Algorithmica (2015) 72:430–449
DOI 10.1007/s00453-013-9859-z

Resequencing a Set of Strings Based on a Target String

Chih-En Kuo · Yue-Li Wang · Jia-Jie Liu · Ming-Tat Ko

Received: 8 September 2012 / Accepted: 16 December 2013 / Published online: 28 December 2013

Abstract Given a set $S = \{S_1, S_2, \ldots, S_l\}$ of l strings, a text T, and a natural number k, find a string M, which is a concatenation of k strings (not necessarily distinct, i.e., a string in S may occur more than once in M) from S, whose longest common subsequence with T is largest, where a string in S may occur more than once in M. Such a string is called a k-inlay. The resequencing longest common subsequence problem (resequencing LCS problem for short) is to find a k-inlay for each query with parameter k after T and S are given. In this paper, we propose an algorithm for solving this problem which takes $O(nml)$ preprocessing time and $O(\vartheta_k k)$ query time for each query with parameter k, where n is the length of T, m is the maximal length of strings in S, and ϑ_k is the length of the longest common subsequence between a k-inlay and T.

Keywords Dynamic programming · Longest common subsequences · Resequencing · Inverted indexing · Totally monotone matrices

This work was supported in part by the National Science Council of the Republic of China under contracts NSC 100-2221-E-011-067-MY3 and NSC 101-2221-E-011-038-MY3.

C.-E. Kuo · Y.-L. Wang (✉)
Department of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan
e-mail: ylwang@cs.ntust.edu.tw

J.-J. Liu
Department of Information Management, Shih Hsin University, Taipei, Taiwan

M.-T. Ko
Institute of Information Science, Academia Sinica, Taipei, Taiwan

1 Introduction

The longest common subsequence problem, abbreviated LCS problem, is a well-known problem in computer science which has been extensively studied in many apparently unrelated fields, such as data comparison, speech recognition, genetic engineering, editing, error correction, mathematics, syntactic pattern recognition, and especially bioinformatics, etc., [2, 7, 8, 11, 13, 22]. Many algorithms use the dynamic programming technique for solving the LCS problem [14, 21, 23]. A lot of variations of LCS problems were researched such as LCS applied to matrix substructures [4], weighted sequences [5], tree edit distances [12, 24], run length encoded strings [20], etc. In this paper, we investigate the *resequencing LCS problem* which is defined as follows:

Definition 1.1 Given a set $S = \{S_1, S_2, \ldots, S_l\}$ of l strings, a text T, and a natural number k, a string M is called a k-mosaic string if it is a concatenation of k strings (not necessarily distinct, i.e., a string in S may occur more than once in M) from S. A k-mosaic string is called a k-inlay if the length of its LCS with T is maximum, denoted by ϑ_k. The resequencing LCS problem is to find a k-inlay for each query with parameter k.

For example, let $S = \{agc, act, aatg, ttcg\}$ and $T = agactagtc$ in which $S_1 = agc$, $S_2 = act$, $S_3 = aatg$, and $S_4 = ttcg$. Both $S_1S_1 = agcagc$ and $S_1S_4 = agcttcg$ are 2-inlays, and the length of their LCSs with T is 6. For $k = 4$, $S_1S_2S_1S_4 = agcactagcttcg$ is a 4-inlay, and the length of its LCS with T is $\vartheta_4 = 9$.

In [14], Hirschberg discussed a special case of the resequencing LCS problem with $k = 2$. In [17], Komatsoulis and Waterman solved a special case of the resequencing LCS problem which is described as follows. Given a database S of DNA sequences and a query sequence T, finding two sequences C_1 and C_2 from S such that C_1C_2 has the maximum chimeric alignment score with respect to T. Komatsoulis and Waterman also applied their algorithm to detect chimeric 16S rRNA artifacts in biology [18]. Their algorithm takes $O(nml)$ time where n is the length of T, m is the maximal length of strings in S, and l is the number of strings in S. In [15], Huang et al. proposed two algorithms for solving the resequencing LCS problem. One of their algorithms takes $O(n^2ml)$ preprocessing time and $O(n^3 \log k)$ query time for finding out k-inlays with respect to all substrings of T while the other algorithm takes $O(nml)$ preprocessing time and $O(nkl)$ query time when considering all substrings of T in the form $T(1, j)$, $1 \le j \le n$, i.e., the prefix of T with j characters. In this paper, we propose two algorithms for solving the resequencing LCS problem. One of them is for all substrings of T and the other is for all substrings in the form $T(1, j)$, $1 \le j \le n$. The former takes $O(nml)$ preprocessing time and $O(n\vartheta_k \log k)$ query time for finding out k-inlays with respect to all substrings of T. The latter which is modified from the former can also find out k-inlays with respect to $T(1, j)$, for $1 \le j \le n$, in $O(nml)$ preprocessing time and $O(\vartheta_k k)$ query time. Note that the query time of our second algorithm is unrelated to n.

The main contribution of this paper is described briefly as follows. To achieve the improvements, two inverted index representations are successively introduced to

reformulate the problem. The first inverted index representation is used frequently in the LCS problem so that the time-complexity is related to the length of an LCS. This also improves the time-complexity of the algorithm proposed in [15] to be in $O(n\vartheta_k^2 \log k)$ time for each query. Based on the first inverted index representation, the second inverted index representation is introduced and we show that the second representation has the totally monotone property. Thus the time-complexity of the above algorithm can be done in $O(n\vartheta_k \log k)$ time. Here we want to point out that our second inverted index representation can be used to solve the problems efficiently when the inverted index representation is used repeatedly on its previous inverted index table.

The remaining part of this paper is organized as follows. In Sect. 2, we briefly introduce a modified version of Algorithm `Formosa1` which was proposed in [15]. In Sect. 3, we introduce an inverted index representation of the LCS table used in Algorithm `Formosa1`. Furthermore, we also propose a recurrence formula for obtaining all inverted indexing tables. In Sect. 4, for computing the inverted indexing tables efficiently, we transform an inverted indexing table to another table which has the totally monotone property. In Sect. 5, we modify our algorithm so that k-inlays with respect to $T(1, j)$, $1 \leq j \leq n$, can be found in $O(\vartheta_k k)$ query time. Finally, the last section contains our concluding remarks.

2 Preliminaries

Let $X = x_1x_2x_3 \cdots x_r$ and $T=t_1t_2 \cdots t_n$ be two strings over a finite alphabet set Σ. Let $T[i, j] = t_i t_{i+1} \cdots t_j$ be a substring of T. If $j < i$, then $T[i, j] = \epsilon$, i.e., an empty string. Moreover, when $i = 1$, $T[i, j]$ is abbreviated as T_j. For brevity, T_n is simply written as T. A *subsequence* of $T[i, j]$ is obtained by deleting zero or some (not necessarily consecutive) characters in this string. A *common subsequence* of X and $T[i, j]$ is a subsequence occurring in both X and $T[i, j]$. A *longest common subsequence* (LCS for short) of X and $T[i, j]$ is a common subsequence of X and $T[i, j]$ with the maximum length.

Definition 2.1 The length of the LCS of X and $T[i, j]$ is denoted by $L_X(i, j)$ for $1 \leq i \leq j \leq n$. Note that $L_X(i, j) = 0$ if $j < i$. When X is a k-inlay with $T[i, j]$, we use $L_k(i, j)$ to replace $L_X(i, j)$ so that we can easily describe recurrence formulas for $L_k(i, j)$.

Definition 2.2 Let k_α stand for the number $b_\alpha b_{\alpha-1} \cdots b_0$ for $0 \leq \alpha \leq w$ if the binary representation of k is $b_w b_{w-1} \cdots b_0$ and let $L_{k_\alpha}(i, j)$ represent the length of the LCS of a k_α-inlay and $T[i, j]$. Note that the LCS length of a k-inlay with T, i.e., ϑ_k, is equal to $L_k(1, n)$.

In [14], Hirschberg proposed a formula described in Lemma 2.3 which can be used to solve the resequencing LCS problem. Lemma 2.4 can be proved directly by Lemma 2.3.

Lemma 2.3 ([14]) *Given a string T_n and a string $M = S_1S_2$, there exists a position r, $0 \le r \le n$, in T_n such that $L_M(1,n) = L_{S_1}(1,r) + L_{S_2}(r+1,n)$.*

Lemma 2.4 *Given a substring $T[i,j]$ of T_n and a set $S = \{S_1, S_2, \ldots, S_l\}$ of l strings,*

$$L_{2^\alpha}(i,j) = \begin{cases} \max_{1\le p\le l}\{L_{S_p}(i,j)\} & \text{if } \alpha = 0 \\ \max_{i-1\le r\le j}\{L_{2^{\alpha-1}}(i,r) + L_{2^{\alpha-1}}(r+1,j)\} & \text{if } \alpha \ge 1 \end{cases}$$

where $1 \le i \le j \le n$.

Based on Lemma 2.4, Algorithm `Formosa1` proposed in [15] solves the resequencing LCS problem as follows.

Algorithm `Formosa1`

Input: A string T_n, a set $S = \{S_1, S_2 \cdots, S_l\}$ of l strings, and a positive integer k whose binary representation is $b_w b_{w-1} \cdots b_0$.
Output: $L_k(1,n)$.
Step 1. /* Merge all LCS tables of S_i, for $1 \le i \le l$, and T to the LCS table L_{2^0}. */
Let $L_1(i,j) = \max_{1\le p\le l}\{L_{S_p}(i,j)\}$, where $1 \le i \le j \le n$.
Step 2. /* Compute $L_{2^\alpha}(i,j)$, for $1 \le \alpha \le w$, by using Lemma 2.4. */
Compute $L_{2^\alpha}(i,j) = \max_{i-1\le r\le j}\{L_{2^{\alpha-1}}(i,r) + L_{2^{\alpha-1}}(r+1,j)\}$, for $\alpha = 1,2,\ldots,w$, where $1 \le i \le j \le n$.
Step 3. /* Compute $L_k(i,j)$ by using tables L_{2^α}, for $1 \le \alpha \le w$. */
If $k \ne 2^w$, then compute $L_{k_\alpha}(i,j) = \max_{i-1\le r\le j}\{b_\alpha \cdot L_{2^\alpha}(i,r) + L_{k_{\alpha-1}}(r+1,j)\}$, for $\alpha = 1,2,\ldots,w$, where $1 \le i \le j \le n$.
Step 4. Return $L_k(1,n)$.

We use the example mentioned in Sect. 1 to illustrate Algorithm `Formosa1` in which $S = \{agc, act, aatg, ttcg\}$ and $T = agactagtc$. In Step 1, LCS tables L_{S_1}, L_{S_2}, L_{S_3}, and L_{S_4} are computed. By definition, $L_{2^\alpha}(i,j)$ is the largest LCS value between $T[i,j]$ and all possible 2^α-mosaic strings for some $0 \le \alpha \le \lfloor \log k \rfloor$. Therefore, table $L_{2^0}(i,j)$ as shown in Fig. 1(a) can be obtained from tables L_{S_i}, for $1 \le i \le 4$. Note that table $L_{2^0}(i,j)$ is also computed in Step 1. Then, by Lemma 2.4, $L_{2^{\alpha+1}}(i,j)$ can be computed by using the values in table L_{2^α} for $\alpha = 0,1,\ldots,\lfloor \log k \rfloor - 1$ in Step 2. Figures 1(a), 1(b), and 1(c) are tables L_{2^0}, L_{2^1}, and L_{2^2}, respectively. Note that Figs. 1(b) and 1(c) are obtained from Figs. 1(a) and 1(b), respectively, by using the formula described in Lemma 2.4. For example,

$$\begin{aligned} L_{2^1}(1,4) &= \max\big\{L_{2^0}(1,1) + L_{2^0}(2,4), L_{2^0}(1,2) + L_{2^0}(3,4), L_{2^0}(1,3) \\ &\quad + L_{2^0}(4,4), L_{2^0}(1,4)\big\} \\ &= \max\{1+2, 2+2, 2+1, 3\} \\ &= 4. \end{aligned}$$

	j								
	1	2	3	4	5	6	7	8	9
$L_{20}(1,j)$	1	2	2	3	3	3	4	4	4
$L_{20}(2,j)$	0	1	1	2	3	3	3	3	3
$L_{20}(3,j)$	0	0	1	2	3	3	3	3	3
$L_{20}(4,j)$	0	0	0	1	2	2	2	2	3
$L_{20}(5,j)$	0	0	0	0	1	1	2	2	3
$L_{20}(6,j)$	0	0	0	0	0	1	2	2	3
$L_{20}(7,j)$	0	0	0	0	0	0	1	1	2
$L_{20}(8,j)$	0	0	0	0	0	0	0	1	2
$L_{20}(9,j)$	0	0	0	0	0	0	0	0	1

(a) $L_{20}(i,j)$

	j								
	1	2	3	4	5	6	7	8	9
$L_{21}(1,j)$	1	2	3	4	5	5	5	5	6
$L_{21}(2,j)$	0	1	2	3	4	4	5	5	6
$L_{21}(3,j)$	0	0	1	2	3	4	5	5	6
$L_{21}(4,j)$	0	0	0	1	2	3	4	4	5
$L_{21}(5,j)$	0	0	0	0	1	2	3	3	4
$L_{21}(6,j)$	0	0	0	0	0	1	2	3	4
$L_{21}(7,j)$	0	0	0	0	0	0	1	2	3
$L_{21}(8,j)$	0	0	0	0	0	0	0	1	2
$L_{21}(9,j)$	0	0	0	0	0	0	0	0	1

(b) $L_{21}(i,j)$

	j								
	1	2	3	4	5	6	7	8	9
$L_{22}(1,j)$	1	2	3	4	5	6	7	8	9
$L_{22}(2,j)$	0	1	2	3	4	5	6	7	8
$L_{22}(3,j)$	0	0	1	2	3	4	5	6	7
$L_{22}(4,j)$	0	0	0	1	2	3	4	5	6
$L_{22}(5,j)$	0	0	0	0	1	2	3	4	5
$L_{22}(6,j)$	0	0	0	0	0	1	2	3	4
$L_{22}(7,j)$	0	0	0	0	0	0	1	2	3
$L_{22}(8,j)$	0	0	0	0	0	0	0	1	2
$L_{22}(9,j)$	0	0	0	0	0	0	0	0	1

(c) $L_{22}(i,j)$

	ℓ			
	1	2	3	4
$L_{20}^{-1}(1,\ell)$	1	2	4	7
$L_{20}^{-1}(2,\ell)$	2	4	5	0
$L_{20}^{-1}(3,\ell)$	3	4	5	0
$L_{20}^{-1}(4,\ell)$	4	5	9	0
$L_{20}^{-1}(5,\ell)$	5	7	9	0
$L_{20}^{-1}(6,\ell)$	6	7	9	0
$L_{20}^{-1}(7,\ell)$	7	9	0	0
$L_{20}^{-1}(8,\ell)$	8	9	0	0
$L_{20}^{-1}(9,\ell)$	9	0	0	0

(d) $L_{20}^{-1}(i,\ell)$

Fig. 1 Tables L_{20}, L_{21}, L_{22}, and L_{20}^{-1} where $S = \{agc, act, aatg, ttcg\}$ and $T = agactagtc$

If $k = 4$, then $L_k(1,n) = L_{2^2}(1,9) = 9$. If $4 < k \leq 7$, then, by combining related tables of L_{2^α} for $\alpha = 0, 1, \ldots, \lfloor \log k \rfloor - 1$, $L_k(i,j)$, for $1 \leq i \leq j \leq n$, can also be obtained in Step 3.

Intuitively, Step 1 of Algorithm `Formosa1`, which can be regarded as the preprocessing step, takes $O(n^2ml)$ time. Steps 2–4 can be regarded as the query procedure which takes k as an input parameter. The most time-consuming steps of Algorithm `Formosa1` are Steps 2 and 3 which take $O(n^3 \log k)$ time. Therefore, the query time of Algorithm `Formosa1` is $O(n^3 \log k)$ [15]. In the next two sections, we propose an efficient algorithm for finding $L_k(i,j)$ with $O(nml)$ preprocessing time and $O(n\vartheta_k \log k)$ query time. With a minor modification, our algorithm can find out $L_k(1,n)$ with $O(nml)$ preprocessing time and $O(\vartheta_k k)$ query time.

3 An Inverted Index Representation

First, let us observe Step 1 of Algorithm `Formosa1`. Intuitively, it takes $O(n^2m)$ time to compute the lengths of LCSs between each string in S and all substrings of T. Thus it takes $O(n^2ml)$ time for all strings in S. However, in [3], Alves et al. presented an algorithm which can compute the lengths of LCSs between one string and all substrings of T in $O(nm)$ time. Hence, by using their algorithm, Step 1 can be done in $O(nml)$ time. To compute $L_k(i,j)$ efficiently, we use an inverted index

representation [16] to represent $L_k(i, j)$ in the rest of this section. The resulting table is called an *inverted indexing table*. That is, a nonzero value ℓ in entry (i, j), i.e., row i and column j, of $L_k(i, j)$ yields a value j in entry (i, ℓ) of the inverted indexing table. Note that if more than one entry has the same value in a row of $L_k(i, j)$, then only the smallest index will be stored as the entry value in the inverted indexing table.

Definition 3.1 Let L_k^{-1} denote the inverted indexing table of L_k. Assume that $L_k(i, 0) = 0$ for $1 \le i \le n$. If $L_k(i, j) = \ell$ and $L_k(i, j-1) \ne \ell$, for $1 \le i \le j \le n$, then $L_k^{-1}(i, \ell) = j$. If no such j exists, then $L_k^{-1}(i, \ell) = 0$.

Clearly, the values of $L_k(i, j)$ can be obtained easily through the values of $L_k^{-1}(i, \ell)$. In Sect. 4, we shall propose an efficient way for obtaining the inverted indexing tables. By the definition of L_k^{-1}, we can obtain Propositions 3.2 and 3.3 directly.

Proposition 3.2 *$L_k(i, L_k^{-1}(i, \ell)) = \ell$ when $L_k^{-1}(i, \ell) \ne 0$ for $1 \le i \le n$.*

Proposition 3.3 *If $L_k^{-1}(i, \ell - 1) = j_1$ and $L_k^{-1}(i, \ell) = j_2$, then $L_k(i, j) = \ell - 1$ for $1 \le i \le n$ and $j_1 \le j < j_2$.*

Definition 3.4 Let $\delta_k(i)$ stand for the maximum value in row i of table L_k for a k-inlay and substring $T(i, n)$.

By Definition 3.4, $\delta_0(i) = 0$, for $i = 1, 2, \ldots, n$. It is obvious that if $\delta_k(i) \ne 0$, then $L_k^{-1}(i, \ell) = 0$, for all $\ell > \delta_k(i)$. Figures 1(d), 2(a), and 2(b) show all values in $L_{2^0}^{-1}$, $L_{2^1}^{-1}$, and $L_{2^2}^{-1}$, respectively, for the tables in Fig. 1. Recall that the values in $L_{2^0}^{-1}$ can be obtained from the LCS table computed by the algorithm in [3]. For example, $L_{2^0}^{-1}(1, 1) = 1$ since $L_{2^0}(1, 1) = 1$ and $L_{2^0}(1, 0) = 0$. $L_{2^0}^{-1}(1, 2) = 2$ since $L_{2^0}(1, 2) = 2$ and $L_{2^0}(1, 1) = 1$. $L_{2^0}^{-1}(1, 3) = 4$ since $L_{2^0}(1, 4) = 3$ and $L_{2^0}(1, 3) = 2$ and so on. In addition, $\delta_{2^0}(1) = 4$, $\delta_{2^0}(2) = \delta_{2^0}(3) = \cdots = \delta_{2^0}(6) = 3$, $\delta_{2^0}(7) = \delta_{2^0}(8) = 2$, and $\delta_{2^0}(9) = 1$. Therefore, $L_{2^0}^{-1}(2, 4) = L_{2^0}^{-1}(3, 4) = \cdots = L_{2^0}^{-1}(6, 4) = 0$, $L_{2^0}^{-1}(7, 3) = L_{2^0}^{-1}(7, 4) = L_{2^0}^{-1}(8, 3) = L_{2^0}^{-1}(8, 4) = 0$, and $L_{2^0}^{-1}(9, 2) = L_{2^0}^{-1}(9, 3) = L_{2^0}^{-1}(9, 4) = 0$.

In the following, Proposition 3.5 describes the difference between two adjacent values in table L_{2^α}. Proposition 3.6 states the fact that all values in tables $L_{k_0}^{-1}$ and L_{k_0} are zero when $b_0 = 0$. Proposition 3.7 describes the fact that the values of $L_{k_\alpha}^{-1}(i, j)$ can be obtained from $L_{k_{\alpha-1}}^{-1}(i, j)$ directly when $b_\alpha = 0$. Lemmas 3.8 and 3.9 present formulas for computing $L_{k_\alpha}(i, j)$ and $L_{2^\alpha}(i, j)$, respectively, by using the values in tables $L_{2^\alpha}^{-1}$ and $L_{2^{\alpha-1}}^{-1}$, respectively. Lemmas 3.10 and 3.11 provide formulas for computing $\delta_{k_\alpha}(i)$ and $\delta_{2^\alpha}(i)$, respectively, by using the values in tables $L_{2^\alpha}^{-1}$ and $L_{2^{\alpha-1}}^{-1}$, respectively. Note that $\delta_{k_\alpha}(1)$ (respectively, $\delta_{2^\alpha}(1)$) is equal to the number of columns of table $L_{k_\alpha}^{-1}$ (respectively, $L_{2^\alpha}^{-1}$). Thus we can determine the size of $L_{k_\alpha}^{-1}$ (respectively, $L_{2^\alpha}^{-1}$) before constructing it.

Proposition 3.5 *$L_{2^\alpha}(i, j+1) - L_{2^\alpha}(i, j) \in \{0, 1\}$, for $1 \le i \le j \le n-1$, and $L_{2^\alpha}(i, j) - L_{2^\alpha}(i-1, j) \in \{0, 1\}$, for $2 \le i \le j \le n$.*

Proposition 3.6 *If $b_0 = 0$, then $L_{k_0}^{-1}(i, j) = 0$ and $L_{k_0}(i, j) = 0$ for $1 \le i \le j \le n$.*

Proof By Definition 2.2, if $b_0 = 0$, then $k_0 = 0$, and L_{k_0} is exactly L_0. By Definition 1.1, there is no 0-mosaic string. Thus all entries in L_{k_0} and $L_{k_0}^{-1}$ are zero. □

Proposition 3.7 *If $b_\alpha = 0$, then $L_{k_\alpha}^{-1}(i, j) = L_{k_{\alpha-1}}^{-1}(i, j)$ for $1 \le \alpha \le \lfloor \log k \rfloor$, where $1 \le i \le j \le n$.*

In Lemma 2.4, the formula $L_{2^{\alpha-1}}(i, r) + L_{2^{\alpha-1}}(r+1, j)$ is used to compute $L_{2^\alpha}(i, j)$. Assume that $r = L_{2^{\alpha-1}}^{-1}(i, \ell)$. Then the above formula becomes $L_{2^{\alpha-1}}(i, L_{2^{\alpha-1}}^{-1}(i, \ell)) + L_{2^{\alpha-1}}(L_{2^{\alpha-1}}^{-1}(i, \ell) + 1, j)$. Since the term $L_{2^{\alpha-1}}^{-1}(i, \ell) + 1$ is used frequently in the rest of this paper, for brevity, we use $\beta_\alpha(i, \ell)$ to stand for it, i.e., $\beta_\alpha(i, \ell) = L_{2^{\alpha-1}}^{-1}(i, \ell) + 1$.

Lemma 3.8

$$L_{k_\alpha}(i, j) = \begin{cases} 0 & \text{if } L_{2^\alpha}(i, j) = 0 \\ L_{k_{\alpha-1}}(i, j) & \text{if } b_\alpha = 0 \\ \max_{0 \le \ell \le L_{2^\alpha}(i,j)} \{\ell + L_{k_{\alpha-1}}(\beta_{\alpha+1}(i, \ell), j)\} & \text{if } b_\alpha = 1, \end{cases}$$

where $1 \le i \le j \le n$.

Proof If $L_{2^\alpha}(i, j) = 0$, then clearly $L_{2^x}(i, j) = 0$, for $x = 0, 1, \ldots, \alpha - 1$. When $b_\alpha = 0$, $k_\alpha = k_{\alpha-1}$. Thus, in this case, it is obvious that $L_{k_\alpha}(i, j) = L_{k_{\alpha-1}}(i, j)$. For the case where $b_\alpha = 1$, by Lemma 2.4 and Proposition 3.5,

$$\begin{aligned} L_{k_\alpha}(i, j) &= \max_{i-1 \le r \le j} \left\{ L_{2^\alpha}(i, r) + L_{k_{\alpha-1}}(r+1, j) \right\} \\ &= \max_{0 \le \ell \le L_{2^\alpha}(i,j)} \left\{ L_{2^\alpha}\left(i, L_{2^\alpha}^{-1}(i, \ell)\right) + L_{k_{\alpha-1}}\left(\beta_{\alpha+1}(i, \ell), j\right) \right\} \quad (1) \\ &= \max_{0 \le \ell \le L_{2^\alpha}(i,j)} \left\{ \ell + L_{k_{\alpha-1}}\left(\beta_{\alpha+1}(i, \ell), j\right) \right\}. \quad (2) \end{aligned}$$

We explain the reason why setting $0 \le \ell \le L_{2^\alpha}(i, j)$ in Eq. (1) as follows. Note that the value of $L_{2^\alpha}(i, i-1)$ is 0. The values from $L_{2^\alpha}(i, i-1)$ to $L_{2^\alpha}(i, j)$ are consecutively in the range from 0 to the value of $L_{2^\alpha}(i, j)$. Therefore, the range $[L_{2^\alpha}(i, i-1), L_{2^\alpha}(i, j)]$ is exactly $[0, L_{2^\alpha}(i, j)]$. Note that $L_{2^\alpha}(i, j) = L_{2^\alpha}^{-1}(i, L_{2^\alpha}(i, j))$. This implies that $0 \le \ell \le L_{2^\alpha}(i, j)$. By Proposition 3.2, Eq. (1) can be simplified as Eq. (2). □

Lemma 3.9 *$L_{2^\alpha}(i, j) = \max_{1 \le \ell \le L_{2^{\alpha-1}}(i,j)} \{\ell + L_{2^{\alpha-1}}(\beta_\alpha(i, \ell), j)\}$, where $1 \le i \le j \le n$.*

Proof Using a similar argument as in Lemma 3.8, we can have the following derivation:

$$\begin{aligned} L_{2^\alpha}(i,j) &= \max_{i-1\le r\le j}\left\{L_{2^{\alpha-1}}(i,r)+L_{2^{\alpha-1}}(r+1,j)\right\} \\ &= \max_{1\le \ell\le L_{2^{\alpha-1}}(i,j)}\left\{L_{2^{\alpha-1}}\left(i,L^{-1}_{2^{\alpha-1}}(i,\ell)\right)+L_{2^{\alpha-1}}\left(\beta_{\alpha,}(i,\ell),j\right)\right\} \\ &= \max_{1\le \ell\le L_{2^{\alpha-1}}(i,j)}\left\{\ell+L_{2^{\alpha-1}}\left(\beta_\alpha(i,\ell),j\right)\right\}. \end{aligned}$$

This completes the proof. □

To illustrate Lemma 3.9, we use the table in Fig. 1(d) as an example to show the computation of $L_{2^1}(2,9)$.

$$\begin{aligned} L_{2^1}(2,9) &= \max_{1\le \ell\le L_{2^0}(2,9)}\left\{\ell+L_{2^0}\left(\beta_1(2,\ell),9\right)\right\} \\ &= \max_{1\le \ell\le 3}\left\{\ell+L_{2^0}\left(L^{-1}_{2^0}(2,\ell)+1,9\right)\right\} \\ &= \max\left\{1+L_{2^0}\left(L^{-1}_{2^0}(2,1)+1,9\right),2+L_{2^0}\left(L^{-1}_{2^0}(2,2)+1,9\right),3\right. \\ &\quad \left.+L_{2^0}\left(L^{-1}_{2^0}(2,3)+1,9\right)\right\} \\ &= \max\left\{1+L_{2^0}(3,9),2+L_{2^0}(5,9),3+L_{2^0}(6,9)\right\} \\ &= \max\{1+3,2+3,3+3\} \\ &= 6. \end{aligned}$$

Lemma 3.10

$$\delta_{k_\alpha}(i)=\begin{cases} L_{2^0}(i,n) & \text{if } \alpha=0 \\ \delta_{k_{\alpha-1}}(i) & \text{if } \alpha>0 \text{ and } b_\alpha=0 \\ \max_{1\le\ell\le\delta_{2^\alpha}(i)}\{\ell+\delta_{k_{\alpha-1}}(\beta_{\alpha+1}(i,\ell))\} & \text{if } \alpha>0 \text{ and } b_\alpha=1, \end{cases}$$

where $1\le i\le n$.

Proof By definition, if $\alpha=0$, then $\delta_{k_\alpha}(i)=L_{2^0}(i,n)$. For the case where $\alpha>0$ and $b_\alpha=0$, by Lemma 3.8, $L_{k_\alpha}(i,j)=L_{k_{\alpha-1}}(i,j)$. This implies $\delta_{k_\alpha}(i)=\delta_{k_{\alpha-1}}(i)$. It remains to prove the case where $\alpha>0$ and $b_\alpha=1$. We can have the following derivations.

$$\begin{aligned} \delta_{k_\alpha}(i) &= L_{k_\alpha}(i,n) && (3)\\ &= \max_{1\le\ell\le L_{2^\alpha}(i,n)}\left\{\ell+L_{k_{\alpha-1}}\left(\beta_{\alpha+1}(i,\ell),n\right)\right\} && (4)\\ &= \max_{1\le\ell\le\delta_{2^\alpha}(i)}\left\{\ell+\delta_{k_{\alpha-1}}\left(\beta_{\alpha+1}(i,\ell)\right)\right\}. && (5) \end{aligned}$$

Equation (3) is obtained by definition. By applying Lemma 3.8 on $L_{k_\alpha}(i,n)$, it yields Eq. (4). By definition, $L_{2^\alpha}(i,n) = \delta_{2^\alpha}(i)$ and $L_{k_{\alpha-1}}(L^{-1}_{2^\alpha}(i,\ell)+1,n) = \delta_{k_{\alpha-1}}(L^{-1}_{2^\alpha}(i,\ell)+1)$. This results in Eq. (5) and the lemma follows. □

Lemma 3.11 *$\delta_{2^\alpha}(i) = \max_{1\le\ell\le\delta_{2^{\alpha-1}}(i)}\{\ell+\delta_{2^{\alpha-1}}(\beta_\alpha(i,\ell))\}$ for $1\le\alpha\le\lfloor\log k\rfloor$ and $1\le i\le n$.*

Proof Using a similar argument as in Lemma 3.10 and applying Lemma 3.9 on $L_{2^\alpha}(i,n)$, we can have the following derivation:

$$\begin{aligned}\delta_{2^\alpha}(i) &= L_{2^\alpha}(i,n)\\ &= \max_{1\le\ell\le L_{2^{\alpha-1}}(i,n)}\left\{\ell + L_{2^{\alpha-1}}\left(\beta_\alpha(i,\ell),n\right)\right\}\\ &= \max_{1\le\ell\le\delta_{2^{\alpha-1}}(i)}\left\{\ell+\delta_{2^{\alpha-1}}\left(\beta_\alpha(i,\ell)\right)\right\}\end{aligned}$$

This completes the proof. □

We also use the previous example for computing $\delta_{2^1}(2)$ to illustrate Lemma 3.11.

$$\begin{aligned}\delta_{2^1}(2) &= \max\left\{1+\delta_{2^0}\left(\beta_1(2,1)\right),2+\delta_{2^0}\left(\beta_1(2,2)\right),3+\delta_{2^0}\left(\beta_1(2,3)\right)\right\}\\ &= \max\left\{1+\delta_{2^0}\left(L^{-1}_{2^0}(2,1)+1\right),2+\delta_{2^0}\left(L^{-1}_{2^0}(2,2)+1\right),3\right.\\ &\quad\left.+\delta_{2^0}\left(L^{-1}_{2^0}(2,3)+1\right)\right\}\\ &= \max\left\{1+\delta_{2^0}(2+1),2+\delta_{2^0}(4+1),3+\delta_{2^0}(5+1)\right\}\\ &= \max\{1+3,2+3,3+3\}\\ &= 6.\end{aligned}$$

After finding all entries of table $L^{-1}_{2^0}$, all entries of table $L^{-1}_{2^\alpha}$ can be computed by using only the entries of $L^{-1}_{2^{\alpha-1}}$, for $1\le\alpha\le\lfloor\log k\rfloor$. Lemma 3.12 describes how to do in this way.

Lemma 3.12

$$L^{-1}_{2^\alpha}(i,\ell) = \begin{cases} L^{-1}_{2^{\alpha-1}}(i,1) & \text{if } \ell=1\\ \min\limits_{\substack{1\le p\le\delta_{2^{\alpha-1}}(i)\\ 1\le\ell-p\le\delta_{2^{\alpha-1}}(\beta_\alpha(i,p))}}\{L^{-1}_{2^{\alpha-1}}(\beta_\alpha(i,p),\ell-p)\} & \text{if } \ell\ge 2\\ 0 & \text{otherwise}\end{cases}$$

for $1\le i\le n$, $1\le\ell\le\delta_{2^\alpha}(i)$, and $1\le\alpha\le\lfloor\log k\rfloor$.

Proof By Lemma 2.4 and Proposition 3.5, it is obvious that if $\delta_{2^{\alpha-1}}(i)\neq 0$, then $L^{-1}_{2^\alpha}(i,1) = L^{-1}_{2^{\alpha-1}}(i,1)$; otherwise, $L^{-1}_{2^\alpha}(i,1) = 0$ for $1\le\alpha\le\lfloor\log k\rfloor$. All we have to consider is the case where $\ell\ge 2$. By definition, if $L_{2^\alpha}(i,j)=\ell$ and

	ℓ					
	1	2	3	4	5	6
$L_{21}^{-1}(1,\ell)$	1	2	3	4	5	9
$L_{21}^{-1}(2,\ell)$	2	3	4	5	7	9
$L_{21}^{-1}(3,\ell)$	3	4	5	6	7	9
$L_{21}^{-1}(4,\ell)$	4	5	6	7	9	0
$L_{21}^{-1}(5,\ell)$	5	6	7	9	0	0
$L_{21}^{-1}(6,\ell)$	6	7	8	9	0	0
$L_{21}^{-1}(7,\ell)$	7	8	9	0	0	0
$L_{21}^{-1}(8,\ell)$	8	9	0	0	0	0
$L_{21}^{-1}(9,\ell)$	9	0	0	0	0	0

(a) $L_{21}^{-1}(i,\ell)$

	ℓ								
	1	2	3	4	5	6	7	8	9
$L_{22}^{-1}(1,\ell)$	1	2	3	4	5	6	7	8	9
$L_{22}^{-1}(2,\ell)$	2	3	4	5	6	7	8	9	0
$L_{22}^{-1}(3,\ell)$	3	4	5	6	7	8	9	0	0
$L_{22}^{-1}(4,\ell)$	4	5	6	7	8	9	0	0	0
$L_{22}^{-1}(5,\ell)$	5	6	7	8	9	0	0	0	0
$L_{22}^{-1}(6,\ell)$	6	7	8	9	0	0	0	0	0
$L_{22}^{-1}(7,\ell)$	7	8	9	0	0	0	0	0	0
$L_{22}^{-1}(8,\ell)$	8	9	0	0	0	0	0	0	0
$L_{22}^{-1}(9,\ell)$	9	0	0	0	0	0	0	0	0

(b) $L_{22}^{-1}(i,\ell)$

Fig. 2 $L_{21}^{-1}(i,\ell)$ and $L_{22}^{-1}(i,\ell)$ where $S=\{agc, act, aatg, ttcg\}$ and $T=agactagtc$

$L_{2^\alpha}(i,j-1)=\ell-1$, then $L_{2^\alpha}^{-1}(i,\ell)=j$. According to Lemma 3.9, $L_{2^\alpha}(i,j)=\max_{1\le p\le L_{2^{\alpha-1}}(i,j)}\{p+L_{2^{\alpha-1}}(\beta_\alpha(i,p),j)\}=\ell$. This means that there exists a p, $1\le p\le L_{2^{\alpha-1}}(i,j)$, such that $p+L_{2^{\alpha-1}}(\beta_\alpha(i,p),j)=\ell$. By rearranging the previous formula, this yields $L_{2^{\alpha-1}}(\beta_\alpha(i,p),j)=\ell-p$. Since $L_{2^\alpha}(i,j)$ is an inverse function of $L_{2^\alpha}^{-1}(i,p)$ when $L_{2^\alpha}^{-1}(i,p)\neq 0$, $L_{2^{\alpha-1}}^{-1}(\beta_\alpha(i,p),\ell-p)=j$. This implies that the computation of $L_{2^\alpha}^{-1}(i,\ell)$ can be expressed as $\min_{1\le p\le\delta_{2^{\alpha-1}}(i)}\{L_{2^{\alpha-1}}^{-1}(\beta_\alpha(i,p),\ell-p)\}$, for $1\le i\le n$ and $1\le\ell\le\delta_{2^\alpha}(i)$, if $1\le\ell-p\le\delta_{2^{\alpha-1}}(\beta_\alpha(i,p))$. Recall that $L_{2^\alpha}^{-1}(i,p)=0$, for all $p>\delta_{2^\alpha}(i)$. Thus if $\ell-p>\delta_{2^{\alpha-1}}(i)$, then $L_{2^{\alpha-1}}^{-1}(i,\ell-p)=0$. This establishes the lemma. □

We use the computation of $L_{21}^{-1}(1,5)$ to illustrate Lemma 3.12. From $L_{21}^{-1}(1,5)$, we know that $\alpha=1$, $i=1$, and $\ell=5$. Thus $1\le p\le\delta_{2^{\alpha-1}}(i)=\delta_{20}(1)=4$. Furthermore, $\beta_1(1,1)=L_{20}^{-1}(1,1)+1=2$, $\beta_1(1,2)=3$, $\beta_1(1,3)=5$, and $\beta_1(1,4)=8$. Accordingly, $\delta_{20}(\beta_1(1,1))=\delta_{20}(2)=3$, $\delta_{20}(\beta_1(1,2))=\delta_{20}(3)=3$, $\delta_{20}(\beta_1(1,3))=\delta_{20}(5)=3$, and $\delta_{20}(\beta_1(1,4))=\delta_{20}(8)=2$. Thus we have the following derivation.

$$
\begin{aligned}
L_{21}^{-1}(1,5) &= \min_{\substack{1\le p\le 4\\ 1\le 5-p\le\delta_{21-1}(\beta_1(1,p))}}\left\{L_{20}^{-1}\big(\beta_1(1,p),5-p\big)\right\}\\
&= \min\left\{L_{20}^{-1}\big(\beta_1(1,2),5-2\big),L_{20}^{-1}\big(\beta_1(1,3),5-3\big),L_{20}^{-1}\big(\beta_1(1,4),5-4\big)\right\}\\
&= \min\left\{L_{20}^{-1}(3,3),L_{20}^{-1}(5,2),L_{20}^{-1}(8,1)\right\}\\
&= \min\{5,7,8\}\\
&= 5.
\end{aligned}
$$

Note that, in the above example, it is not necessary to consider the case where $p=1$ since $\delta_{20}(\beta_1(1,1))=\delta_{20}(2)=3$ which does not satisfy the inequality $1\le\ell-p\le\delta_{2^{\alpha-1}}(\beta_\alpha(i,p))$. Figures 2(a) and 2(b) show the resulting $L_{21}^{-1}(i,\ell)$ and $L_{22}^{-1}(i,\ell)$, respectively.

Lemma 3.13 describes a recurrence formula for computing $L_{k_\alpha}^{-1}(i,\ell)$ when k_α is not an integer to the power of 2.

Lemma 3.13

$$L_{k_\alpha}^{-1}(i,\ell) = \begin{cases} L_{2^\alpha}^{-1}(i,1) & \text{if } \ell = 1 \\ \min_{\substack{1\le p\le \delta_{2^\alpha}(i) \\ 1\le \ell-p\le \delta_{k_{\alpha-1}}(\beta_{\alpha+1}(i,p))}} \{L_{k_{\alpha-1}}^{-1}(\beta_{\alpha+1}(i,p), \ell-p)\} & \text{if } \ell \ge 2 \\ 0 & \text{otherwise,} \end{cases}$$

for $1 \le i \le n$, $1 \le \ell \le \delta_{k_\alpha}(i)$, *and* $1 \le \alpha \le \lfloor \log k \rfloor$.

Proof By definition, if $\ell < 1$, then $L_{k_\alpha}^{-1}(i,\ell) = 0$. For the case where $\ell = 1$, by the property that $L_{2^\alpha}^{-1}(i,1) = L_{2^{\alpha-1}}^{-1}(i,1)$, the value of $L_{2^\alpha}^{-1}(i,1)$, say j, will be the smallest index such that $L_{k_\alpha}(i,j) = 1$. Thus, by definition, $L_{k_\alpha}^{-1}(i,1) = L_{2^\alpha}^{-1}(i,1)$. Now we consider the case where $\ell \ge 2$. The argument for this case is similar to that of Lemma 3.12. We only need to replace 2^α and $2^{\alpha-1}$ with k_α and $k_{\alpha-1}$, respectively. By Lemma 3.8, we can obtain $L_{k_{\alpha-1}}(\beta_{\alpha+1}(i,p), j) = \ell - p$. After simplifying, the resulting formula follows. □

Here, we want to point out that a k-inlay can also be constructed from table $L_{k_\alpha}^{-1}$ (or $L_{2^\alpha}^{-1}$). For example, if the given k of the resequencing LCS problem is equal to 2, then Fig. 2(a) contains all of the lengths of the LCSs between k-inlays and all substrings of T. For example, the length of the LCS between a k-inlay and T is $\delta_{2^1}(1) = 6$ through which we can find $L_{2^1}^{-1}(1,6) = 9$ (see Fig. 2(a)). By using Lemmas 3.9 and 3.12 to traverse back, we can find that, by Lemma 3.12, $L_{2^1}^{-1}(1,6)$ is obtained from $L_{2^0}^{-1}(5,3)$ when $h = 3$, and, by Lemma 3.9, $p = 3$ occurs at $L_{2^0}(1,4)$ (see Fig. 1(d)). From table $L_{2^0}^{-1}$, $L_{2^0}^{-1}(5,3)$ corresponds to $L_{2^0}(5,9)$. By keeping track of the corresponding segments of S in table L_{2^0}, the value in $L_{2^0}(1,4)$ (respectively, $L_{2^0}(5,9)$) is obtained from L_{S_1} (respectively, L_{S_1} or L_{S_4}). Therefore, we can obtain two 2-inlays S_1S_1 or S_1S_4 with respect to T.

4 An Efficient Way for Finding Inverted Indexing Tables

By Lemmas 2.4, 3.9, 3.12, and 3.13, a k-inlay with respect to any substring of T can be found in $O(n\vartheta_k^2 \log k)$ time after table $L_{2^0}^{-1}$ is already constructed from S and T in the preprocessing step where S is a set of strings and T is a target string. The total time for constructing all aforementioned inverted indexing tables takes $O(n\vartheta_k^2 \log k)$ time. In this section, we propose an efficient way for computing each entry in tables $L_{k_\alpha}^{-1}$ and $L_{2^\alpha}^{-1}$, for $1 \le \alpha \le \lfloor \log k \rfloor$, in constant time so that constructing all inverted indexing tables can be done in $O(n\vartheta_k \log k)$ time.

To build each row, say i, of table $L_{2^{\alpha+1}}^{-1}$ efficiently, we construct a new inverted indexing table. Note that, in the inverted index representation of $L_{2^\alpha}(i,j)$, the index of each row is still i, for $1 \le i \le n$, the index of each column becomes ℓ, i.e., the

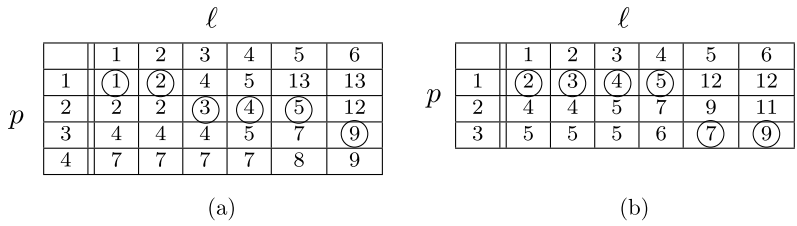

Fig. 3 $R^{-1}_{2^0,1}(p,\ell)$ and $R^{-1}_{2^0,2}(p,\ell)$

length of $L_{2^\alpha}(i,j)$, and its corresponding content is j. The new inverted indexing table is denoted by $R^{-1}_{2^\alpha,i}$. Both row and column indexes of $R^{-1}_{2^\alpha,i}$ are lengths and the content of $R^{-1}_{2^\alpha,i}(p,\ell)$ is j such that $L_{2^\alpha}(L^{-1}_{2^\alpha}(i,p)+1,j)=\ell-p$. By collecting those nonzero items considered in Lemma 3.12 on computing $L^{-1}_{2^{\alpha+1}}(i,\ell)$, for $1\le\ell\le\delta_{2^\alpha}(i)$, $R^{-1}_{2^\alpha,i}(p,\ell)$ is defined as follows:

$$R^{-1}_{2^\alpha,i}(p,\ell)=\begin{cases}L^{-1}_{2^\alpha}(i,p) & \text{if } \ell\le p,\\ L^{-1}_{2^\alpha}(x,\ell-p) & \text{if } 1\le\ell-p\le\delta_{2^\alpha}(x) \text{ and } x\le n,\\ n+\delta_{2^\alpha}(i)-p+1 & \text{otherwise,}\end{cases}\tag{6}$$

for $1\le p\le\delta_{2^\alpha}(i)$, $1\le\ell\le\delta_{2^{\alpha+1}}(i)$, $0\le\alpha\le\lfloor\log k\rfloor-1$, and $1\le i\le n$ where $x=\beta_{\alpha+1}(i,p)$.

Table $R^{-1}_{2^\alpha,i}$ is also an inverted indexing table. The value of $R^{-1}_{2^\alpha,i}(p,\ell)$ is the smallest position, say j, in T such that $L_{2^{\alpha+1}}(i,j)=L_{2^\alpha}(i,r)+L_{2^\alpha}(r+1,j)=\ell$ with $L_{2^\alpha}(i,r)=p$ and $L_{2^\alpha}(r+1,j)=\ell-p$. Note that $L_{2^\alpha}(i,r)+L_{2^\alpha}(r+1,j)$ is the formula used in Lemma 2.4 to find the length of the LCS of a $2^{\alpha+1}$-inlay with $T[i,j]$. For example, $R^{-1}_{2^0,1}$ is an auxiliary table for computing the first row of table L^{-1}_2, i.e., $L^{-1}_2(1,j)$, for $1\le j\le n$ (see Fig. 3(a)). Since $\delta_{2^0}(1)=4$ and $\delta_{2^1}(1)=6$, $R^{-1}_{2^0,1}$ is a 4×6 table. The first entry in the first row of table $R^{-1}_{2^0,1}$ is equal to $L^{-1}_{2^\alpha}(1,1)$ since $\ell\le p$ when $p=\ell=1$. The next three entries in the first row of table $R^{-1}_{2^0,1}$ are computed as follows:

$$\begin{aligned}R^{-1}_{2^0,1}(1,\ell)&=L^{-1}_{2^0}\big(\beta_1(1,1),\ell-1\big)\\&=L^{-1}_{2^0}(2,\ell-1),\end{aligned}$$

for $1\le\ell-1\le\delta_{2^0}(\beta_1(1,1))=3$, namely $2\le\ell\le 4$. We can find that $R^{-1}_{2^0,1}(1,2)=L^{-1}_{2^0}(2,2-1)=2$, $R^{-1}_{2^0,1}(1,3)=L^{-1}_{2^0}(2,3-1)=4$, and $R^{-1}_{2^0,1}(1,4)=L^{-1}_{2^0}(2,4-1)=5$. Finally, the last two entries of the first row are computed by $n+\delta_{2^\alpha}(i)-p+1=9+4-1+1=13$. Figures 3(a) and 3(b) depict $R^{-1}_{2^0,1}(p,\ell)$ and $R^{-1}_{2^0,2}(p,\ell)$, respectively.

In Fig. 4(a), we use a line to indicate the two related positions in table L_{2^0} for each of the above computations. The first four entries in the first row of table $R^{-1}_{2^0,1}$

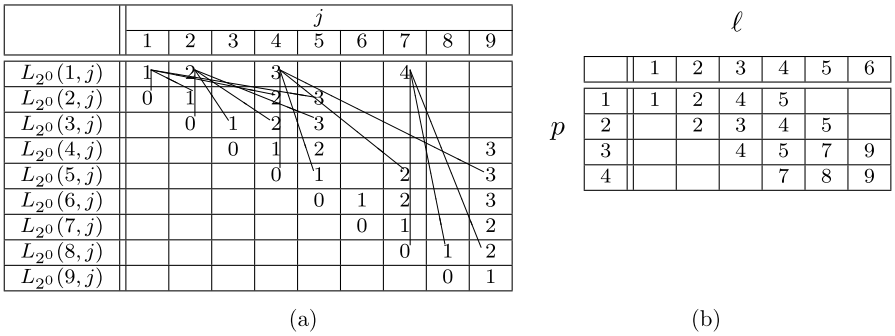

Fig. 4 An illustration for the meaning of $R^{-1}_{2^0,1}$

are computed by

$$L_1(1,1) + L_1(2,1) = 1 + 0 = 1,$$
$$L_1(1,1) + L_1(2,2) = 1 + 1 = 2,$$
$$L_1(1,1) + L_1(2,4) = 1 + 2 = 3, \quad \text{and}$$
$$L_1(1,1) + L_1(2,5) = 1 + 3 = 4.$$

Each of the above equations are indicated by a line in Fig. 4(a). Figure 4(b) shows the values of those related entries. Note that the entries in the lower and upper triangles of Fig. 4(b) have no values. However, to prove that this table has the totally monotone property, we add some values to those empty entries which are defined in table $R^{-1}_{2^\alpha,i}$. That is, if an entry in the lower triangle (as shown in Fig. 4(b)) is empty, then its value is set to the smallest value in that row. For an empty entry in the upper triangle, its value is set to $n + \delta_{2^\alpha}(i) - p + 1$. The values inside a circle in Fig. 3(a) and 3(b) are the values of the entries in the first and second, respectively, rows of table $L^{-1}_{2^1}$ in Fig. 2(a).

Lemma 4.1 describes the relation between tables $L^{-1}_{2^\alpha}$ and $R^{-1}_{2^{\alpha-1},i}$.

Lemma 4.1 $L^{-1}_{2^\alpha}(i,\ell) = \min_{1\le p\le \min\{\ell,\delta_{2^{\alpha-1}}(i)\}}\{R^{-1}_{2^{\alpha-1},i}(p,\ell)\}$, *for* $1 \le \alpha \le \lfloor \log k \rfloor$, $1 \le i \le n$, *and* $1 \le \ell \le \delta_{2^\alpha}(i)$.

Proof By Lemma 3.12 and the definition of $R^{-1}_{2^\alpha,i}(p,\ell)$, for $1 \le i \le n$ and $1 \le \alpha \le \lfloor \log k \rfloor$,

$$L^{-1}_{2^\alpha}(i,\ell) = \min_{\substack{1\le p\le \min\{\ell,\delta_{2^{\alpha-1}}(i)\} \\ 1\le \ell-p\le \delta_{2^{\alpha-1}}(\beta_\alpha(i,p))}} \left\{L^{-1}_{2^{\alpha-1}}\left(\beta_\alpha(i,p), \ell - p\right)\right\}$$
$$= \min_{1\le p\le \min\{\ell,\delta_{2^{\alpha-1}}(i)\}} \left\{R^{-1}_{2^{\alpha-1},i}(p,\ell)\right\}.$$

This completes the proof. □

We use the third column of Fig. 3(a), i.e., $R^{-1}_{2^0,1}(p,3)$, to illustrate Lemma 4.1. That is, we can find the smallest j such that $L_{2^1}(1,j)=3$, equivalently $L^{-1}_{2^1}(1,3)=j$, through the values in column 3 of table $R^{-1}_{2^0,1}$. Namely,

$$\begin{aligned} j &= \min_{1\le p\le \min\{3,\delta_1(1)\}}\left\{R^{-1}_{2^0,1}(p,3)\right\} \\ &= \min\left\{R^{-1}_{2^0,1}(1,3), R^{-1}_{2^0,1}(2,3), R^{-1}_{2^0,1}(3,3)\right\} \\ &= \{4,3,4\} \\ &= 3. \end{aligned}$$

Analogous to the definition of $R^{-1}_{2^\alpha,i}(p,\ell)$, we also define $R^{-1}_{k_\alpha,i}(p,\ell)$ as follows when k_α is not an integer to the power of 2.

$$R^{-1}_{k_\alpha,i}(p,\ell)=\begin{cases} L^{-1}_{2^\alpha}(i,p) & \text{if } \ell\le p, \\ L^{-1}_{k_{\alpha-1}}(x,\ell-p) & \text{if } 1\le \ell-p\le \delta_{k_{\alpha-1}}(x) \text{ and } x\le n, \\ n+\delta_{2^\alpha}(i)-p+1 & \text{otherwise,}\end{cases}$$

for $1\le p\le \delta_{2^\alpha}(i)$, $1\le \ell\le \delta_{k_\alpha}(i)$, $0\le\alpha\le\lfloor\log k\rfloor-1$, and $1\le i\le n$ where $x=\beta_{\alpha+1}(i,p)$.

Lemma 4.2 describes the relation between tables $L^{-1}_{k_\alpha}$ and $R^{-1}_{k_\alpha,i}$.

Lemma 4.2 *$L^{-1}_{k_\alpha}(i,\ell)=\min_{1\le p\le\min\{\ell,\delta_{2^\alpha}(i)\}}\{R^{-1}_{k_\alpha,i}(p,\ell)\}$ for $1\le\alpha\le\lfloor\log k\rfloor$, $1\le i\le n$, and $1\le\ell\le\delta_{k_\alpha}(i)$.*

Proof By Lemma 3.13 and the definition of $R^{-1}_{k_\alpha,i}(p,\ell)$, for $1\le i\le n$ and $1\le\alpha\le\lfloor\log k\rfloor$,

$$\begin{aligned} L^{-1}_{k_\alpha}(i,\ell) &= \min_{\substack{1\le p\le\min\{\ell,\delta_{2^\alpha}(i)\}\\ 1\le\ell-p\le\delta_{k_{\alpha-1}}(L^{-1}_{2^\alpha}(i,p)+1)}}\left\{L^{-1}_{k_{\alpha-1}}\left(L^{-1}_{2^\alpha}(i,p)+1,\ell-p\right)\right\} \\ &= \min_{1\le p\le\min\{\ell,\delta_{2^\alpha}(i)\}}\left\{R^{-1}_{k_\alpha,i}(p,\ell)\right\}. \end{aligned}$$

This completes the proof. □

For example, Figs. 5(a) and 5(b) are $R^{-1}_{k_1,1}(p,\ell)$ and $R^{-1}_{k_1,2}(p,\ell)$, for $k=3=2^1+2^0$. After finding the minimum values in each column of $R^{-1}_{k_1,1}(p,\ell)$ and $R^{-1}_{k_1,2}(p,\ell)$, we can find the corresponding values of $L^{-1}_{k_1}(1,\ell)$ and $L^{-1}_{k_1}(2,\ell)$, respectively (see Fig. 6).

By the definition of $R^{-1}_{2^\alpha,i}(p,\ell)$ (respectively, $R^{-1}_{k_\alpha,i}(p,\ell)$), each entry of table $R^{-1}_{2^\alpha,i}$ (respectively, $R^{-1}_{k_\alpha,i}$) can be computed in constant time. Thus, by Lemma 4.1 (respectively, Lemma 4.2), $L^{-1}_{2^\alpha}(i,\ell)$ (respectively, $L^{-1}_{k_\alpha}(i,\ell)$) can be calculated in $O(\delta_{2^{\alpha-1}}(i))$ (respectively, $O(\delta_{2^\alpha}(i))$) time when table $L^{-1}_{2^{\alpha-1}}$ (respectively, tables

p \ ℓ	1	2	3	4	5	6	7	8
1	1	2	4	5	15	15	15	15
2	2	2	3	4	5	14	14	14
3	3	3	3	4	5	9	13	13
4	4	4	4	4	5	7	9	12
5	5	5	5	5	5	6	7	9
6	9	9	9	9	9	9	10	10

(a) $R^{-1}_{k_1,1}(p,\ell)$

p \ ℓ	1	2	3	4	5	6	7	8
1	2	3	4	5	15	15	15	15
2	3	3	4	5	9	14	14	14
3	4	4	4	5	7	9	13	13
4	5	5	5	5	6	7	9	9
5	5	5	5	5	6	7	9	9
6	9	9	9	9	9	9	10	10

(b) $R^{-1}_{k_1,2}(p,\ell)$

Fig. 5 $R^{-1}_{k_1,1}(p,\ell)$ and $R^{-1}_{k_1,2}(p,\ell)$

Fig. 6 $L^{-1}_{k_1}(i,\ell)$

	ℓ 1	2	3	4	5	6	7	8
$L^{-1}_{k_1}(1,\ell)$	1	2	3	4	5	6	7	9
$L^{-1}_{k_1}(2,\ell)$	2	3	4	5	6	7	9	9
$L^{-1}_{k_1}(3,\ell)$	3	4	5	6	7	8	9	0
$L^{-1}_{k_1}(4,\ell)$	4	5	6	7	8	9	0	0
$L^{-1}_{k_1}(5,\ell)$	5	6	7	8	9	0	0	0
$L^{-1}_{k_1}(6,\ell)$	6	7	8	9	0	0	0	0
$L^{-1}_{k_1}(7,\ell)$	7	8	9	0	0	0	0	0
$L^{-1}_{k_1}(8,\ell)$	8	9	0	0	0	0	0	0
$L^{-1}_{k_1}(9,\ell)$	9	0	0	0	0	0	0	0

$L^{-1}_{2^\alpha}$ and $L^{-1}_{k_{\alpha-1}}$) is given. Obviously, table $L^{-1}_{2^\alpha}$ (respectively, $L^{-1}_{k_\alpha}$) can be built in $O(n\delta_{2^\alpha}(i)\delta_{2^{\alpha-1}}(i))$ (respectively, $O(n\delta_{k_\alpha}(i)\delta_{2^\alpha}(i))$) time. In the following, we apply a technique introduced in [1] to speed up the computations of the values in tables $L^{-1}_{2^\alpha}$ and $L^{-1}_{k_\alpha}$ so that these two tables can be built in $O(n\delta_{2^\alpha}(i))$ time and $O(n\delta_{k_\alpha}(i)))$ time, respectively.

An $m \times n$ matrix $M = (c_{i,j})_{m\times n}$ is called *totally monotone* [19] if either condition (1) or (2) below holds:

(1) $c_{i,r} \le c_{j,r}$ implies $c_{i,s} \le c_{j,s}$
(2) $c_{i,r} \ge c_{j,r}$ implies $c_{i,s} \ge c_{j,s}$

for all $1 \le i < j \le m$ and $1 \le r < s \le n$.

Totally monotone matrices have many nice properties and important applications. For surveys and recent applications of totally monotone matrices, please refer to [6, 9, 10, 19]. Lemma 4.3 describes that all row and column minima can be found efficiently.

Lemma 4.3 ([1]) *All column minima of an $m \times n$ totally monotone matrix can be determined in time $O(m)$ when $m \ge n$ and $O(m \log n/m)$ when $m < n$, provided that each entry in the matrix can be accessed in time $O(1)$.*

In Lemma 4.6, we show that both tables $R^{-1}_{2^\alpha,i}$ and $R^{-1}_{k_\alpha,i}$ satisfy condition (2) of the totally monotone property. In the proof of Lemma 4.6, we shall use a property

described in Lemma 4.5 which is derived from a result proposed by Alves, Cáceres, and Song in [3] which is described in Lemma 4.4.

Lemma 4.4 (Property 2.1(3) in [3]) *For all i and all j $(1 \le i < j \le n)$, if $L_X(i-1, j) = L_X(i-1, j-1) + 1$, then $L_X(i, j) = L_X(i, j-1) + 1$.*

Lemma 4.5 *For all i and all j $(1 \le i < j \le n)$, if $L_X(i-x, j) = L_X(i-x, j-y) + 1$, then $L_X(i, j) = L_X(i, j-y) + 1$, for some $1 \le x \le i$ and some $1 \le y < j - i$.*

Proof By repeatedly applying Lemma 4.4, this lemma follows. □

Lemma 4.6 *Tables $R^{-1}_{2^\alpha,i}$ and $R^{-1}_{k_\alpha,i}$ are totally monotone matrices, for $0 \le \alpha \le \lfloor \log k \rfloor - 1$ and $1 \le i \le n$.*

Proof We only prove that matrix $R^{-1}_{2^\alpha,i}$ is a totally monotone matrix. With a similar reasoning, we can also prove that matrix $R^{-1}_{k_\alpha,i}$ is a totally monotone matrix. By the definition of totally monotone matrices, it suffices to show that if $R^{-1}_{2^\alpha,i}(p, \ell) \ge R^{-1}_{2^\alpha,i}(p+x, \ell)$, then $R^{-1}_{2^\alpha,i}(p, \ell+1) \ge R^{-1}_{2^\alpha,i}(p+x, \ell+1)$ for $1 \le x \le \delta_{2^\alpha}(i) - p$. By the construction of $R^{-1}_{2^\alpha,i}$, we consider the following three cases.

Case 1. $\ell \le p$. In this case, $R^{-1}_{2^\alpha,i}(p, \ell) = L^{-1}_{2^\alpha}(i, p)$ and $R^{-1}_{2^\alpha,i}(p+x, \ell) = L^{-1}_{2^\alpha}(i, p+x)$. Since $L^{-1}_{2^\alpha}(i, p+x) > L^{-1}_{2^\alpha}(i, p)$, $R^{-1}_{2^\alpha,i}(p, \ell) < R^{-1}_{2^\alpha,i}(p+x, \ell)$, for $1 \le x \le \delta_{2^\alpha}(i) - p$. Thus this case is impossible.

Case 2. $1 \le \ell - p \le \delta_{2^\alpha}(\beta_\alpha(i, p))$ and $\beta_\alpha(i, p) \le n$. By the assumption that $R^{-1}_{2^\alpha,i}(p, \ell) \ge R^{-1}_{2^\alpha,i}(p+x, \ell)$, we may assume that $R^{-1}_{2^\alpha,i}(p, \ell) = j$ and $R^{-1}_{2^\alpha,i}(p+x, \ell) = j'$ with $j' \le j$. By $R^{-1}_{2^\alpha,i}(p, \ell) = j$, this implies that there exists $i \le r \le j$ such that $L_{2^{\alpha+1}}(i, j) = L_{2^\alpha}(i, r) + L_{2^\alpha}(r+1, j) = \ell$ with $L_{2^\alpha}(i, r) = p$ and $L_{2^\alpha}(r+1, j) = \ell - p$. By $R^{-1}_{2^\alpha,i}(p+x, \ell) = j'$, this implies that there exist $r < r' \le j' \le j$ such that $L_{2^{\alpha+1}}(i, j') = L_{2^\alpha}(i, r') + L_{2^\alpha}(r'+1, j') = \ell$ with $L_{2^\alpha}(i, r') = p + x$ and $L_{2^\alpha}(r+1, j') = \ell - (p+x)$. Accordingly, $R^{-1}_{2^\alpha,i}(p, \ell+1)$ yields $L_{2^{\alpha+1}}(i, j+d_1) = L_{2^\alpha}(i, r) + L_{2^\alpha}(r+1, j+d_1) = \ell + 1$ with $L_{2^\alpha}(i, r) = p$ and $L_{2^\alpha}(r+1, j+d_1) = \ell + 1 - p$, and $R^{-1}_{2^\alpha,i}(p+x, \ell+1)$ yields $L_{2^{\alpha+1}}(i, j'+d_2) = L_{2^\alpha}(i, r') + L_{2^\alpha}(r'+1, j'+d_2) = \ell + 1$ with $L_{2^\alpha}(i, r') = p + x$ and $L_{2^\alpha}(r'+1, j'+d_2) = \ell + 1 - (p+x)$. All we have to prove is that $d_2 \le d_1$ holds. By Lemma 4.5, if $L_X(r+1, j+d_1) = L_X(r+1, j) + 1$, then $L_X(r'+1, j+d_1) = L_X(r'+1, j) + 1$, for $r < r' \le j < j + d_1$. That is, if $L_{2^\alpha}(r+1, j+d_1) = L_{2^\alpha}(r+1, j) + 1$, then $L_{2^\alpha}(r'+1, j+d_1) = L_{2^\alpha}(r'+1, j) + 1$, for $r < r' \le j < j + d_1$. As a consequence, $L_{2^\alpha}(r'+1, j+d_1) = L_{2^\alpha}(r'+1, j) + 1 \ge L_{2^\alpha}(r'+1, j') + 1 = \ell + 1 - (p+x)$. This implies that there exists d_2 with $d_2 \le d_1$ such that $L_{2^\alpha}(r'+1, j'+d_2) = \ell + 1 - (p+x)$. Thus this case holds.

Case 3. $\ell - p > \delta_{2^\alpha}(\beta_\alpha(i, p))$ or $\beta_\alpha(i, p) > n$. In this case, by definition, $R^{-1}_{2^\alpha,i}(p, \ell) = R^{-1}_{2^\alpha,i}(p, \ell+1) = n + \delta_{2^\alpha}(i) - p + 1$. If $1 \le (\ell+1) - (p+x) \le \delta_{2^\alpha}(\beta_\alpha(i, p+x))$ and $\beta_\alpha(i, p+x) \le n$, then, by definition, $R^{-1}_{2^\alpha,i}(p+x, \ell+1) = L^{-1}_{2^\alpha}(\beta_{\alpha+1}(i, p+x), (\ell+1) - (p+x)) \le n < n + \delta_{2^\alpha}(i) - p + 1 = R^{-1}_{2^\alpha,i}(p, \ell+1)$;

otherwise, $R^{-1}_{2^\alpha,i}(p+x,\ell+1) = n+\delta_{2^\alpha}(i)-(p+x)+1 < n+\delta_{2^\alpha}(i)-p+1 = R^{-1}_{2^\alpha,i}(p,\ell+1)$. This completes the proof. □

By Lemmas 4.3 and 4.6, it is easy to see that all entries in a row of $L^{-1}_{2^\alpha}$ (respectively, $L^{-1}_{k_\alpha}$) can be computed in $O(\delta_{2^\alpha}(i))$ (respectively, $O(\delta_{k_\alpha}(i))$) time. Therefore, all entries of table $L^{-1}_{2^\alpha}$ (respectively, $L^{-1}_{k_\alpha}$) can be computed in $O(n\delta_{2^\alpha}(i))$ (respectively, $O(n\delta_{k_\alpha}(i))$) time. Now we are at a position to describe our algorithm for solving the resequencing LCS problem. For simplicity, we only describe the query procedure as follows.

Algorithm `Mosaic1`

Input: A positive integer k and table $L^{-1}_{2^0}(i,\ell)$, for $1 \le i \le n$ and $1 \le \ell \le \delta_{2^0}(1)$.
Output: $L_k(i,j)$ for $1 \le i \le j \le n$.
Step 1. Compute $L^{-1}_{2^\alpha}(i,\ell)$ by using the formula in Lemma 4.1, for $1 \le \alpha \le \lfloor \log k \rfloor$, $1 \le i \le n$, and $1 \le \ell \le \delta_{2^\alpha}(i)$.
Step 2. If $k = 2^w$, then find $L_k(i,j)$, for $1 \le i \le j \le n$, from table $L^{-1}_{2^w}$ and return.
Step 3. If $k \ne 2^w$, then do the following substeps.

Substep 3.1. Compute $L^{-1}_{k_\alpha}(i,\ell)$ by using the formulas in Lemma 4.2 and Propositions 3.6 and 3.7, for $1 \le \alpha \le \lfloor \log k \rfloor$, $1 \le i \le n$, and $1 \le \ell \le \delta_\alpha(i)$.
Substep 3.2. Find $L_k(i,j)$, for $1 \le i \le j \le n$, from table $L^{-1}_{k_w}$ and return.

We summarize our result as the following theorem.

Theorem 4.7 *The preprocessing procedure of the resequencing LCS problem can be done in $O(nml)$ time. The query procedure of the resequencing LCS problem can be done in $O(n\vartheta_k \log k)$ time. Furthermore, inlays with respect to all substrings of T can also be found.*

Proof The correctness of Algorithm `Mosaic1` directly follows from Propositions 3.6 and 3.7, Lemmas 3.8, 3.9, and 4.6. Now we analyze the time-complexity of Algorithm `Mosaic1`. Using the formulas in Lemmas 4.1 and 4.2, Step 1 and Substep 3.1, respectively, can be done in $O(n\vartheta_k \log k)$ time. Clearly, in Step 2 and Substep 3.2, $L_k(i,j)$, namely the lengths of the LCSs of all k-inlays with T, for $1 \le i \le j \le n$, can be found in $O(n\vartheta_k)$ time. Therefore, the query time is $O(n\vartheta_k \log k)$. This completes the proof. □

5 Computing $L_k(1, j)$

In this section, we introduce how to compute $L_k(1,j)$ for $1 \le j \le n$ efficiently. Since $L_k(1,j)$, for $1 \le j \le n$, are always from position 1 of T, we can modify tables $L^{-1}_{k_\alpha}$

Fig. 7 $\mathcal{R}_4^{-1}(p,\ell)$

p \ ℓ	1	2	3	4	5	6	7	8	8
1	1	2	4	5	17	17	17	17	17
2	2	2	3	4	5	16	16	16	16
3	3	3	3	4	5	9	15	15	15
4	4	4	4	4	5	7	9	14	14
5	5	5	5	5	5	6	7	9	13
6	6	6	6	6	6	6	7	9	12
7	7	7	7	7	7	7	7	8	9
8	9	9	9	9	9	9	9	9	10

and $R_{k_\alpha,i}^{-1}$ so that they can be computed efficiently. The modified tables are named as $\mathcal{L}_h^{-1}(\ell)$ and $\mathcal{R}_h^{-1}(p,\ell)$, respectively, for $1 \le h \le k$. First, we define $\mathcal{L}_h^{-1}(\ell)$ as follows:

$$\mathcal{L}_h^{-1}(\ell) = \begin{cases} L_{h-1}^{-1}(1) & \text{if } \ell = 1 \\ \min\limits_{\substack{1 \le p \le \delta_{h-1}(1) \\ 1 \le \ell - p \le \delta_{2^0}(\beta'_h(p))}} \{L_{2^0}^{-1}(\beta'_h(p), \ell - p)\} & \text{if } \ell \ge 2 \\ 0 & \text{otherwise,} \end{cases}$$

for $1 \le \ell \le \delta_h(1)$ and $2 \le h \le k$, where $\beta'_h(p) = \mathcal{L}_{h-1}^{-1}(p) + 1$.

In addition, table $\mathcal{R}_h^{-1}(p,\ell)$ is defined as follows.

$$\mathcal{R}_h^{-1}(p,\ell) = \begin{cases} \mathcal{L}_{h-1}^{-1}(p) & \text{if } \ell \le p, \\ L_{2^0}^{-1}(\beta'_h(p), \ell - p) & \text{if } 1 \le \ell - p \le \delta_{2^0}(\beta'_h(p)) \text{ and } \beta'_h(p) \le n, \\ n + \delta_{h-1}(1) - p + 1 & \text{otherwise,} \end{cases}$$

for $1 \le p \le \delta_{h-1}(1)$, $1 \le \ell \le \delta_h(1)$, and $2 \le h \le k$.

Analogous to Lemma 4.1, Lemma 5.1 describes the relation between tables $\mathcal{L}_h^{-1}$ and $\mathcal{R}_h^{-1}$ and we also omit the proof. Note that we use $\mathcal{R}_h^{-1}$ as an auxiliary table for computing $\mathcal{L}_h^{-1}$ but not $\mathcal{L}_{h+1}^{-1}$.

Lemma 5.1 $\mathcal{L}_h^{-1}(\ell) = \min_{1 \le p \le \min\{\ell, \delta_{h-1}(1)\}} \{\mathcal{R}_h^{-1}(p,\ell)\}$, *for* $1 \le \ell \le \delta_\alpha(1)$ *and* $2 \le h \le k$.

Figure 7 is an illustration for computing $\mathcal{L}_4^{-1}$ by using the formula described in Lemma 5.1. After finding the minimum value in each column of $\mathcal{R}_4^{-1}(p,\ell)$, we can find the corresponding values of $\mathcal{L}_4^{-1}(\ell)$ (see Fig. 2(b)). Note that $\mathcal{R}_4^{-1}(p,\ell)$ itself is computed through tables $\mathcal{L}_3^{-1}$ and $L_{2^0}^{-1}$ (see the first row in Figs. 6 and 1(d), respectively).

Similar to Lemma 4.6, Lemma 5.2 shows that matrices $\mathcal{R}_h^{-1}$ are totally monotone matrices, for $h = 2, 3, \ldots, k$, and we omit the proof.

Lemma 5.2 *Matrices* $\mathcal{R}_h^{-1}$ *are totally monotone matrices, for* $h = 2, 3, \ldots, k$.

With a minor modification of Algorithm `Mosaic1`, Algorithm `Mosaic2` is proposed as follows.

Algorithm `Mosaic2`

Input: A positive integer k and $L_{2^0}^{-1}(i, \ell)$, $1 \le i \le n$ and $1 \le \ell \le \delta_{2^0}(1)$.
Output: $L_k(1, j)$, for $1 \le j \le n$.
Step 1. Compute $\mathcal{L}_h^{-1}(\ell)$ by using the formula in Lemma 5.1 for $1 \le \ell \le \delta_h(1)$ where $h = 2, 3, \ldots, k$.
Step 2. Return $L_k(1, j)$, for $1 \le j \le n$, by scanning $\mathcal{L}_k^{-1}(\ell)$.

Theorem 5.3 *The query procedure for finding inlays with respect to all substrings $T(1, j)$, $1 \le j \le n$, can be done in $O(\vartheta_k k)$ time.*

Proof The correctness of Algorithm `Mosaic2` is similar to that of Algorithm `Mosaic1`. In Algorithm `Mosaic2`, by using the formula in Lemma 5.1, Step 1 can be done in $O(\vartheta_k k)$ time. Clearly, Step 2 takes at most $O(\vartheta_k)$ time. Note that if consecutive entries of T have the same $L_k(1, j)$, we only return one value of $L_k(1, j)$ companied with this consecutive interval. This completes the proof. □

6 Concluding Remarks

In this paper, we propose two algorithms for solving the resequencing LCS problem. Both algorithms take $O(nml)$ preprocessing time. The time-complexities of their query time are $O(n\vartheta_k \log k)$ and $O(\vartheta_k k)$, respectively, where the former is used to find inlays with respect to all substrings of T while the latter is for finding inlays with respect to all substrings $T(1, j)$, $1 \le j \le n$. The reason why our proposed algorithms can be performed efficiently is that tables $R_{2^\alpha,i}^{-1}$ have the totally monotone property. This raises the following interesting questions: (1) Is it possible to further improve the results? (2) Which kind of inverted indexing tables have the totally monotone property?

Acknowledgements The authors would like to thank anonymous referees for their careful reading with corrections and useful comments which helped to improve the paper.

References

1. Aggarwal, A., Klawe, M.M., Moran, S., Shor, P., Wilber, R.: Geometric applications of a matrix-searching algorithm. Algorithmica **2**(1), 195–208 (1987)
2. Aho, A., Hirschberg, D., Ullman, J.: Bounds on the complexity of the longest common subsequence problem. J. ACM **23**(1), 1–12 (1976)
3. Alves, C.E.R., Cáceres, E.N., Song, S.W.: An all-substrings common subsequence algorithm. Discrete Appl. Math. **156**(7), 1025–1035 (2008)
4. Amir, A., Hartman, T., Kapah, O., Shalom, R., Tsur, D.: Generalized LCS. Theor. Comput. Sci. **409**(3), 438–449 (2008)
5. Amir, A., Gothilf, T., Shalom, R.: Weighted LCS. In: Proceedings of Combinatorial Algorithms: 20th International Workshop, IWOCA 2009, pp. 36–47 (2009)
6. Bein, W.W., Golin, M.J., Larmore, L.L., Zhang, Y.: The Knuth-Yao quadrangle-inequality speedup is a consequence of total-monotonicity. In: Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006), pp. 31–40 (2006)

7. Bergroth, L., Hakonen, H., Raita, T.: A survey of longest common subsequence algorithms. In: Proceedings of 7th Symposium on String Processing and Information Retrieval (SPIRE 2000), pp. 39–48 (2000)
8. Brent, R.P.: The parallel evaluation of general arithmetic expressions. J. ACM **21**, 201–206 (1974)
9. Burkard, R.E.: Monge properties, discrete convexity and applications. Eur. J. Oper. Res. **176**(1), 1–14 (2007)
10. Burkard, R.E., Klinz, B., Rudolf, R.: Perspectives of Monge properties in optimization. Discrete Appl. Math. **70**(2), 95–161 (1996)
11. Chvatal, V., Klarner, D.A., Knuth, D.E.: Selected combinatorial research problem. Technical Report CS-TR-72-292, Stanford University (1972)
12. Demaine, E.D., Mozes, S., Rossman, B., Weimann, O.: An optimal decomposition algorithm for tree edit distance. In: Proc. 34th International Colloquium on Automata, Languages and Programming (ICALP). Lecture Notes in Computer Science, vol. 4596, pp. 146–157 (2007)
13. Gusfield, D.: Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge University Press, Cambridge (1997)
14. Hirschberg, D.S.: Algorithms for the longest common subsequence problem. J. ACM **24**(4), 664–675 (1977)
15. Huang, K.S., Yang, C.B., Tseng, K.T., Peng, Y.H., Ann, H.Y.: Dynamic programming algorithms for the mosaic longest common subsequence problem. Inf. Process. Lett. **102**, 99–103 (2007)
16. Knuth, D.E.: The Art of Computer Programming, pp. 560–563. Addison-Wesley, Reading (1973)
17. Komatsoulis, G.A., Waterman, M.S.: Chimeric alignment by dynamic programming: algorithm and biological uses. In: RECOMB97: Proceedings of the First Annual International Conference on Computational Molecular Biology, pp. 174–180. ACM Press, New York (1997)
18. Komatsoulis, G.A., Waterman, M.S.: A new computational method for detection of chimeric 16S rRNA artifacts generated by PCR amplification from mixed bacterial populations. Appl. Environ. Microbiol. **63**(6), 2338–2346 (1997)
19. Landau, G.M., Ziv-Ukelson, M.: On the common substring alignment problem. J. Algorithms **41**(2), 338–359 (2001)
20. Liu, J.J., Wang, Y.L., Lee, R.C.T.: Finding a longest common subsequence between a run-length-encoded string and an uncompressed string. J. Complex. **24**, 173–184 (2008)
21. Masek, W.J., Paterson, M.S.: A faster algorithm computing string edit distances. J. Comput. Syst. Sci. **20**, 18–31 (1980)
22. Modelevsky, J.: Computer applications in applied genetic engineering. Adv. Appl. Microbiol. **30**, 169–195 (1984)
23. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. J. ACM **21**(1), 168–173 (1974)
24. Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. SIAM J. Comput. **18**(6), 1245–1262 (1989)