Theoretical Computer Science 911 (2022) 41–54

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

A data structure for substring-substring LCS length queries

Yoshifumi Sakai

Graduate School of Agricultural Science, Tohoku University, 468-1, Aoba, Aramaki, Aoba-ku, Sendai 980-0845, Japan

ARTICLE INFO

Article history:
Received 14 September 2021
Received in revised form 13 January 2022
Accepted 8 February 2022
Available online 14 February 2022
Communicated by R. Giancarlo

Keywords:
Algorithms
Longest common subsequence
Semi-local string comparison

ABSTRACT

The longest common subsequence (LCS) length of two strings is used as one of the most fundamental metrics measuring the similarity between the strings. To find out the local structures common to the strings under this similarity metric, we need a fast calculation of the LCS length of any pair of substrings of the two strings. For supporting such queries, it makes sense to preprocess the two strings in a quadratic time, because it takes about the same amount of time to compute the LCS length of the entire strings from scratch. We propose a quadratic-time constructible data structure that supports sublinear-time queries of the LCS length for any pair of substrings. The query time is $O(\sqrt{l}\log^{1+\epsilon} l)$, where ϵ is a positive constant arbitrarily small and l is the sum of the substring lengths.

1. Introduction

Measuring the similarity between two strings (i.e., sequences of symbols) has many important applications, such as data compression, pattern recognition, data mining, and biological sequence comparison. The longest common subsequence (LCS) length is one of the most fundamental similarity metrics in widespread use. The LCS length of two strings is defined as the greatest possible length of any subsequence common to the strings, where a subsequence of a string is the string after deleting zero or more symbols at any position (not necessarily contiguous). Due to the simplicity of its definition and the wide range of applications, not only the original LCS length problem but also a number of related problems have been enthusiastically studied. Such related problems include, for example, the parameterized LCS problems [2,3,8,11–13,16,17], the conditional LCS problems (such as the constrained LCS problem [6,7,24] and the restricted LCS problem [6,10]), and the reductions of the rational-weighted variants of other metrics (including the edit distance [22] and the dynamic time warping distance [21]) to the LCS length.

Focusing on the original problem, it is well known that the LCS length of any pair of strings both of length $O(n)$ can be computed in $O(n^2)$ time using the dynamic programming algorithm [25] (and the Four-Russians technique reduces this running time by a logarithmic factor [15]). It was also revealed that, unless the strong exponential time hypothesis (SETH) does not hold, for any positive constant ε, no $O(n^{2-\varepsilon})$-time algorithm can compute the LCS length [1,4]. This implies that there exists only a slight gap between the asymptotic lower and upper bounds of the time complexity under the SETH assumption.

Suppose that we want to seek for the local structures common to the two strings, by checking the LCS length of each of the pairs of substrings (i.e., contiguous subsequences) of the strings possibly having high similarity. Here, the next pair of substrings whose LCS length is to be computed may be given depending on the LCS lengths previously determined, or even given arbitrarily by the adversary, in an online environment. Thus, we need a fast calculation of the LCS length for any pair of substrings of the strings. A natural idea to deal with this situation is to prepare a quickly constructible data structure to

E-mail address: yoshifumi.sakai.c7@tohoku.ac.jp.

https://doi.org/10.1016/j.tcs.2022.02.004
0304-3975/

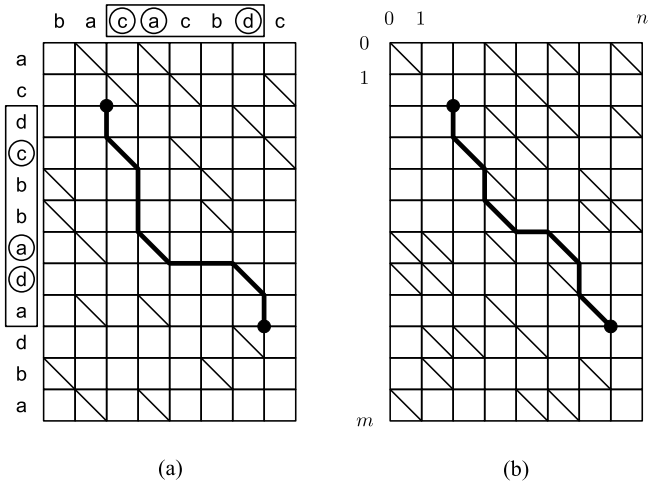

Fig. 1. (a) Grid graph $G_{A,B}$ for $A = \mathsf{acdcbbadadba}$ and $B = \mathsf{bacacbdc}$, where the path indicated by the thick polygonal line represents an LCS cad of substrings $A' = \mathsf{dcbbada}$ of A and $B' = \mathsf{cacbd}$ of B; (b) a concrete example of grid graph G with $m = 12$ and $n = 8$, where the path indicated by the thick polygonal line represents a shortest path between vertices $(2, 2)$ and $(9, 7)$.

support fast such queries. We call such a data structure a substring-substring LCS length data structure. Since it takes an almost quadratic time to compute the LCS length of a pair of substrings from scratch as mentioned earlier, it may make sense to preprocess the two strings in quadratic time to construct the substring-substring LCS length data structure. The aim of this article is to propose such a data structure for any pair of strings, which is hence quadratic-time constructible and supports subquadratic-time queries of the LCS length for any pair of substrings of the strings.

Considering the trade-off between the size of the substring-substring LCS length data structure and its query time, there seems to be a consensus regarding the sum of the exponents of the data structure size and the query time as follows. For any pair of $O(n)$-length strings A and B, the strings themselves immediately compose a naive data structure of size $O(n)$ supporting almost $O(l^2)$-time queries for any pair of $O(l)$-length substrings, by computing the LCS length of them from scratch. On the other hand, the semi-local LCS length framework of Tiskin [22] allows us to construct a data structure of size $O(n^3)$ that supports $O(\log n)$-time queries. In this framework, a set of $O(n)$ two-dimensional points for any pair of $O(n)$-length strings is used to represent the LCS length of either any pair of a substring of one string and the entire string the other or any pair of a prefix of one string and a suffix of the other as the number of points located in a rectangular region. Implementing this set as the two-dimensional range counting tree [5] immediately yields a data structure of size $O(n)$ supporting $O(\log n)$-time semi-local LCS length queries. Hence, the collection of this semi-local LCS length data structure for A and B' over all substrings B' of B works as a substring-substring LCS length data structure supporting $O(\log n)$-time queries. Since the number of substrings of B is $O(n^2)$, the size of this collection is $O(n^3)$. The last example is the data structure of size $O(n^2)$ that supports $O(l)$-time queries of not only the LCS length but also an LCS itself for any pair of $O(l)$-length substrings [20]. Ignoring any poly-logarithmic factor, the sum of the exponents of the size and the query time for any of the three data structures with different sizes and query times is three. The challenge in this article is to break this consensus by achieving a sublinear query time on a quadratic-time constructible (and hence quadratic-size) data structure.

1.1. Our contribution

In this article, we propose for any positive constant ϵ and any pair of an m-length string A and an n-length string B, an $O(mn)$-time constructible data structure that supports $O(\sqrt{l}\log^{1+\epsilon} l)$-time queries of the LCS length of A' and B' for any pair of an m'-length substring A' of A and an n'-length substring B' of B with $m' + n' = l$.

The problem of computing the LCS length for any pair of substrings A' and B' can be reduced to the problem of computing the shortest path length of a grid graph $G_{A,B}$ introduced later, where the shortest path length between a pair of vertices is the least possible number of edges that can compose a path between the vertices. The proposed data structure is designed to support queries of the shortest path length on $G_{A,B}$ between any pair of vertices. The grid graph $G_{A,B}$ consists of m rows and n columns of grid units, where the ith row corresponds to the ith symbol of A, and the jth column corresponds to the jth symbol of B (see Fig. 1(a)). Each grid unit has a diagonal edge, if and only if the corresponding symbols in A and B are identical. Due to this definition, the LCS length of A' and B' is equal to the number of diagonal edges in any shortest path between the vertices corresponding to the left and right ends of A' and B'. Therefore, the LCS length of A' and B' can be obtained as the sum of the lengths of A' and B' minus the shortest path length on $G_{A,B}$ between these vertices.

Our approach to designing the proposed data structure achieves both the preprocessing and query times without using conditions regarding the number and position of diagonal edges in the grid graph. Hence, the technique we develop applies

to any grid graph G consisting of m rows and n columns of grid units, each potentially having a diagonal edge (see Fig. 1(b)), whether or not $G = G_{A,B}$ for some pair of strings A and B. We propose in this article, for any such grid graph G and any positive constant ϵ, an $O(mn)$-time constructible data structure that supports $O(\sqrt{l}\log^{1+\epsilon} l)$-time queries of the shortest path length for any pair of vertices (i, j) and (i', j') with $0 \le i < i' \le m$ and $0 \le j < j' \le n$, where $l = (i' - i) + (j' - j)$. Note that if we set G to $G_{A,B}$, then this data structure works as a substring-substring LCS length data structure we aim to.

The rest of this article is organized as follows. Section 2 defines notations and terminology used in this article and introduces the semi-local LCS length technique of Tiskin [22], based on which the proposed data structure is designed. Section 3 presents an $O(mn)$-time constructible data structure supporting $O(l)$-time queries of the shortest path length between any pair of vertices in G, and Section 4 modifies this so as to support $O(\sqrt{l}\log^{1+\epsilon} l)$-time queries with no asymptotic increase of the preprocessing time. Section 5 concludes this article.

2. Preliminaries

Let m and n be arbitrary positive integers. Let G be an undirected grid graph consisting of all pairs (i, j) of integers with $0 \le i \le m$ and $0 \le j \le n$ representing vertices, *vertical edges* between $(i-1, j)$ and (i, j) for all pairs of such vertices, *horizontal edges* between $(i, j-1)$ and (i, j) for all pairs of such vertices, and *diagonal edges* between $(i-1, j-1)$ and (i, j) for arbitrary pairs of such vertices. A *grid unit* of G is a subgraph of G induced by four vertices $(i-1, j-1)$, $(i-1, j)$, $(i, j-1)$, and (i, j) with $1 \le i \le m$ and $1 \le j \le n$. Note that G has mn grid units.

For any vertex w in G, let i_w (resp. j_w) denote the *vertical coordinate* (resp. the *horizontal coordinate*) of w, so that $w = (i_w, j_w)$. In addition, let d_w denote the *diagonal coordinate* $j_w - i_w$ of w, and let a_w denote the even integer $2\lfloor (i_w + j_w)/2 \rfloor$, which approximately represents the *anti-diagonal coordinate* $i_w + j_w$ of w. For any vertices w and w' in G, let $w \searrow w'$ (resp. $w \nearrow w'$) mean that both $i_w < i_{w'}$ and $j_w < j_{w'}$ (resp. $i_w \ge i_{w'}$ and $j_w \le j_{w'}$) hold. Note that if $w \searrow w'$, then none of $w \nearrow w'$, $w' \nearrow w$, and $w' \searrow w$ holds, and also that if $w \nearrow w'$ and $w \ne w'$, then none of $w \searrow w'$, $w' \searrow w$ and $w' \nearrow w$ holds.

For any vertices w and w' in G, we define the length of a path between w and w' as the number of edges in the path. The shortest path length between w and w' is the least possible number of edges that compose a path between w and w'. A shortest path between w and w' is a path between w and w' whose length is equal to the shortest path length between w and w'. If $w \nearrow w'$, then the shortest path length between w and w' is exactly equal to $(i_w - i_{w'}) + (j_{w'} - j_w)$, because any shortest path passes through no diagonal edge. In contrast, if $w \searrow w'$, then this length varies depending on the location of diagonal edges in G, although it is between $\max(i_{w'} - i_w, j_{w'} - j_w)$ and $(i_{w'} - i_w) + (j_{w'} - j_w)$, and is hence $\Theta(a_{w'} - a_w)$. For any pair of vertices w and w' in G with $w \searrow w'$, let $l(w, w')$ denote the shortest path length between w and w'.

The aim of this article is to propose an $O(mn)$-time constructible data structure that supports $O(\sqrt{a_v - a_u}\log^{1+\epsilon}(a_v - a_u))$-time queries of $l(u, v)$ for any pair of vertices u and v in G with $u \searrow v$, where ϵ is an arbitrary positive constant. We design such a data structure based on (a straightforward generalization [19] of) the semi-local LCS length technique due to Tiskin [22]. To introduce this technique, we use the following notations and terminology.

For any vertices w and w' in G with $w \nearrow w'$, we call any shortest path between w and w', hence consisting of $i_w - i_{w'}$ vertical edges and $j_{w'} - j_w$ horizontal edges in any order, a *diagonally-incremental* (*d-inc*) *path*, because the diagonal coordinate of each vertex in the path from w to w' increases incrementally. We sometimes treat any *non-diagonal* (i.e., vertical or horizontal) edge as a d-inc path of length one, and also treat any vertex as a d-inc path of length zero. For any d-inc path R between w and w' with $w \nearrow w'$, we call w (resp. w') the *diagonally lower* (*d-lower*) (resp. *diagonally upper* (*d-upper*)) *end vertex* of R, and denote it by ${}_*R$ (resp. R^*). If R consists of a single vertex, then both ${}_*R$ and R^* represent the only vertex composing R. For any d-inc path R and any grid unit g, we use $R \searrow g$ (resp. $g \searrow R$) to mean that there exist a vertex w in R and a vertex w' in g such that $w \searrow w'$ (resp. $w' \searrow w$). Similarly, for any d-inc paths R and R', let $R \searrow R'$ mean that there exist a vertex w in R and a vertex w' in R' such that $w \searrow w'$. On the other hand, we use $R \nearrow R'$ to mean that $R^* \nearrow {}_*R'$. Furthermore, we say that non-diagonal edges $e_1, e_2, \ldots, e_\ell$ on any d-inc path are in the *diagonally ascending* (resp. *descending*) *order*, if $e_1 \nearrow e_2 \nearrow \cdots \nearrow e_\ell$ (resp. $e_\ell \nearrow \cdots \nearrow e_2 \nearrow e_1$).

Any pair of d-inc paths both of length more than one that share only their d-lower and d-upper end vertices naturally defines a subgraph of G, which we call a *d-inc subgraph*, as follows. Let P and Q be any d-inc paths composing such a pair, where we assume without loss of generality that $P \searrow Q$ because either $P \searrow Q$ or $Q \searrow P$ holds due to the condition between P and Q. The d-inc subgraph specified by this pair of d-inc paths, which we denote by $G_{P,Q}$, consists of the union of all grid units g such that $P \searrow g \searrow Q$ (see Fig. 2(a)). We call P (resp. Q) the *anti-diagonally lower* (*a-lower*) (resp. *anti-diagonally upper* (*a-upper*)) *boundary path* of $G_{P,Q}$. The *boundary path length* of a d-inc subgraph is the common length of the a-lower and a-upper boundary paths of the d-inc subgraph. We say that a d-inc path R *passes across* a d-inc subgraph $G_{P,Q}$, if R *partitions* $G_{P,Q}$ into two d-inc subgraphs in the sense that the union of the two d-inc subgraphs composes $G_{P,Q}$ while the intersection of the two d-inc subgraphs composes R (again see Fig. 2(a)).

2.1. Semi-local LCS length technique of Tiskin [22]

A straightforward generalization [19] of the semi-local LCS length technique of Tiskin [22] provides an approach to managing recurrence relations of values $l(u, v)$ for all pairs of vertices u and v in the boundary paths of a d-inc subgraph

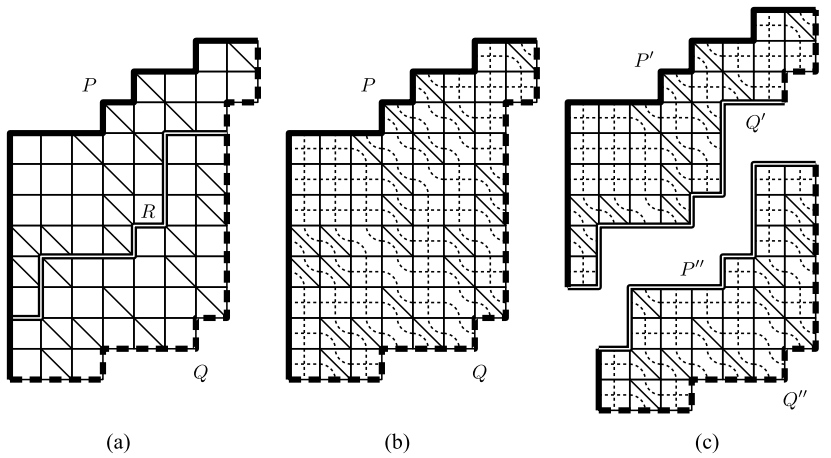

Fig. 2. (a) A d-inc subgraph $G_{P,Q}$ of the same G as in Fig. 1(b) with ${}_*P = {}_*Q = (11, 0)$ and $P^* = Q^* = (0, 8)$, together with a d-inc path R passing across $G_{P,Q}$, where P, Q, and R are indicated by the solid, dashed, and doubled polygonal lines, respectively; (b) the d-inc bijection $\beta_{P,Q}$ for $G_{P,Q}$, where each dotted curve connects an edge p in P with the edge $\beta_{P,Q}(p)$ in Q; (c) the d-inc subgraphs $G_{P',Q'}$ and $G_{P'',Q''}$ such that the intersection of Q' and P'' is R into which R partitions $G_{P,Q}$, with the d-inc bijections $\beta_{P',Q'}$ and $\beta_{P'',Q''}$ drawn in the same manner as (b).

of G. For technical reasons, for any vertices w and w' in G, let $l(w, w') = j_{w'} - j_w$, if $w \nearrow w'$, and let $l(w, w') = i_{w'} - i_w$, if $w' \nearrow w$.

The following lemma claims that for any d-inc subgraph $G_{P,Q}$, a bijection mapping any edge in P to an edge in Q represents such recurrence relations (see also Fig. 2(b)).

Lemma 1. *For any d-inc subgraph $G_{P,Q}$, there exists a bijection β from the set of all edges in P to the set of all edges in Q such that, for any edge p in P and any edge q in Q,*

$$l({}_*p, q^*) - l(p^*, q^*) = l({}_*p, {}_*q) - l(p^*, {}_*q) + \begin{cases} 1 & \text{if } q = \beta(p); \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, this β satisfies that $p \searrow \beta(p)$ for any edge p in P.

Proof. For any edge p in P and any edge q in Q, let $\delta_{p,q}$ denote the integer such that

$$l({}_*p, q^*) - l(p^*, q^*) = l({}_*p, {}_*q) - l(p^*, {}_*q) + \delta_{(p,q)}. \tag{1}$$

It follows from definition of $l(w, w')$ with $w \nearrow w'$ or $w' \nearrow w$ that if either $q \nearrow p$ or $p \nearrow q$, then $\delta_{p,q} = 0$.

Let p be an arbitrary edge in P and let R be the d-inc path consisting of all edges q in Q such that $p \searrow q$ (i.e., neither $q \nearrow p$ nor $p \searrow q$). Since ${}_*R \nearrow p \nearrow R^*$, it follows from definition of $l(w, w')$ with $w \nearrow w'$ or $w' \nearrow w$ that

$$l({}_*p, R^*) - l(p^*, R^*) = l({}_*p, {}_*R) - l(p^*, {}_*R) + 1 \tag{2}$$

holds regardless of whether p is vertical or horizontal. On the other hand, by summing the left and right sides of Equality (1), respectively, over all edges q in R and removing terms that can cancel each other out, we obtain

$$l({}_*p, R^*) - l(p^*, R^*) = l({}_*p, {}_*R) - l(p^*, {}_*R) + \textstyle\sum_q \delta_{p,q}. \tag{3}$$

This can be verified because $l({}_*p, v) - l(p^*, v)$ for any vertex v of R other than the end vertices ${}_*R$ and R^* appears in both sides and hence is canceled out. Equalities (2) and (3) imply that $\sum_q \delta_{p,q} = 1$. Thus, if integers $\delta_{p,q}$ for all edges q in R are non-negative, then there exists an edge q_p in R such that $\delta_{p,q_p} = 1$ and $\delta_{p,q} = 0$ for any edge q in Q other than q_p.

For any edge q in R, any shortest path between ${}_*p$ and q^* and any shortest path between p^* and ${}_*q$ cross at some vertex w. Since $l({}_*p, w) + l(w, q^*) = l({}_*p, q^*)$, $l(p^*, w) + l(w, {}_*q) = l(p^*, {}_*q)$, $l({}_*p, {}_*q) \le l({}_*p, w) + l(w, {}_*q)$, and $l(p^*, q^*) \le l(p^*, w) + l(w, q^*)$ hold, we have that $l({}_*p, {}_*q) + l(p^*, q^*) \le l({}_*p, q^*) + l(p^*, {}_*q)$, and hence $l({}_*p, q^*) - l(p^*, q^*) \ge l({}_*p, q_*) - l(p^*, q_*)$. This implies that $\delta_{p,q}$ is non-negative.

From the above, for any edge p in P, there exists an edge q_p in Q such that $p \searrow q_p$, $\delta_{p,q_p} = 1$, and $\delta_{p,q} = 0$ for any edge q in Q other than q_p. By a symmetric argument, we can also show that for any edge q in Q, there exists an edge p_q in P such that $p_q \searrow q$, $\delta_{p_q,q} = 1$, and $\delta_{p,q} = 0$ for any edge p in P other than p_q. It is easy to verify that, for any pair of an edge p in P and an edge q in Q, $q = q_p$ if and only if $p = p_q$. Therefore, the bijection that maps any edge p in P to edge q_p in Q satisfies the condition of β in the lemma. □

For any d-inc subgraph $G_{P,Q}$, let $\beta_{P,Q}$ denote the bijection β in Lemma 1. Since P and Q share no edges, for convenience, we also use $\beta_{P,Q}$ to denote the inverse bijection of $\beta_{P,Q}$, so that $\beta_{P,Q}(p) = q$ if and only if $\beta_{P,Q}(q) = p$. We call $\beta_{P,Q}$ the d-*inc bijection* for $G_{P,Q}$. Lemma 1 reveals that, as long as the d-inc bijection for $G_{P,Q}$ is available, direct access to $G_{P,Q}$ is no longer necessary for determining $l(u, v)$ for any vertex u in P and any vertex v in Q. That is, as claimed in the following corollary, we can determine $l(u, v)$ by counting edges p in P such that both p and edge $\beta_{P,Q}(p)$ satisfy certain conditions with respect to u and v, respectively.

Corollary 1. *For any d-inc subgraph $G_{P,Q}$, any vertex u in P, and any vertex v in Q, $l(u, v)$ is equal to $i_v - i_u$ plus the number of edges p in P such that $u \nearrow p$ and $\beta_{P,Q}(p) \nearrow v$.*

Proof. Let $\tilde{l}$ be the number of edges p in P such that $u \nearrow p$ and $\beta_{P,Q}(p) \nearrow v$. By summing the left and right sides of the equality in Lemma 1, respectively, over all pairs of an edge p in P with $u \nearrow p$ and an edge q in Q with $q \nearrow v$ and removing terms that can cancel each other out, we obtain

$$l(u, v) - l(P^*, v) = l(u, {}_*Q) - l(P^*, {}_*Q) + \tilde{l}.$$

Furthermore, it follows from $v \nearrow P^*$, ${}_*Q \nearrow u$, and ${}_*Q \nearrow P^*$ that $l(P^*, v) = i_v - i_{P^*}$, $l(u, {}_*Q) = i_{{}_*Q} - i_u$, and $l(P^*, {}_*Q) = i_{{}_*Q} - i_{P^*}$, respectively. □

The following lemma provides how to recursively construct the d-inc bijection for any d-inc subgraph of G.

Lemma 2. *Let $G_{P,Q}$ be an arbitrary d-inc subgraph of G and let R be an arbitrary d-inc path passing across $G_{P,Q}$. Let $G_{P',Q'}$ and $G_{P'',Q''}$ be the d-inc subgraphs of G into which R partitions $G_{P,Q}$, where Q' and P'' share R as their intersection (see Fig. 2(c)). Let any edge p in P be called trivial, if either p is an edge in P'' or $\beta_{P',Q'}(p)$ is an edge in Q, and called involved, otherwise. For any trivial edge p in P, if p is an edge in P'', then $\beta_{P,Q}(p) = \beta_{P'',Q''}(p)$; otherwise, $\beta_{P,Q}(p) = \beta_{P',Q'}(p)$. If edges $\beta_{P',Q'}(r)$ and $\beta_{P'',Q''}(r)$ for all edges r in R are available, then edges $\beta_{P,Q}(p)$ for all involved edges p in P can be determined in $O(\ell \log \ell)$ time and $O(\ell)$ space, where ℓ is the length of R.*

Proof. Since $\beta_{P,Q}(p)$ for any trivial edge p in P is obvious, we focus only on how to determine edges $\beta_{P,Q}(p)$ for all involved edges p in P in $O(\ell \log \ell)$ time and $O(\ell)$ space using edges $\beta_{P',Q'}(r)$ and $\beta_{P'',Q''}(r)$ for all edges r in R.

Let D' (resp. D'') be the matrix consisting of elements $D'[u, w]$ (resp. $D''[w, v]$) for all pairs of a vertex u in P' (resp. w in R) and a vertex w in R (resp. v in Q''), where $D'[u, w]$ (resp. $D''[w, v]$) is the number of edges r in R such that $u \nearrow \beta_{P',Q'}(r)$ and $r \nearrow w$ (resp. $w \nearrow r$ and $\beta_{P'',Q''}(r) \nearrow v$). Let D be the min-sum product of matrices D' and D'', which consists of elements $D[u, v]$ for all pairs of a vertex u in P' and a vertex v in Q'', where $D[u, v]$ is the minimum of $D'[u, w] + D''[w, v]$ over all vertices w in R. It suffices to show that $D[u, v]$ is equal to the number of involved edges p in P such that $u \nearrow p$ and $\beta_{P,Q}(p) \nearrow v$. This is because it follows from [23,18] that, given edges $\beta_{P',Q'}(r)$ and edges $\beta_{P'',Q''}(r)$ for all edges r in R as a representation of matrices D' and D'' as input, their min-sum product D (represented in the same manner as D' and D'') can be obtained in $O(\ell \log \ell)$ time and $O(\ell)$ space.

Let u be an arbitrary vertex in P' and let v be an arbitrary vertex in Q''. Since any shortest path between u and v shares at least one vertex with R, $l(u, v)$ is equal to the minimum of $l(u, w) + l(w, v)$ over all vertices w in R. For any vertex w in R, it follows from Corollary 1 that

$$l(u, w) + l(w, v) = (i_v - i_u) + D'[u, w] + D''[w, v] + c(u, v),$$

where $c(u, v)$ is the sum of the number of edges p in P'' such that $u \nearrow p$ and $\beta_{P'',Q''}(p) \nearrow v$ and the number of edges q in Q' such that $u \nearrow \beta_{P',Q'}(q)$ and $q \nearrow v$. Therefore, we have that $D[u, v] + c(u, v) = l(u, v) - (i_v - i_u)$, which is equal to the number of edges p in P with $u \nearrow p$ and $\beta_{P,Q}(p) \nearrow v$ due to Corollary 1. On the other hand, $c(u, v)$ is equal to the number of trivial edges p in P such that $u \nearrow p$ and $\beta_{P,Q}(p) \nearrow v$. Thus, $D[u, v]$ represents the number of involved edges p in P such that $u \nearrow p$ and $\beta_{P,Q}(p) \nearrow v$. □

The following corollary of Lemma 2 is useful to deal with the d-inc bijection without explicitly determining it.

Corollary 2. *Let $G_{P,Q}$ and $G_{P',Q'}$ be arbitrary d-inc subgraphs of G. For any edge p shared by P and P' and any vertex v shared by Q and Q', $\beta_{P,Q}(p) \nearrow v$ if and only if $\beta_{P',Q'}(p) \nearrow v$. Similarly, for any vertex u shared by P and P' and any edge q shared by Q and Q', $u \nearrow \beta_{P,Q}(q)$ if and only if $u \nearrow \beta_{P',Q'}(q)$.*

Proof. By symmetry, we show only the first half of the corollary. Let p be an arbitrary edge shared by P and P' and let v be an arbitrary vertex shared by Q and Q'. If $p \nearrow v$, then both $\beta_{P,Q}(p) \nearrow v$ and $\beta_{P',Q'}(p) \nearrow v$ hold due to Lemma 1. Similarly, if $v \nearrow p$, then both $v \nearrow \beta_{P,Q}(p)$ and $v \nearrow \beta_{P',Q'}(p)$ hold.

Suppose that $p \searrow v$, and let G_{P_0,Q_0} be the d-inc subgraph consisting of all grid units g of G such that $p \searrow g \searrow v$. Consider an arbitrary sequence of distinct grid units $g_1, g_2, \ldots, g_\ell$ of G that are not ones in G_{P_0,P_0} such that, for any index

k with $1 \le k \le \ell$, the union of G_{P_0,Q_0} and $g_1, g_2, \ldots, g_k$ composes a d-inc subgraph, which we denote by G_{P_k,Q_k}, p is an edge in P_k, and v is a vertex in Q_k. There exists such a sequence with $G_{P_\ell,Q_\ell} = G_{P,Q}$, and analogously with respect to $G_{P',Q'}$. Since G_{P_k,Q_k} is partitioned into d-inc subgraphs $G_{P_{k-1},Q_{k-1}}$ and g_k, it follows from Lemma 2 that $\beta_{P_{k-1},Q_{k-1}}(p) \nearrow v$ if and only if $\beta_{P_k,Q_k}(p) \nearrow v$. This implies by induction that $\beta_{P_0,Q_0}(p) \nearrow v$ if and only if $\beta_{P_\ell,Q_\ell}(p) \nearrow v$, which further implies that $\beta_{P,Q}(p) \nearrow v$ if and only if $\beta_{P',Q'}(p) \nearrow v$. □

3. Basic data structure supporting linear-time queries

We start with designing a basic data structure that is $O(mn)$-time constructible (hence is of $O(mn)$ size) and supports $O(a_v - a_u)$-time queries of $l(u, v)$ for any pair of vertices u and v in G with $u \searrow v$, where we recall that $a_v - a_u = \Theta(l(u, v))$. This data structure will be modified in Section 4 to support faster queries. In what follows, we assume without loss of generality that $m \ge n$.

To define the basic data structure, we introduce the d-inc bijections for the following d-inc subgraphs of G each in the shape of an anti-diagonal strip.

Definition 1. For any even integer a'' with $0 < a'' < m + n$, let $R_{a''}$ denote the d-inc path passing across G such that $a_w = a''$ for any vertex w in the path. Let $R_{a_{(0,0)}}$ (resp. $R_{a_{(m,n)}}$) denote the d-inc path consisting only of a single vertex $(0, 0)$ (resp. (m, n)). For any even integers a and a' with $0 \le a < a' \le m + n$, the anti-diagonal strip (a-strip) between a and a' of width $a' - a$, which we denote $G_{a..a'}$, is defined as the d-inc subgraph of G consisting of all grid units g such that $R_a \searrow g \searrow R_{a'}$. Let $P_{a..a'}$ and $Q_{a..a'}$ denote the a-lower and a-upper boundary paths of $G_{a..a'}$, respectively, and let $\beta_{a..a'}$ denote the d-inc bijection for $G_{a..a'}$. As an implementation of $\beta_{a..a'}$, we adopt the array of size $O(n)$ consisting of edges $\beta_{a..a'}(p)$ for all edges p in $P_{a..a'}$ with $(0, 0) \searrow p$ or $(0, 0) \nearrow p$ and edges $\beta_{a..a'}(q)$ for all edges q in $Q_{a..a'}$ with $q \nearrow (m, n)$ or $q \searrow (m, n)$.

The reason for defining $G_{a..a'}$ only for even integers a and a' is that if a and a' are too close together, such as $a' = a + 1$, then $G_{a..a'}$ cannot be defined as a d-inc subgraph. Moreover, this restriction on a and a' does not affect our data structure design.

The basic data structure, which we denote by $\mathcal{B}$, consists of the d-inc bijections for all a-strips between a and a' such that the width $a' - a$ is a power of two and both a and a' are multiples of this power. We call any such a-strip *basic*. Since there are $O(m/2^k)$ basic a-strips of width 2^k for each positive integer k, and the d-inc bijection for each basic a-strip is an array of $O(n)$ edges, $\mathcal{B}$ can be stored in $O(mn)$ space. Later we will show that $\mathcal{B}$ is $O(mn)$-time constructible.

Suppose that $\mathcal{B}$ is available. To calculate $l(u, v)$, we use the d-inc bijections for all basic a-strips $G_{a..a'}$ that are maximal in the sense that $G_{a..a'}$ is the only basic a-strip $G_{b..b'}$ that satisfies both $a_u \le b \le a$ and $a' \le b' \le a_v$. We call any such a-strip (u, v)-*related*. Due to this definition, the width of any (u, v)-related a-strip is at most $2^{\lfloor \log_2(a_v - a_u) \rfloor}$. Furthermore, any basic a-strip $G_{a..a'}$ of width 2^k such that $a_u + 2^k \le a$ and $a' \le a_v - 2^k$ is not (u, v)-related because either $G_{a-2^k..a'}$ or $G_{a..a'+2^k}$ is a basic a-strip. Hence, for any integer k with $1 \le k \le \lfloor \log_2(a_v - a_u) \rfloor$, there exist at most two (u, v)-related a-strips of width 2^k. The reason for considering (u, v)-related a-strips is that $G_{a_u..a_v}$ is partitioned into these $O(\log(a_v - a_u))$ (u, v)-related a-strips (see Fig. 3(a)).

A naive algorithm determines $l(u, v)$ in $O(n \log n \log(a_v - a_u))$ time as follows. Using the d-inc bijections for all (u, v)-related a-strips, which are available in $\mathcal{B}$, the algorithm constructs the d-inc bijection for $G_{a_u..a_v}$ in $O((\log(a_v - a_u))n \log n)$ time based on Lemma 2. After doing this, it counts the number of edges p in $P_{a_u..a_v}$ such that $u \nearrow p$ and $\beta_{a_u..a_v}(p) \nearrow v$ in $O(n)$ time to apply Corollary 1.

To improve execution time of the above naive algorithm to $O(a_v - a_u)$, our idea is to consider a subgraph $G^{\square}_{a..a'}$ of each a-strip $G_{a..a'}$ with $a_u \le a$ and $a' \le a_v$, which is defined by focusing only on grid units between u and v as follows. For any pair of even integers a and a' with $a_u \le a < a' \le a_v$, let $G^{\square}_{a..a'}$ denote the d-inc subgraph consisting of the union of all grid units g in $G_{a..a'}$ such that $u \searrow g \searrow v$ (see Fig. 3(b)). The boundary path length of $G^{\square}_{a..a'}$ for any (u, v)-related $G_{a..a'}$ is hence only $O(a' - a)$. Utilizing this property, we apply Corollary 1 somehow to determine $l(u, v)$.

Since no confusion arises, we use superscript $\square$ in notation $G^{\square}_{a..a'}$ instead of specifying (u, v) for simplicity. Adopting this style, we also introduce the following notations. Let $\beta^{\square}_{a..a'}$ denote the d-inc bijection for $G^{\square}_{a..a'}$, and let $P^{\square}_{a..a'}$ and $Q^{\square}_{a..a'}$ denote the a-lower and a-upper boundary paths of $G^{\square}_{a..a'}$, respectively. For any even integer a'' with $a_u < a'' < a_v$, let $R^{\square}_{a''}$ denote the d-inc path passing across $G^{\square}_{a_u..a_v}$ such that $a_w = a''$ for any vertex w in the path (i.e., the path consisting of all edges r in $R_{a''}$ with $u \searrow r \searrow v$). Let $R^{\square}_{a_u}$ and $R^{\square}_{a_v}$ denote the d-inc paths u and v of length zero, respectively. Let $u^{\square}_{a''}$ and $v^{\square}_{a''}$ denote the d-upper and d-lower end vertices of $R^{\square}_{a''}$, respectively. Hence, for any a-strip $G_{a..a'}$ with $a_u \le a < a' \le a_v$, $P^{\square}_{a..a'}$ and $P_{a..a'}$ (resp. $Q^{\square}_{a..a'}$ and $Q_{a..a'}$) share all edges in $R^{\square}_a$ (resp. $R^{\square}_{a'}$). Furthermore, for any edge p in any of $P^{\square}_{a..a'}$ or $P_{a..a'}$, $u \nearrow p$ if and only if $u^{\square}_a \nearrow p$. Similarly, for any edge q in any of $Q^{\square}_{a..a'}$ or $Q_{a..a'}$, $q \nearrow v$ if and only if $q \nearrow v^{\square}_{a'}$.

For any even integer a with $a_u \le a < a_v$, let Π_a denote the set of all edges p in $R^{\square}_a$ with $\beta^{\square}_{a..a_v}(p) \nearrow v$, and let π_a denote the number of all edges p in $P^{\square}_{a..a_v}$ such that $u \nearrow p$ and $\beta^{\square}_{a..a_v}(p) \nearrow v$. Since $l(u, v)$ can be calculated as $i_v - i_u + \pi_{a_u}$ due to Corollary 1, if $\Pi_{a'}$ and $\pi_{a'}$ can be updated to Π_a and π_a in $O(a' - a)$ time for any (u, v)-related a-strip $G_{a..a'}$, then $l(u, v)$ can be determined in $O(a_v - a_u)$ time. Data structure $\mathcal{B}$ allows us to take this approach as shown below.

For any (u, v)-related a-strip $G_{a..a'}$, let Π'_a (resp. Π''_a) denote the set of all edges p in $R^{\square}_a$ such that $\beta^{\square}_{a..a_v}(p) \nearrow v^{\square}_{a'}$ (resp. $v^{\square}_{a'} \nearrow \beta^{\square}_{a..a_v}(p) \nearrow v$), so that Π_a is partitioned into Π'_a and Π''_a. Similarly, let π'_a (resp. π''_a) denote the number of all edges

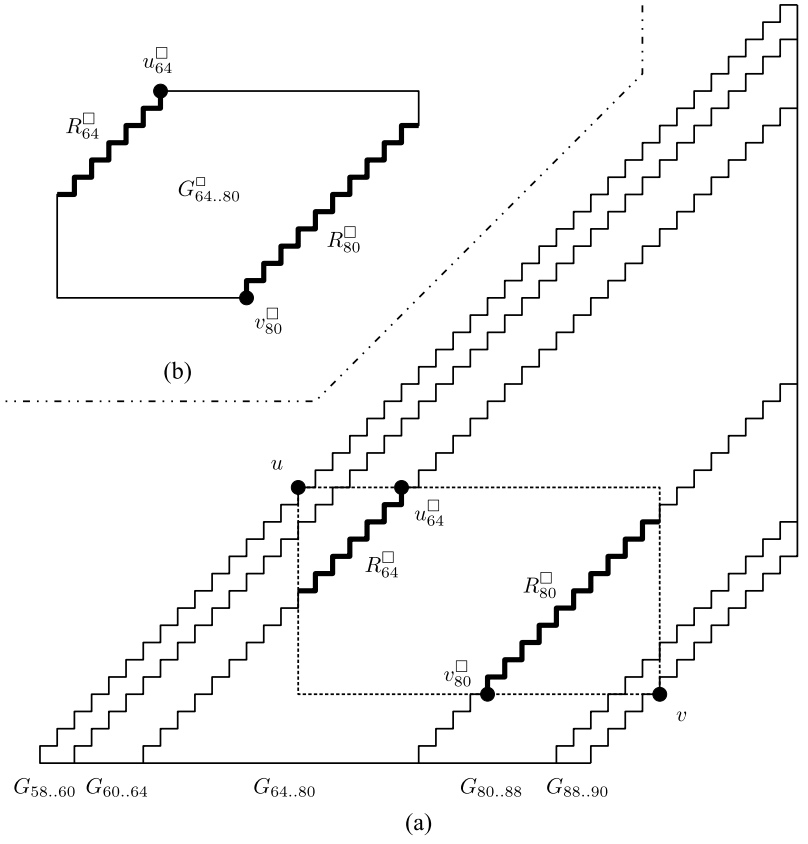

Fig. 3. (a) The (u, v)-related a-strips, composing $G_{a_u..a_v}$, for $u = (39, 19)$ and $v = (51, 40)$, where $m = 55$, $n = 48$, and the dotted box indicates $G^{\square}_{a_u..a_v}$; (b) $G^{\square}_{64..80}$ for the same u and v as (a).

p in $P^{\square}_{a..a_v}$ with $u^{\square}_a \nearrow p$ such that $\beta^{\square}_{a..a_v}(p) \nearrow v^{\square}_{a'}$ (resp. $v^{\square}_{a'} \nearrow \beta^{\square}_{a..a_v}(p) \nearrow v$), so that π_a is decomposed into the sum of π'_a and π''_a. We determine all of Π'_a, Π''_a, π'_a, and π''_a from $\Pi_{a'}$ and $\pi_{a'}$ previously determined for each (u, v)-related a-strip $G_{a..a'}$ in the descending order of a.

The following facts claim that Π'_a and π'_a can be determined only from the d-inc bijection for $G_{a..a'}$.

Fact 1. *For any pair of even integers a and a' with $a_u \le a < a' \le a_v$, Π'_a consists of all edges p in $R^{\square}_a$ with $\beta_{a..a'}(p) \nearrow v^{\square}_{a'}$.*

Proof. Any edge in $R^{\square}_a$ is shared by $P^{\square}_{a..a_v}$ and $P_{a..a'}$ and $v^{\square}_{a'}$ is shared by $Q^{\square}_{a..a_v}$ and $Q_{a..a'}$ (see Fig. 4 (a)). Thus, the fact follows from Corollary 2. □

Fact 2. *For any pair of even integers a and a' with $a_u \le a < a' \le a_v$, π'_a is equal to the number of edges p in $P_{a..a'}$ such that $u^{\square}_a \nearrow p$ and $\beta_{a..a'}(p) \nearrow v^{\square}_{a'}$.*

Proof. Let P^- (resp. $P^/$) be the d-inc path consisting of all edges p in $P^{\square}_{a..a_v}$ (resp. $P_{a..a'}$) with $u \nearrow p \searrow v^{\square}_{a'}$, and let Q^- (resp. $Q^/$) be the d-inc path consisting of all edges q in $Q^{\square}_{a..a_v}$ (resp. $Q_{a..a'}$) with $u \searrow q \nearrow v^{\square}_{a'}$ (see Fig. 4 (b)). Since $p \searrow \beta^{\square}_{a..a_v}(p)$ (resp. $p \searrow \beta_{a..a'}(p)$) for any edge p in $P^{\square}_{a..a_v}$ (resp. $P_{a..a'}$) due to Lemma 1, it suffices to show that the number of edges p in P^- such that $\beta^{\square}_{a..a_v}(p)$ is an edge in Q^- is equal to the number of edge p in $P^/$ such that $\beta_{a..a'}(p)$ is an edge in $Q^/$. From Corollary 2, both the above numbers of edges are equal to the number of edges p in P^- such that $\beta_{P,Q}(p)$ is an edge in $Q^/$, where $G_{P,Q}$ is an arbitrary d-inc subgraph of G such that any edge in P^- is an edge in P and any edge in $Q^/$ is an edge in Q. □

Unlike in the case of determining Π'_a and π'_a, we use $\Pi_{a'}$ as well as the d-inc bijection for $G_{a..a'}$ to determine Π''_a and π''_a. This is the reason why we determine not only π_a but also Π_a. From Corollary 2 and Lemma 1, for any edge p in $P^{\square}_{a..a'}$, if $v^{\square}_{a'} \nearrow \beta^{\square}_{a..a_v}(p) \nearrow v$, then $\beta^{\square}_{a..a'}(p)$ is an edge in $R^{\square}_{a'}$. Furthermore, $\pi_{a'}$ is equal to the number of edges p in $P^{\square}_{a..a_v}$ with $\beta^{\square}_{a..a_v}(p) \nearrow v$ that is not an edge in $P^{\square}_{a..a'}$. Hence, to determine Π''_a and π''_a, we concentrate on edges p in $P^{\square}_{a..a'}$ such that $\beta^{\square}_{a..a'}(p)$ is an edge in $R^{\square}_{a'}$.

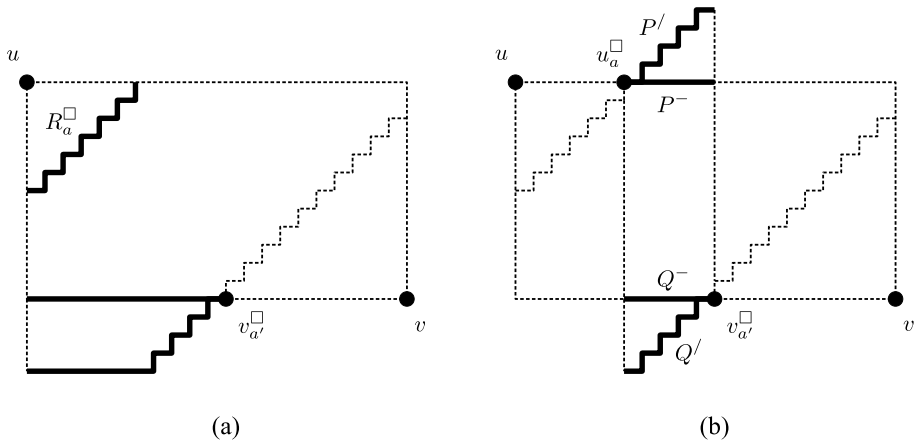

Fig. 4. (a) Path $R^{\square}_{a}$, edges q in $Q^{\square}_{a'}$ with $q \nearrow v^{\square}_{a'}$, and edges q in $Q_{a..a'}$ with $q \nearrow v^{\square}_{a'}$; (b) paths P^{-}, $P^{/}$, Q^{-}, and $Q^{/}$, both for the same G, u, and v as Fig. 3, where $G_{a..a'} = G_{64..80}$.

For any pair of vertices y and z in G, let $\tilde{l}(y,z)$ denote $l(y,z) - (i_z - i_y)$. It follows from Corollary 1 that, for any edge p in $P^{\square}_{a..a'}$,

$$\tilde{l}(*p, v) = \tilde{l}(p^*, v) + \begin{cases} 1 & \text{if } \beta^{\square}_{a..a_v}(p) \nearrow v; \\ 0 & \text{otherwise.} \end{cases}$$

We determine Π''_a and π''_a based on this observation, using the following array somehow. For any vertex w in $P^{\square}_{a..a'}$, let L_w denote the array of $L_w[x] = \tilde{l}(w,x) + \tilde{l}(x,v)$ over all vertices x in $R^{\square}_{a'}$, so that $\tilde{l}(w,v)$ is represented as the minimum element of L_w. It follows from Corollary 1 that, for any edge p in $P^{\square}_{a..a'}$ and any vertex x in $R^{\square}_{a'}$,

$$\tilde{l}(*p, x) = \tilde{l}(p^*, x) + \begin{cases} 1 & \text{if } \beta^{\square}_{a..a'}(p) \nearrow x; \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, L_{p^*} can be updated to L_{*p} by increasing $L_{p^*}[x]$ by one for each vertex x such that $\beta^{\square}_{a..a'}(p) \nearrow x$.

A standard technique [14,18,19] allows us to maintain L_w by focusing only on edges r in $R^{\square}_{a'}$ such that $L_w[x] > L_w[r^*]$ for all vertices x with $x \nearrow r$. Let $\hat{L}_w$ denote the list of all such edges r in the diagonally ascending order. Since the difference $L_w[*r] - L_w[r^*]$ for any edge r in $R^{\square}_{a'}$ is one of -1, 0, or 1, if r is the hth edge in $\hat{L}_w$, then $L_w[r^*] = L_w[v^{\square}_{a'}] - h$. This implies that $\tilde{l}(w,v)$ is equal to $L_w[v^{\square}_{a'}]$ minus the number of edges in $\hat{L}_w$. Furthermore, for any edge p in $P^{\square}_{a..a'}$, $\hat{L}_{p^*}$ can be updated to $\hat{L}_{*p}$ only by deleting the first edge r such that $\beta^{\square}_{a..a'}(p) \nearrow r$, if $\beta^{\square}_{a..a'}(p)$ is an edge in $R^{\square}_{a'}$ and such r exists, or remaining unchanged, otherwise. Therefore, for any edge p in $P^{\square}_{a..a'}$ such that $\beta^{\square}_{a..a'}(p)$ is an edge in $R^{\square}_{a'}$, $\hat{L}_{p^*}$ has an edge r such that $\beta^{\square}_{a..a'}(p) \nearrow r$ if and only if $\tilde{l}(*p, v) = \tilde{l}(p^*, v) + 1$, which holds if and only if $\beta^{\square}_{a..a_v}(p) \nearrow v$ as mentioned earlier.

To implement $\hat{L}_w$ so as to allow us to efficiently update $\hat{L}_{p^*}$ to $\hat{L}_{*p}$ using $\beta^{\square}_{a..a'}(p)$, we can utilize the algorithm for the static tree set union problem [9]. For any vertex w in $P^{\square}_{a..a'}$, if we know which edges compose $\hat{L}_w$, then this algorithm initializes data structure $\hat{L}$ to $\hat{L}_w$ in time linear in the number of edges in $R^{\square}_{a'}$. Furthermore, for each edge q in any sequence of distinct edges in $R^{\square}_{a'}$, this algorithm deletes the first edge r with $q \nearrow r$ from $\hat{L}$, if any, or does nothing, otherwise, in amortized $O(1)$ time.

Adopting the above implementation $\hat{L}$, we initially construct $\hat{L}_w$ for the d-upper end vertex w of $P^{\square}_{a..a'}$, update it to $\hat{L}_{u^{\square}_a}$ to determine π''_a, which is equal to the sum of $\pi_{a'}$ and the number of elements deleted from $\hat{L}$ during this update, and use $\hat{L}_{u^{\square}_a}$ to obtain Π''_a based on the following facts.

Fact 3. *For any pair of even integers a and a' with $a_u \le a < a' \le a_v$, $\hat{L}_w$ for the d-upper end vertex w of $P^{\square}_{a..a'}$ consists of all edges in $\Pi_{a'}$.*

Proof. For any vertex x in $R^{\square}_{a'}$, $\tilde{l}(w,x) = 0$ due to $x \nearrow w$, and $\tilde{l}(x,v)$ is equal to the number of edges r in $R^{\square}_{a'}$ with $x \nearrow r$ and $\beta^{\square}_{a'..a_v}(r) \nearrow v$ due to Corollary 1. Thus, the fact holds because $\Pi_{a'}$ consists of all edges r in $R^{\square}_{a'}$ such that $\beta^{\square}_{a'..a_v}(r) \nearrow v$. □

Fact 4. *For any pair of even integers a and a' with $a_u \le a < a' \le a_v$, $\hat{L}_{u^{\square}_a}$ can be obtained from $\hat{L}_w$ for the d-upper end vertex w of $P^{\square}_{a..a'}$ by deleting the first element r with $q \nearrow r$, if any, for each edge q in $R^{\square}_{a'}$ with $u^{\square}_a \nearrow \beta_{a..a'}(q)$ in an arbitrary order.*

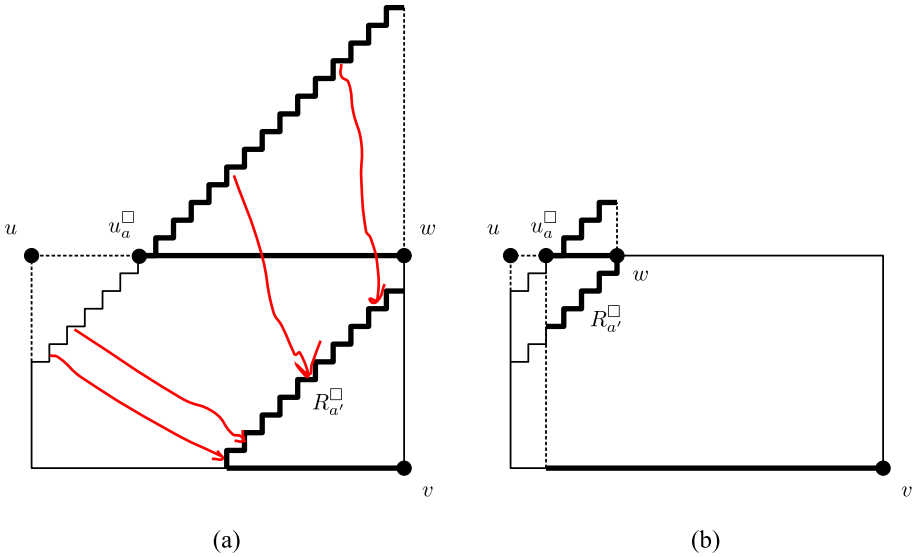

Fig. 5. (a) The union of $G^{\square}_{a..a'}$, $G^{\square}_{a'..a_v}$, and edges p in $P_{a..a'}$ such that $u_a^{\square} \nearrow p \searrow w$ for the same G, u, and v as Fig. 3, where $G_{a..a'} = G_{64..80}$; (b) the same as (a) with $G_{a..a'} = G_{60..64}$.

Proof. Since $u_a^{\square}$ is shared by $P_{a..a'}$ and $P^{\square}_{a..a'}$ and any edge in $R^{\square}_{a'}$ is shared by $Q_{a..a'}$ and $Q^{\square}_{a..a'}$, it follows from Corollary 2 that, for any edge q in $R^{\square}_{a'}$, $u_a^{\square} \nearrow \beta_{a..a'}(q)$ if and only if $u_a^{\square} \nearrow \beta^{\square}_{a..a'}(q)$ (see Fig. 5). Hence, it suffices to show the fact with $\beta_{a..a'}(q)$ replaced by $\beta^{\square}_{a..a'}(q)$.

Let $\hat{P}$ be the set of all edges p in $P^{\square}_{a..a'}$ with $u_a^{\square} \nearrow p$ such that $\beta^{\square}_{a..a'}(p)$ is an edge in $R^{\square}_{a'}$, and let $\hat{Q}$ be the set of edges $\beta^{\square}_{a..a'}(p)$ for all edges p in $\hat{P}$. From this definition, $\hat{L}_{u_a^{\square}}$ can be obtained from $\hat{L}_w$ by deleting the first edge r such that $\beta^{\square}_{a..a'}(p) \nearrow r$, if any, for each edge p in $\hat{P}$ in the diagonally descending order. Let $\hat{Q}$ consist of edges $q_1, q_2, \ldots, q_\ell$. For any permutation ϖ on indices $1, 2, \ldots, \ell$, let $\hat{L}^{\varpi}$ denote the list obtained from $\hat{L}_w$ by deleting the first edge r with $q \nearrow r$, if any, for each edge q in $\hat{Q}$ in the order of $q_{\varpi(1)}, q_{\varpi(2)}, \ldots, q_{\varpi(\ell)}$. Hence, $\hat{L}^{\varpi} = \hat{L}_{u_a^{\square}}$, if ϖ is the permutation such that $\beta^{\square}_{a..a'}(q_{\varpi(\ell)}) \nearrow \beta^{\square}_{a..a'}(q_{\varpi(\ell-1)}) \nearrow \cdots \nearrow \beta^{\square}_{a..a'}(q_{\varpi(1)})$. It is easy to verify that $\hat{L}^{\varpi} = \hat{L}^{\varpi'}$ for any permutation ϖ and any index h with $2 \le h \le \ell$, where ϖ' is the permutation obtained from ϖ by exchanging $\varpi(h-1)$ and $\varpi(h)$. Thus, we can prove by induction that, independent of ϖ, $\hat{L}_{u_a^{\square}} = \hat{L}^{\varpi}$ holds. □

Fact 5. *For any pair of even integers a and a' with $a_u \le a < a' \le a_v$, Π''_a can be obtained from $\hat{L}_{u_a^{\square}}$ by updating $\hat{L}_{p^*}$ to $\hat{L}_{*p}$ for each edge p in $R^{\square}_a$ in the diagonally descending order and collecting all edges p such that $\beta_{a..a'}(p)$ is an edge in $R^{\square}_{a'}$ and $\hat{L}_{p^*}$ has an edge r with $\beta_{a..a'}(p) \nearrow r$.*

Proof. The fact holds because for any edge p in $R^{\square}_a$, any of $\beta^{\square}_{a..a'}(p)$ or $\beta_{a..a'}(p)$ is an edge in $R^{\square}_{a'}$ if and only if $\beta^{\square}_{a..a'}(p) = \beta_{a..a'}(p)$. □

From the above facts, we have the following lemma.

Lemma 3. *Data structure $\mathcal{B}$ supports $O(a_v - a_u)$-time queries of $l(u, v)$ for any pair of vertices u and v in G with $u \searrow v$ using Algorithm* Basic(u, v) *presented in Fig. 6.*

Proof. It suffices to show that Algorithm Basic(u, v) outputs $l(u, v)$ in $O(a_v - a_u)$ time. After an appropriate initialization of Π_{a_v} and π_{a_v} by lines 1 and 2 of the algorithm, for each (u, v)-related a-strip $G_{a..a'}$ in the descending order of a, lines 4 through 18 determine Π_a and π_a in $O(a' - a)$ time using $\Pi_{a'}$, $\pi_{a'}$, and the d-inc bijection for $G_{a..a'}$ as follows. Recall that the boundary path length of $G^{\square}_{a..a'}$ is $O(a' - a)$ and the d-inc bijection for $G_{a..a'}$ is available in $\mathcal{B}$. Lines 4 and 5 initialize Π_a to the empty set and π_a to $\pi_{a'}$. Lines 6 and 7 add all edges in Π'_a to Π_a in $O(a' - a)$ time based on Fact 1. Lines 8 and 9 increase π_a by π'_a based on Fact 2. This is done in $O(a' - a)$ time because any edge p in $P_{a..a'}$ with $u_a^{\square} \nearrow p$ and $\beta_{a..a'}(p) \nearrow v^{\square}_{a'}$ satisfies that $p \nearrow v^{\square}_{a'}$ due to Lemma 1, and the number of such edges p is $O(a' - a)$. Line 10 initializes $\hat{L}$ to $\hat{L}_w$ for the d-upper end vertex w of $P^{\square}_{a..a'}$ in $O(a' - a)$ time based on Fact 3, where $\hat{L}$ is implemented by the algorithm for the static tree set union problem [9]. Lines 11 through 14 increase π_a by π''_a in $O(a' - a)$ time based on Fact 4. Line 15 through 18 add all edges in Π''_a to Π_a in $O(a' - a)$ time based on Fact 5. Thus, lines 4 though 18 determine Π_a and π_a

```
Let Π_{a_v} be the empty set;
let π_{a_v} = 0;
for each (u, v)-related a-strip G_{a..a'} in the descending order of a,
    let Π_a be the empty set;
    let π_a = π_{a'};
    for each edge p in R_a^□ with β_{a..a'}(p) ↗ v_{a'}^□,
        add p to Π_a;
    for each edge p in P_{a..a'} with u_a^□ ↗ p and β_{a..a'}(p) ↗ v_{a'}^□,
        increase π_a by one;
    let L̂ be the list of all edges r in Π_{a'} in the diagonally ascending order;
    for each edge q in R_{a'}^□ (in an arbitrary order),
        if u_a^□ ↗ β_{a..a'}(q) and L̂ has an edge r with q ↗ r, then
            delete the first such r from L̂;
            increase π_a by one;
    for each edge p in R_a^□ in the diagonally descending order,
        if β_{a..a'}(p) is an edge in R_{a'}^□ and L̂ has an edge r with β_{a..a'}(p) ↗ r, then
            delete the first such r from L̂;
            add p to Π_a;
output i_v − i_u + π_{a_u}.
```

Fig. 6. Algorithm Basic(u, v).

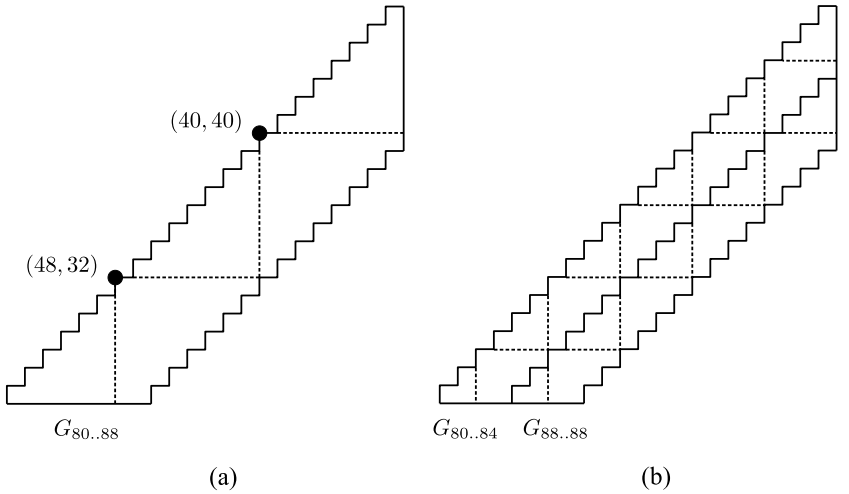

Fig. 7. (a) Basic a-strip $G_{80..88}$ of width 8 and (b) basic a-strips $G_{80..84}$ and $G_{84..88}$ of width 4, into which $G_{80..88}$ is partitioned, for the same G as Fig. 3, where dotted lines partition the a-strips into basic triangles of their width.

correctly in $O(a' - a)$ time. Since the sum of $a' - a$ over all (u, v)-related a-strips $G_{a..a'}$ is $O(a_v - a_u)$, Algorithm Basic(u, v) outputs $l(u, v)$ in $O(a_v - a_u)$ time. □

As mentioned before, our basic data structure is $O(mn)$-time constructible.

Lemma 4. *Data structure $\mathcal{B}$ can be constructed in $O(mn)$ time.*

Proof. To efficiently construct the d-inc bijections for all basic a-strips, we utilize the fact that any basic a-strip can be decomposed into d-inc subgraphs each in the shape of a half-square triangle as follows. Let $G_{a..a'}$ be an arbitrary basic a-strip of width 2^k. For any vertex w in $P_{a..a'}$ with $w \searrow Q_{a..a'}$ such that i_w (resp. j_w) is a non-zero multiple of 2^k, consider the d-inc path R passing across $G_{a..a'}$ such that $_*R = w$ (resp. $R^* = w$) and any edge in R is horizontal (resp. vertical). All of $O(n/2^k)$ such d-inc paths R partition $G_{a..a'}$ into $O(n/2^k)$ d-inc subgraphs, which we call *basic triangles* between a and a' of width 2^k (see Fig. 7). Let the d-inc bijection for any basic triangle $G_{P,Q}$ of width 2^k be implemented as the array of size $O(2^k)$ consisting of edges $\beta_{P,Q}(p)$ for all edges p in P and edges $\beta_{P,Q}(q)$ for all edges q in Q.

To construct $\mathcal{B}$, we use the d-inc bijections for all basic triangles. For any basic triangle of width 2, we can construct the d-inc bijection in $O(1)$ time from scratch. On the other hand, any basic triangle of width 2^k with $k \geq 2$ can be partitioned into at most four basic triangles of width 2^{k-1}. Hence, if the d-inc bijections for all of such basic triangles of width 2^{k-1} are available, then the d-inc bijection for the basic triangle of width 2^k can be obtained in $O(2^k k)$ time due to Lemma 2. There exist $O((m/2^k)(n/2^k))$ basic triangles of width 2^k for each positive integer k. Thus, by constructing the d-inc bijection for each basic triangle in ascending order of its width, the d-inc bijections for all basic triangles can be obtained in $O(mn)$ time, because $\sum_{k=1}^{\infty}(2^k k)(m/2^k)(n/2^k) = O(mn)$.

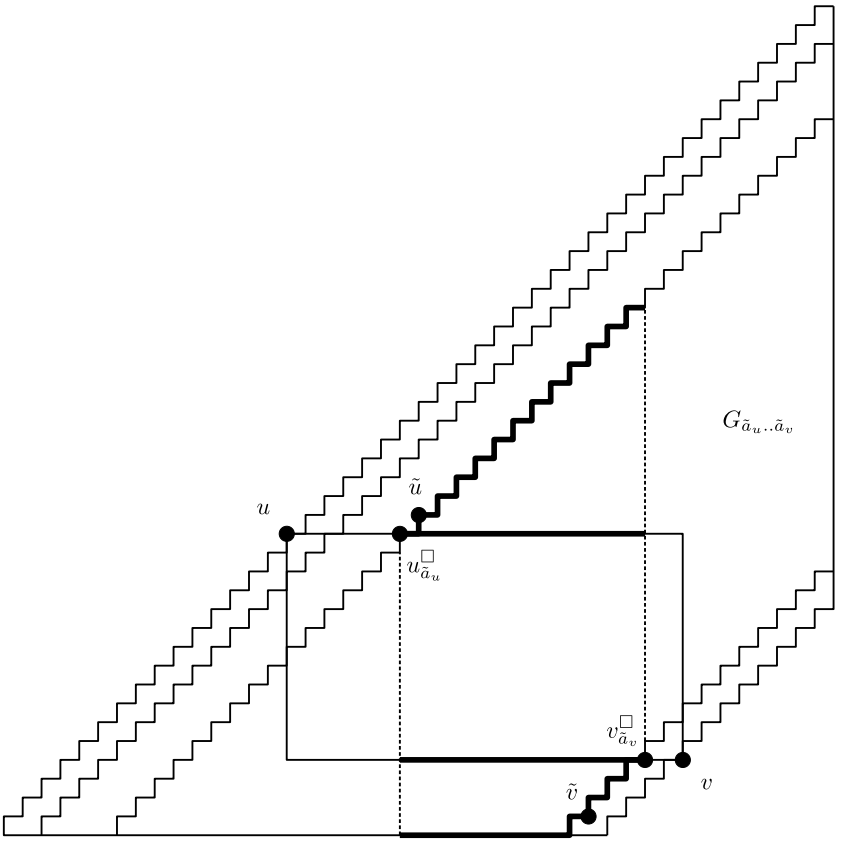

Fig. 8. The same $G_{a_u..a_v}$ as Fig. 3 with $\tilde{u} = (38, 26)$ and $\tilde{v} = (54, 35)$.

Recall that any basic a-strip of width 2^k can be partitioned into $O(n/2^k)$ basic triangles of width 2^k. Using the d-inc bijections for all such basic triangles, the d-inc bijection for the basic a-strip can be obtained in $O(kn)$ time due to Lemma 2. Since there are $O(m/2^k)$ basic a-strips of width 2^k for each positive integer k, the d-inc bijections for all basic a-strips can be obtained in $O(mn)$ time, because $\sum_{k=1}^{\infty}(kn)(m/2^k) = O(mn)$. □

4. Proposed data structure, supporting fast queries

Let ϵ be an arbitrary positive constant. This section modifies the basic data structure $\mathcal{B}$ presented in Section 3, without increasing its asymptotic construction time, so as to support $O(\sqrt{a_v - a_u}\log^{1+\epsilon}(a_v - a_u))$-time queries of $l(u, v)$ for any pair (u, v) of vertices in G with $u \searrow v$, where we recall again that $a_v - a_u = \Theta(l(u, v))$.

In what follows, for any positive integer k, let $2^{\langle k/2\rangle}$ denote the greatest power of two that is less than or equal to $2^{k/2}k^{1+\epsilon}$. Furthermore, for any even integers a and a' with $0 \le a < a' \le m + n$, let $k(a, a')$ denote the greatest integer k such that there exists a basic a-strip $G_{b..b'}$ of width 2^k with $a \le b$ and $b' \le a'$. Note that $2^{k(a_u,a_v)} = \Theta(a_v - a_u)$ and hence $2^{\langle k(a_u,a_v)/2\rangle} = \Theta(\sqrt{a_v - a_u}\log^{1+\epsilon}(a_v - a_u))$.

Our approach to improving the query time is to skip a dominant part of the process executed by Algorithm Basic(u, v). This is done by using an approximation $l(\tilde{u}, \tilde{v})$ of $l(u, v)$, together with the d-inc bijection for $G_{\tilde{a}_u..\tilde{a}_v}$ such that $\tilde{u}$ and $\tilde{v}$ are respectively vertices in the a-lower and a-upper boundary paths of $G_{\tilde{a}_u..\tilde{a}_v}$, where $(\tilde{u}, \tilde{v})$ is a certain pair of vertices in G such that both $l(u, \tilde{u})$ and $l(\tilde{v}, v)$ are $O(2^{\langle k(a_u,a_v)/2\rangle})$. We will hence design later the proposed data structure so as to contain $l(\tilde{u}, \tilde{v})$, if $\tilde{u} \searrow \tilde{v}$, as well as the d-inc bijection for $G_{\tilde{a}_u..\tilde{a}_v}$, for any pair of vertices u and v in G with $u \searrow v$.

As $\tilde{a}_u$ (resp. $\tilde{a}_v$), we adopt the least (resp. greatest) multiple of $2^{\langle k(a_u,a_v)/2\rangle}$ such that $a_u \le \tilde{a}_u$ (resp. $\tilde{a}_v \le a_v$), so that $G_{\tilde{a}_u..\tilde{a}_v}$ is the union of all (u, v)-related a-strips $G_{a..a'}$ of width greater than or equal to $2^{\langle k(a_u,a_v)/2\rangle}$ (see Fig. 8). Suppose that line 3 of Algorithm Basic(u, v) chooses $G_{\tilde{a}_u..\tilde{a}_v}$ as $G_{a..a'}$, instead of any (u, v)-related a-strip of width greater than or equal to $2^{\langle k(a_u,a_v)/2\rangle}$. Since both the lengths of $R^{\square}_{\tilde{a}_u}$ and $R^{\square}_{\tilde{a}_v}$ are $O(2^{\langle k(a_u,a_v)/2\rangle})$, in execution of lines 4 through 18 of Algorithm Basic(u, v) for $G_{\tilde{a}_u..\tilde{a}_v}$ as $G_{a..a'}$, lines 8 and 9 are executed in $O(\tilde{a}_v - \tilde{a}_u)$ time while all the other lines are executed in $O(2^{\langle k(a_u,a_v)/2\rangle})$ time. Thus, if execution time of lines 8 and 9 for $G_{\tilde{a}_u..\tilde{a}_v}$ is improved to $O(2^{\langle k(a_u,a_v)/2\rangle})$ somehow using $l(\tilde{u}, \tilde{v})$, Algorithm Basic(u, v) runs in $O(2^{\langle k(a_u,a_v)/2\rangle})$ time.

1: Do the same as lines 1 and 2 of Algorithm Basic(u, v);
2: for each $G_{a..a'}$ of $G_{\tilde{a}_u..\tilde{a}_v}$ and all (u, v)-related a-strips of width less than $2^{\langle k(a_u,a_v)/2\rangle}$ in the descending order of a,
3: do the same as lines 4 through 7 of Algorithm Basic(u, v);
4: if $a = \tilde{a}_u$ and $\tilde{u} \searrow \tilde{v}$, then
5: increase π_a by $l(\tilde{u}, \tilde{v}) - (i_{\tilde{v}} - i_{\tilde{u}})$;
6: for each edge p in $P_{a..a'}$ with $u_a^{\square} \nearrow p \nearrow \tilde{u}$ and $\beta_{a..a'}(p) \nearrow v_{a'}^{\square}$,
7: increase π_a by one;
8: for each edge q in $Q_{a..a'}$ with $\tilde{v} \nearrow q \nearrow v_{a'}^{\square}$ and $\tilde{u} \nearrow \beta_{a'..a}(q)$,
9: increase π_a by one,
10: otherwise,
11: do the same as lines 8 and 9 of Algorithm Basic(u, v);
12: do the same as lines 10 through 18 of Algorithm Basic(u, v);
13: output $i_v - i_u + \pi_{a_u}$.

Fig. 9. Algorithm Fast(u, v), where $\tilde{a}_u$ (resp. $\tilde{a}_v$) is the least (resp. greatest) multiple of $2^{\langle k(a_u,a_v)/2\rangle}$ such that $a_u \le \tilde{a}_u$ (resp. $\tilde{a}_v \le a_v$), and $\tilde{u}$ (resp. $\tilde{v}$) is the vertex in $P_{\tilde{a}_u..\tilde{a}_v}$ (resp. $Q_{\tilde{a}_u..\tilde{a}_v}$) such that $d_{\tilde{u}}$ (resp. $d_{\tilde{v}}$) is the least (resp. greatest) multiple of $2^{\langle k(a_u,a_v)/2\rangle}$ that is greater (resp. less) than or equal to d_u (resp. d_v), if any, or the d-upper (resp. d-lower) end vertex of $P_{\tilde{a}_u..\tilde{a}_v}$ (resp. $Q_{\tilde{a}_u..\tilde{a}_v}$), otherwise.

To achieve the above improvement, we adopt as $\tilde{u}$ (resp. $\tilde{v}$) the vertex in $P_{\tilde{a}_u..\tilde{a}_v}$ (resp. $Q_{\tilde{a}_u..\tilde{a}_v}$) such that $d_{\tilde{u}}$ (resp. $d_{\tilde{v}}$) is the least (resp. greatest) multiple of $2^{\langle k(a_u,a_v)/2\rangle}$ that is greater (resp. less) than or equal to d_u (resp. d_v), if any, or the d-upper (resp. d-lower) end vertex of $P_{\tilde{a}_u..\tilde{a}_v}$ (resp. $Q_{\tilde{a}_u..\tilde{a}_v}$), otherwise. Hence, the number of edges p (resp. q) in $P_{\tilde{a}_u..\tilde{a}_v}$ (resp. $Q_{\tilde{a}_u..\tilde{a}_v}$) such that $u \nearrow p \nearrow \tilde{u}$ (resp. $\tilde{v} \nearrow q \nearrow v$) is $O(2^{\langle k(a_u,a_v)/2\rangle})$. Recall that lines 8 and 9 of Algorithm Basic(u, v) determine $\pi'_{\tilde{a}_u}$ as explained in the proof of Lemma 3, and also recall that $\pi'_{\tilde{a}_u}$ is equal to the number of edges p in $P_{\tilde{a}_u..\tilde{a}_v}$ such that $u_{\tilde{a}_u}^{\square} \nearrow p$ and $\beta_{\tilde{a}_u..\tilde{a}_v}(p) \nearrow v_{\tilde{a}_v}^{\square}$ as claimed in Fact 2. If $\tilde{v} \nearrow \tilde{u}$, then the number of edges p in $P_{\tilde{a}_u..\tilde{a}_v}$ with $u_{\tilde{a}_u}^{\square} \nearrow p$ and $\beta_{\tilde{a}_u..\tilde{a}_v}(p) \nearrow v_{\tilde{a}_v}^{\square}$ is $O(2^{\langle k(a_u,a_v)/2\rangle})$, implying that lines 8 and 9 with no modification execute in $O(2^{\langle k(a_u,a_v)/2\rangle})$ time using the d-inc bijection for $G_{\tilde{a}_u..\tilde{a}_v}$. Otherwise, we can decompose $\pi'_{\tilde{a}_u}$ as follows.

Fact 6. *The sum of $l(\tilde{u}, \tilde{v}) - (i_{\tilde{v}} - i_{\tilde{u}})$, the number of edges p in $P_{\tilde{a}_u..\tilde{a}_v}$ such that $u_{\tilde{a}_u}^{\square} \nearrow p \nearrow \tilde{u}$ and $\beta_{\tilde{a}_u..\tilde{a}_v}(p) \nearrow v_{\tilde{a}_v}^{\square}$, and the number of edges q in $Q_{\tilde{a}_u..\tilde{a}_v}$ such that $\tilde{v} \nearrow q \nearrow v_{\tilde{a}_v}^{\square}$ and $\tilde{u} \nearrow \beta_{\tilde{a}_u..\tilde{a}_v}(q)$ is equal to $\pi'_{\tilde{a}_u}$.*

Proof. Among the three items claimed to compose $\pi'_{\tilde{a}_u}$, the first item is equal to the number of edges p in $P_{\tilde{a}_u..\tilde{a}_v}$ such that $\tilde{u} \nearrow p$ and $\beta_{\tilde{a}_u..\tilde{a}_v}(p) \nearrow \tilde{v}$ due to Corollary 1. The third item is equal to the number of edges p in $P_{\tilde{a}_u..\tilde{a}_v}$ such that $\tilde{u} \nearrow p$ and $\tilde{v} \nearrow \beta_{\tilde{a}_u..\tilde{a}_v}(p) \nearrow v_{\tilde{a}_v}^{\square}$. Thus, the fact follows from Corollary 1 and definition of $\pi'_{\tilde{a}_u}$. □

Due to this decomposition, we can determine $\pi'_{\tilde{a}_u}$ also in $O(2^{\langle k(a_u,a_v)/2\rangle})$ time using the d-inc bijection for $G_{\tilde{a}_u..\tilde{a}_v}$. According to the above, we can modify Algorithm Basic(u, v) so as to execute in $O(2^{\langle k(a_u,a_v)/2\rangle})$ time as Algorithm Fast(u, v) presented in Fig. 9.

The data structure we propose consists of two components, set $\mathcal{S}$ containing the d-inc bijections for $G_{\tilde{a}_u..\tilde{a}_v}$ and all (u, v)-related a-strips of width less than $2^{\langle k(a_u,a_v)/2\rangle}$ and set $\mathcal{L}$ containing $l(\tilde{u}, \tilde{v})$ for all pairs (u, v) of vertices in G with $u \searrow v$. We define these components as follows.

Definition 2. For any positive integer k and any multiples a and a' of $2^{\langle k/2\rangle}$ with $0 \le a < a' \le m + n$ such that $k(a, a') = k$, let the a-strip between a and a' be called regular. Let $\mathcal{S}$ denote the set of the d-inc bijections for all regular a-strips.

Definition 3. Let the length collection for any regular strip $G_{a..a'}$ be the array of lengths $l(w, x)$ for all pairs of a vertex w in P and a vertex x in Q with $w \searrow x$ such that both d_w and d_x are multiples of $2^{\langle k(a,a')/2\rangle}$, where P (resp. Q) is the path consisting of all edges p (resp. q) in $P_{a..a'}$ (resp. $Q_{a..a'}$) such that $p \nearrow (0, 0)$ (resp. $(m, n) \nearrow q$) does not hold. Let $\mathcal{L}$ denote the set of the length collections for all regular a-strips.

From Definitions 2 and 3, we immediately obtain the following lemma because $2^{\langle k(a_u,a_v)/2\rangle} = \Theta(\sqrt{a_v - a_u}\log^{1+\epsilon}(a_v - a_u))$.

Lemma 5. *Data structure $(\mathcal{S}, \mathcal{L})$ supports $O(\sqrt{a_v - a_u}\log^{1+\epsilon}(a_v - a_u))$-time queries of $l(u, v)$ for any pair of vertices u and v in G with $u \searrow v$ using Algorithm* Fast(u, v).

Data structure $(\mathcal{S}, \mathcal{L})$ is $O(mn)$-time constructible as follows.

Lemma 6. *Set $\mathcal{S}$ can be constructed in $O(mn)$ time.*

Proof. Suppose that the d-inc bijections for all basic triangles introduced in the proof of Lemma 4 are available because they can be prepared in $O(mn)$ time. We obtain the d-inc bijection for each regular a-strip in ascending order of its width.

Let k be an arbitrary positive integer and let b and b' be arbitrary multiples of 2^k with $0 \leq b < b' \leq m+n$ such that $k(b,b') = k$. Hence, $G_{b..b'}$ is either a basic a-strip of width 2^k or a non-basic a-strip of width 2^{k+1} consisting of the union of two basic a-strips of width 2^k. There are $O((2^k/2^{\langle k/2\rangle})^2)$ regular a-strips $G_{a..a'}$ such that $b - 2^k < a \leq b$, $b' \leq a' < b' + 2^k$, and both a and a' are multiples of $2^{\langle k/2\rangle}$. If $a = b$ and $a' = b'$, then $G_{a..a'}$ is decomposed into the union of $O(n/2^k)$ basic triangles of width 2^k, implying that the d-inc bijection for $G_{a..a'}$ can be obtained in $O(kn)$ time. Otherwise, $G_{a..a'}$ can be decomposed into the union of $O(n/2^{\langle k/2\rangle})$ basic triangles of width $2^{\langle k/2\rangle}$ and any of $G_{a+2^{\langle k/2\rangle}..a'}$ with $a < b$ or $G_{a..a'-2^{\langle k/2\rangle}}$ with $b' < a'$, implying that the d-inc bijection for $G_{a..a'}$ can be obtained in $O((\log_2 2^{\langle k/2\rangle})n)$ time. Hence, the d-inc bijections for all such regular a-strips $G_{a..a'}$ for each pair of b and b' can be obtained in $O((k + (2^k/2^{\langle k/2\rangle})^2 \log_2 2^{\langle k/2\rangle})n)$ time. Since $2^{\langle k/2\rangle} = \Theta(2^{k/2}k^{1+\epsilon})$ and hence $\log_2 2^{\langle k/2\rangle} = \Theta(k)$, this execution time is $O(2^k n/k^{1+2\epsilon})$. There exist $O(m/2^k)$ pairs of b and b' for each k. Therefore, the lemma follows from the fact that the sum of $1/k^{1+2\epsilon}$ over all positive integers k is $O(1)$. □

Lemma 7. *If $\mathcal{S}$ is available, then $\mathcal{L}$ can be constructed in $O(mn)$ time.*

Proof. Let $G_{a..a'}$ be an arbitrary regular a-strip. Since the number of regular a-strips is $O(m)$ as shown in the proof of Lemma 6, it suffices to show below how to obtain the length collection for $G_{a..a'}$ in $O(n)$ time.

Let P and Q be the paths in Definition 3. Let W (resp. X) be the set of all vertices w (resp. x) in P (resp. Q) such that d_w (resp. d_x) is a multiple of $2^{\langle k(a,a')/2\rangle}$. Let Y be the set of all pairs (w,x) of w in W and x in X such that $w \searrow x$, so that the length collection for $G_{a..a'}$ consists of lengths $l(w,x)$ for all pairs (w,x) in Y. For any vertex w (resp. x) in W (resp. X), let w' (resp. x') denote the vertex in W (resp. X) such that $d_{w'} = d_w + 2^{\langle k/2\rangle}$ (resp. $d_{x'} = d_x - 2^{\langle k/2\rangle}$), if any, or the d-upper (resp. d-lower) end vertex of $P_{a..a'}$ (resp. $Q_{a..a'}$), otherwise. For any pair (w,x) in Y, let $\#(w,x)$ (resp. $\#_w(x)$) denote the number of edges p in $P_{a..a'}$ such that $w \nearrow p \nearrow w'$ (resp. $w \nearrow p$) and $x' \nearrow \beta_{a..a'}(p) \nearrow x$.

We obtain the length collection for each $G_{a..a'}$ in $O(n)$ time by executing the following four steps. The first step initializes $\#(w,z)$ to zero for all pairs (w,x) in Y. The second step determines values $\#(w,x)$ for all pairs (w,x) in Y in $O(n)$ time by increasing $\#(w,x)$ such that $w \nearrow p \nearrow w'$ and $x' \nearrow \beta_{a..a'}(p) \nearrow x$ by one for each edge p in P. The third step calculates values $\#_w(x)$ for all pairs (w,x) in Y based on the recurrence

$$\#_w(x) = \#(w,x) + \#_{w'}(x),$$

where $\#_{w'}(x) = 0$ for any pair (w,x) with $x \nearrow w'$ due to Lemma 1. The fourth step obtains values $l(w,x)$ for all pairs (w,x) in Y based on the recurrence

$$l(w,x) = l(w,x') - (i_{x'} - i_x) + \#_w(x),$$

where $l(w,x') = i_{x'} - i_w$ for any pair (w,x) with $x' \nearrow w$. The first, third, and fourth steps can be executed in time linear in the number of pairs (w,x) in Y, which is $O(n/\log^{2+2\epsilon}(a_v - a_u))$ because there are $O(n/2^{\langle k(a,a')/2\rangle})$ vertices w in W, each having $O((a'-a)/2^{\langle k(a,a')/2\rangle})$ vertices x in X such that $w \searrow x$. □

Consequently, from Lemmas 5, 6, and 7, the following theorem holds.

Theorem 1. *Data structure $(\mathcal{S},\mathcal{L})$ is $O(mn)$-time constructible (hence of size $O(mn)$) and supports $O(\sqrt{a_v - a_u}\log^{1+\epsilon}(a_v - a_u))$-time queries of $l(u,v)$ for any pair of vertices u and v in G with $u \searrow v$ using Algorithm* Fast(u,v).

As a particular case of Theorem 1, we finally obtain the following corollary.

Corollary 3. *For any positive constant ϵ and any pair of strings A and B, there exists an $O(mn)$-time constructible data structure that is of size $O(mn)$ and supports $O(\sqrt{m'+n'}\log^{1+\epsilon}(m'+n'))$-time queries of the LCS lengths of any pair of a substring A' of A and a substring B' of B, where m, n, m', and n' are respectively the lengths of A, B, A', and B'.*

5. Conclusion

This article proposed, for any positive constant ϵ and any pair of strings, a data structure of size $O(mn)$ that can be constructed in $O(mn)$ time and supports $O(\sqrt{m'+n'}\log^{1+\epsilon}(m'+n'))$-time queries of the LCS lengths of any pair of a substring of one string and a substring of the other, where m and n are the lengths of the strings for which the data structure is constructed and m' and n' are the lengths of the substrings whose LCS length is queried. To the best of the author's knowledge, this data structure is the first to achieve $o(m'+n')$-time queries only with $O(mn)$-time preprocessing.

There are still many unsolved problems with substring-substring LCS length data structures to be tackled. Such problems include, for example, whether a data structure of size $O((mn)^{1-\varepsilon})$ can support $O(m'+n')$-time queries for some positive constant ε and whether an $O(mn)$-time constructible data structure can support $O(\min(m',n'))$-time queries. Whether an $O(mn)$-time constructible data structure can support $O((m'+n')^{1/2-\varepsilon})$-time queries for some positive constant ε is also an interesting question.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] A. Abboud, A. Backurs, V.V. Williams, Tight hardness results for LCS and other sequence similarity measures, in: Proc. of the 56th Annual Symposium on Foundations of Computer Science, 2015, pp. 59–78.
[2] A. Apostolico, Improving the worst-case performance of the Hunt-Szymanski strategy for the longest common subsequence of two strings, Inf. Process. Lett. 23 (1986) 63–69.
[3] A. Apostolico, C. Guerra, The longest common subsequence problem revisited, Algorithmica 2 (1987) 315–336.
[4] K. Bringmann, M. Künnemann, Quadratic conditional lower bounds for string problems and dynamic time warping, in: Proc. of the 56th Annual Symposium on Foundations of Computer Science, 2015, pp. 79–97.
[5] B. Chazelle, A functional approach to data structures and its use in multidimensional searching, SIAM J. Comput. 17 (1988) 427–462.
[6] Y.-C. Chen, K.-M. Chao, On the generalized constrained longest common subsequence problems, J. Comb. Optim. 21 (2011) 383–392.
[7] F.Y.L. Chin, A. De Santis, A. Ferrara, N.L. Ho, S.K. Kim, A simple algorithm for the constrained sequence problems, Inf. Process. Lett. 90 (2004) 175–179.
[8] F.Y.L. Chin, C.K. Poon, A fast algorithm for computing longest common subsequences of small alphabet size, J. Inf. Process. 13 (1990) 463–469.
[9] H.N. Gabow, R.E. Tarjan, A linear time algorithm for a special case of joint set union, J. Comput. Syst. Sci. 30 (1985) 209–221.
[10] Z. Gotthilf, D. Hermelin, G.M. Landau, M. Lewenstein, Restricted LCS, in: Proc. International Symposium on String Processing and Information Retrieval, 2010, pp. 250–257.
[11] J.Y. Guo, F.K. Hwang, An almost-linear time and linear space algorithm for the longest common subsequence problem, Inf. Process. Lett. 94 (2005) 131–135.
[12] J.W. Hunt, T.G. Szymanski, A fast algorithm for computing longest common subsequences, Commun. ACM 20 (1977) 350–353.
[13] C.S. Iliopoulos, M.S. Rahman, A new efficient algorithm for computing the longest common subsequence, Theory Comput. Syst. 45 (2009) 355–371.
[14] G.M. Landau, M. Ziv-Ukelson, On the common substring alignment problem, J. Algorithms 41 (2001) 338–359.
[15] W.J. Masek, M.S. Paterson, A faster algorithm for computing string edit distances, J. Comput. Syst. Sci. 20 (1980) 18–31.
[16] N. Nakatsu, Y. Kambayashi, S. Yajima, A longest common subsequence algorithm suitable for similar text strings, Acta Inform. 18 (1982) 171–179.
[17] C. Rick, New algorithms for the longest common subsequence problem, Research Report No. 85123-CS, University of Bonn, 1994.
[18] Y. Sakai, An almost quadratic time algorithm for sparse spliced alignment, Theory Comput. Syst. 48 (2011) 189–210.
[19] Y. Sakai, A fast algorithm for multiplying min-sum permutations, Discrete Appl. Math. 159 (2011) 2175–2183.
[20] Y. Sakai, A substring-substring LCS data structure, Theor. Comput. Sci. 753 (2019) 16–34.
[21] Y. Sakai, S. Inenaga, A reduction of the dynamic time warping distance to the longest increasing subsequence length, in: Proc. the 31st International Symposium on Algorithms and Computation, 2020, pp. 6:1–6:16.
[22] A. Tiskin, Semi-local string comparison: algorithmic techniques and applications, Math. Comput. Sci. 1 (2008) 570–581.
[23] A. Tiskin, Fast distance multiplication of unit-Monge matrices, Algorithmica 71 (2015) 859–888, in: Proc. of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms, 2010, pp. 1287–1295.
[24] Y.-T. Tsai, The constrained longest common subsequence problem, Inf. Process. Lett. 88 (2003) 173–176.
[25] R. Wagner, M. Fisher, The string to string correction problem, J. ACM 21 (1974) 168–178.