic longest common subsequence problem, Information Processing Letters 102 (2–3) (2007) 99–103.

[9] O. Jaillon, et al., Genome duplication in the teleost fish *tetraodon nigroviridis* reveals the early vertebrate proto-karyotype, Nature 431 (7011) (2004) 946–957.

[10] M. Kellis, B.W. Birren, E.S. Lander, Proof and evolutionary analysis of ancient genome duplication in the yeast *saccharomyces cerevisiae*, Nature 428 (6983) (2004) 617–624.

[11] G.M. Landau, E. Myers, M. Ziv-Ukelson, Two algorithms for lcs consecutive suffix alignment, in: Combinatorial Pattern Matching, in: Lecture Notes in Computer Science, vol. 3109, Springer, Berlin, Heidelberg, 2004, pp. 173–193.

[12] G.M. Landau, B. Schieber, M. Ziv-Ukelson, Sparse LCS common substring alignment, Information Processing Letters 88 (6) (2003) 259–270.

[13] G.M. Landau, M. Ziv-Ukelson, On the common substring alignment problem, Journal of Algorithms 41 (2) (2001) 338–354.

[14] D. Maier, The complexity of some problems on subsequences and supersequences, Journal of the ACM 25 (2) (1978) 322–336.

[15] S. Ohno, Evolution by Gene Duplication, Springer, Heidelberg, 1970.

[16] C.-L. Peng, An approach for solving the constrained longest common subsequence problem, Master Thesis, Department of Computer Science and Engineering, National Sun Yat-sen University, Taiwan, July 2003.

[17] J.P. Schmidt, All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings, SIAM Journal on Computing 27 (4) (1998) 972–992.

[18] T.F. Smith, M.S. Waterman, Identification of common molecular subsequences, Journal of Molecular Biology 147 (1) (1981) 195–197.

[19] C.B. Yang, R.C.T. Lee, Systolic algorithms for the longest common subsequence problem, Journal of the Chinese Institute of Engineers 10 (6) (1987) 691–699.

[20] I.-H. Yang, C.-P. Huang, K.-M. Chao, A fast algorithm for computing a longest common increasing subsequence, Information Processing Letters 93 (5) (2005) 249–253.