International Journal of Computational Geometry & Applications

k-PAIRS NON-CROSSING SHORTEST PATHS IN A SIMPLE POLYGON*

EVANTHIA PAPADOPOULOU†
IBM TJ Watson Research Center
P.O. Box 218, Yorktown Heights, NY 10598, USA
E-mail: evanthia@watson.ibm.com

Received 26 August 1997
Revised 5 May 1999
Communicated by M. J. Atallah

ABSTRACT

This paper presents a simple $O(n+k)$ time algorithm to compute the set of k *non-crossing* shortest paths between k source-destination pairs of points on the boundary of a simple polygon of n vertices. Paths are allowed to overlap but are not allowed to cross in the plane. A byproduct of this result is an $O(n)$ time algorithm to compute a balanced geodesic triangulation which is easy to implement. The algorithm extends to a simple polygon with one hole where source-destination pairs may appear on both the inner and outer boundary of the polygon. In the latter case, the goal is to compute a collection of non-crossing paths of minimum total cost. The case of a rectangular polygonal domain where source-destination pairs appear on the outer and one inner boundary[12] is briefly discussed.

Keywords: non-crossing paths, simple polygon, polygon decomposition, geodesic triangulation, axis-parallel paths.

1. Introduction

The *k-pairs non-crossing shortest path problem* in a simple polygon is defined as follows. Given a simple polygon P of n vertices and k source-destination pairs of points $(s_i, t_i), i = 1, \ldots, k$, along the boundary of P, find the collection of k *non-crossing* shortest paths connecting every pair (s_i, t_i) that lie entirely in P. *Non-crossing* paths are allowed to overlap i.e., share vertices or edges, but they are not allowed to cross each other. Note that if the source-destination pairs are arbitrarily located on the polygon boundary there need not be any set of non-crossing paths between them. In the case where P contains a hole and the k source-destination pairs of points may appear in both the outer and the inner boundary of P the

*An extended abstract appeared in the proceedings of the *7th International Symposium on Algorithms and Computation*, December 1996, LNCS 1178, 305-314.

†This research was conducted while the author was at the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208, USA.

problem is to compute the collection of k *non-crossing* paths whose total length is minimum.

The *non-crossing* shortest path problem was introduced by D.T. Lee.[7] Takahashi *et al.*[11] considered the following problem: Given an undirected plane graph with nonnegative edge lengths, and k terminal pairs on two specified face boundaries, find k *non-crossing* paths in G, each connecting a terminal pair, whose total length is minimum. They presented an $O(n \log n)$ time algorithm, where n is the total number of vertices in the graph (including terminal pairs), using divide and conquer. In a subsequent paper,[12] Takahashi *et al.* considered the non-crossing shortest path problem in a polygonal domain of n rectangular obstacles and the L_1 metric. Given a set of n rectangular obstacles inside an outer rectangle and a set of $2k$ terminals (points) located on the boundaries of the outer rectangle and one of the obstacles, the problem asks for a set of non-crossing paths, each one connecting a set of terminal pairs without passing through any of the obstacles, and whose total length is minimum. Using a divide and conquer approach they provided an $O((n+k)\log(n+k))$ time algorithm.[12] Motivation for non-crossing shortest path problems comes from VLSI layout design.[7,11,12]

In the case of a simple polygon or a simple polygon with one hole, we can obtain an $O(k + n \log k)$ time algorithm using a divide and conquer approach similar to Refs. [11,12] and the $O(n)$ time algorithm of Lee and Preparata[8] for the single pair shortest path problem in a simple polygon. Alternatively, the problem can be solved by answering k shortest path queries using the data structure of Guibas and Hershberger.[4] This data structure can be constructed in $O(n)$ time and space and can answer shortest path queries between pairs of points in a simple polygon in $O(\log n + l)$ time where l is the number of links of the shortest path.

In this paper we present an $O(n+k)$ time algorithm for the k pairs non-crossing shortest path problem in a simple polygon and a polygon with one hole. Sources are assumed to be sorted around the polygon boundary. The collection of non-crossing shortest paths is organized in a forest that allows shortest path queries between any given pair to be answered in time proportional to the size of the path. In section 6, we make an observation that can slightly improve to $O(k+n \log n)$ the time complexity of the algorithm of Takahashi *et al.*[12] for a rectangular polygonal domain when sources are given sorted around the boundary. A byproduct of our algorithm for a simple polygon is a simple $O(n)$ method for computing a *balanced geodesic triangulation* of a simple polygon. A *balanced geodesic triangulation* is a decomposition of a simple polygon into triangle-like regions, called *geodesic triangles*, with the property that any line segment interior to P crosses only $O(\log n)$ geodesic triangles.[2,3] Such a decomposition finds often applications in problems involving simple polygons e.g., ray shooting or shortest path queries[2,3] and can be computed in $O(n)$ time[2]. The idea is to compute the collection of shortest paths between the following pairs of vertices $(v_1, v_{n/3}), (v_{n/3}, v_{2n/3}), (v_{2n/3}, v_1), (v_1, v_{n/6}), (v_{n/6}, v_{n/3}), (v_{n/3}, v_{n/2})$, etc. until a pair consists of consecutive vertices. Our $O(n)$ time algorithm for computing the collection of non-crossing shortest paths among the above $O(n)$ pairs of vertices gives an alternative method for computing a balanced geodesic triangulation which

is simple to implement. In general, a collection of non-crossing shortest paths between pairs of points on the boundary of a simple polygon can be used to obtain a useful polygon decomposition by including in the decomposition shortest paths between certain boundary points.

2. Preliminaries

Let P be a simple polygon and let ∂P be its boundary. Given two points $x, y \in P$, $\pi(x, y)$ denotes the shortest path from x to y that lies entirely in P. $\pi(x, y)$ is assumed to be directed from x to y. We are given a set of k source-destination pairs along ∂P, $\mathcal{ST} = \{(s_i, t_i), 1 \leq i \leq k\}$. We shall adopt the convention that sources $s_i, 1 \leq i \leq k$, are ordered clockwise along ∂P and that for any pair, s_i appears before t_i when we traverse ∂P clockwise starting at s_1. Source s_1 is regarded as the starting point along ∂P. Unless otherwise noted, we assume a clockwise traversal of ∂P starting at s_1.

Let ∂P be mapped onto a unit circle. Then the k source-destination pairs of points get mapped to k circle chords, $\overline{s_i t_i}, i = 1, \ldots, k$. Fig. 1 shows the mapping. If chord $\overline{s_i t_i}$ intersects $\overline{s_j t_j}$ then clearly $\pi(s_i, t_i)$ and $\pi(s_j, t_j)$ must cross each other. We say that pairs (s_i, t_i) and (s_j, t_j) are *non-crossing*, if and only if the corresponding chords in the unit circle, $\overline{s_i t_i}$ and $\overline{s_j t_j}$, do not intersect. For example, in Fig. 1, pairs $(s_1, t_1), (s_2, t_2), (s_3, t_3)$, and (s_5, t_5) are non-crossing while pairs (s_4, t_4) and (s_5, t_5) are crossing. It is easy to see that shortest paths between two pairs of points in $\mathcal{ST}$ are non-crossing if and only if the pairs are themselves non-crossing. We therfore have the following lemma.

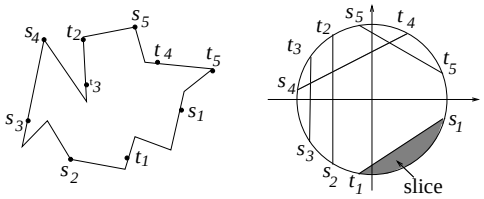

Fig. 1. Mapping of k source-destination pairs of vertices into k chords on the unit circle.

Lemma 1 *Given a simple polygon P of n vertices and k source-destination pairs along ∂P, (s_i, t_i), $1, \ldots, k$, detecting if any two shortest paths connecting s_i to t_i cross each other can be done in $O(n + k)$ time.*

Given the unit circle representing ∂P and a pair (s_i, t_i), the region of the unit circle enclosed by the chord $\overline{s_i t_i}$ and the arc from s_i to t_i is referred to as a *slice* and is denoted as $sl(s_i, t_i)$. (A clockwise traversal is assumed). For the pair (s_1, t_1) we define two slices, $sl(s_1, t_1)$ and $sl'(s_1, t_1)$, where $sl'(s_1, t_1)$ is the complement of $sl(s_1, t_1)$ i.e., the region of the unit circle enclosed by the chord $\overline{s_1 t_1}$ and the arc from t_1 to s_1. In Fig. 1, $sl(s_1, t_1)$ is shown shaded. Note that paths $\pi(s_i, t_i)$ and $\pi(s_j, t_j)$ are *non-crossing* if and only if the corresponding slices, $sl(s_i, t_i)$ and $sl(s_j, t_j)$, do not intersect or one of them totally contains the other. In the former case, the two non-crossing pairs are said to be in *series*, and in the latter, they are

said to be in *parallel*. The destination points of pairs that are in parallel are in reverse order as the source points. Fig. 2 shows an example of a set of non-crossing slices; slices $sl(s_1,t_1), sl(s_3,t_3), sl(s_6,t_6)$ are in series while $sl(s_1,t_1), sl(s_2,t_2)$ are in parallel.

Consider now a tree, T_{sl}, representing the set of non-crossing slices where the root corresponds to the unit disk and each node represents a slice. The children of the root are $sl(s_1,t_1)$ and $sl'(s_1,t_1)$. Slices that are totally contained in $sl(s_i,t_i)$ are descendants of $sl(s_i,t_i)$ in the tree. The immediate children of $sl(s_i,t_i)$ (resp. $sl'(s_1,t_1)$) are those slices in series that are totally contained in $sl(s_i,t_i)$ (resp. $sl'(s_1,t_1)$) but are not contained in any other descendent of $sl(s_i,t_i)$. Fig. 2 shows the tree representation of the corresponding slices. The nodes are labeled by the slice number; slice 0 refers to the unit disk. Note that T_{sl} may be balanced or completely unbalanced.

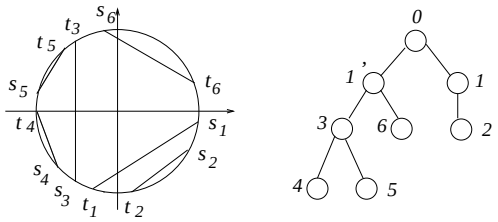

Fig. 2. Non-crossing slices and the tree representation.

Let $R = \{r_1, r_2, \ldots, r_{2k}\}$ be the set of all points in $\mathcal{ST}$. We assume that R is ordered according to a clockwise traversal of ∂P starting at $r_1 = s_1$. Given any two points x, y along ∂P, we say that y *follows* x, and denote as $x < y$, if and only if y follows x in a clockwise traversal of the polygon boundary starting at s_1. Given $R' \subseteq R$ and $r_j \in R'$, let $n_{R'}(r_j)$ denote the point in R' following r_j and $p_{R'}(r_j)$ denote the point in R' preceding r_j. For $R' = R$, $n_R(r_j) = r_{j+1}$ and $p_R(r_j) = r_{j-1}$. Let $R' \subseteq R$ be such that the shortest paths between any two pairs of distinct points are disjoint. The area enclosed by the collection of shortest paths $\pi(r_j, n_{R'}(r_j))$ for every $r_j \in R'$ is called the *geodesic polygon* induced by R'. In Fig. 3, the shaded area is the geodesic polygon induced by $R' = \{r_1, r_4, r_6, r_7, r_{10}\}$. The last common vertex along the paths emanating from r_j, $\pi(r_j, n_{R'}(r_j))$ and $\pi(r_j, p_{R'}(r_j))$, is an *apex* of the geodesic polygon and is denoted as $\alpha(r_j)$. $\alpha(r_j)$ is also referred to as the *apex* of r_j. If r_j is the only common point of $\pi(r_j, n_{R'}(r_j))$ and $\pi(r_j, p_{R'}(r_j))$ then $\alpha(r_j) = r_j$. In Fig. 3, $\alpha(r_4) = a$ and $\alpha(r_1) = r_1$. The common part of $\pi(r_j, n_{R'}(r_j))$ and $\pi(r_j, p_{R'}(r_j))$ (i.e., $\pi(r_j, \alpha(r_j))$) is referred to as the *geodesic link* of r_j. Two consecutive apexes of the geodesic polygon, $\alpha(r_j)$ and $\alpha(n_{R'}(r_j))$, are joined by their shortest path, $\pi(\alpha(r_j), \alpha(r_{j+1}))$, which is clearly a convex chain with convexity facing towards the interior of the geodesic polygon.

If R' contains pairs of distinct points whose shortest paths share some common part, the collection of shortest paths $\pi(r_j, n_{R'}(r_j))$ induces more than one geodesic polygon in P. In Fig. 4, set $R = \{r_1, \ldots, r_{10}\}$ induces four geodesic polygons g_1, g_2, g_3, g_4, where $g_1 = \{r_1, a, r_5, r_6, c, r_{10}\}$, $g_2 = \{a, r_2, r_3, b\}$, $g_3 = \{d, e, r_9\}$, and $g_4 = \{e, r_7, r_8\}$. Two paths $\pi(r_i, n_{R'}(r_i))$ and $\pi(r_j, n_{R'}(r_j))$, $j > i+1$, that are not

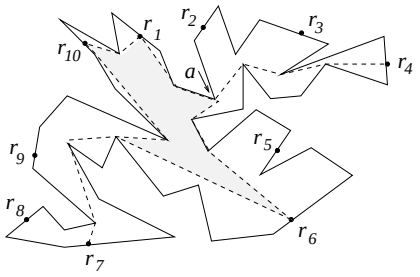

Fig. 3. The geodesic polygon induced by $R' = \{r_1, r_4, r_6, r_7, r_{10}\}$.

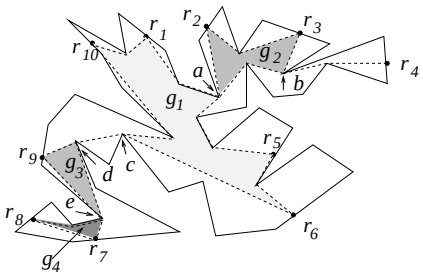

Fig. 4. The geodesic decomposition of $R = \{r_1, \ldots, r_{10}\}$.

disjoint share a subpath of at least one vertex which is referred to as a *geodesic link*. For example, in Fig. 4, segment $\overline{cd}$ is common to $\pi(r_6, r_7)$ and $\pi(r_9, r_{10})$ i.e., $\overline{cd}$ is a geodesic link. Note that a geodesic link may consist of a single vertex in which case the incident geodesic polygons have a common apex (e.g., vertex a in Fig. 4). The collection of geodesic polygons and links induced by $\pi(r_i, n_{R'}(r_i))$, for every $r_i \in R'$, is called the *geodesic decomposition* of P induced by R' and is denoted as $\gamma(R')$. In Fig. 4, the geodesic decomposition of R consists of four geodesic polygons g_1, g_2, g_3, g_4. The geodesic link joining g_1 and g_2 is vertex a, the geodesic link joining g_1 and g_3 is $\pi(c, d) = \overline{cd}$, and the geodesic link joining g_3 and g_4 is vertex e. The geodesic link of r_4 is $\pi(r_4, b)$. For every $r_j, j \neq 4$, $\alpha(r_j) = r_j$, and $\alpha(r_4) = b$.

The root of the geodesic decomposition is considered to be point r_1. This induces a unique directed path from the root to every geodesic polygon which defines a parent-child relation between the geodesic polygons. The apex of a geodesic polygon g incident to its parent is called the *main apex* of g and is denoted as $\alpha(g)$. Given a geodesic polygon g and a point $r_j \in R'$, let $\alpha(r_j, g)$ denote the first apex of g along $\pi(r_j, \alpha(g))$. (If g contains $\alpha(r_j)$ then $\alpha(r_j, g) = \alpha(r_j)$). In Fig. 4, $\alpha(r_4, g_1) = a$ and $\alpha(r_4, g_3) = d$.

Given a vertex v along a convex chain C and a point p, we say that v is *supporting* (with respect to $\overline{pv}$) or that segment $\overline{pv}$ is *supporting* to C if the line containing $\overline{pv}$ is tangent to C.

3. Overview and Data Structures

The collection of shortest paths between every pair of points in $\mathcal{ST}$ (given that $\mathcal{ST}$ is non-crossing) forms a forest denoted as $\mathcal{E}$. Our algorithm computes $\mathcal{E}$ in

$O(n+k)$ time and processes it to answer shortest path queries between any pair in $\mathcal{ST}$ in time proportional to the length of the path. In particular, any tree of $\mathcal{E}$ can be transformed into a rooted tree by assigning the source of minimum index as the root. Let $nca_{\mathcal{E}}(s_i,t_i),(s_i,t_i) \in \mathcal{ST}$, denote the nearest common ancestor of s_i and t_i in $\mathcal{E}$. (Note that s_i and t_i must belong to the same tree of $\mathcal{E}$). Then $\pi(s_i,t_i) = \pi(s_i, nca_{\mathcal{E}}(s_i,t_i)) \cup \pi(nca_{\mathcal{E}}(s_i,t_i),t_i)$. Computing the nearest common ancestor in $\mathcal{E}$ for every pair $(s_i,t_i) \in \mathcal{ST}$ can be easily done in linear time in a bottom-up fashion using T_{sl} as a guide. In the case where all source-destination pairs are in series, we can compute the collection of shortest paths between them in linear time using any ordinary shortest path algorithm for a single pair of points.[8,1,5] In particular, we can use the Lee-Preparata algorithm[8] for each pair independently. The following lemma assures that the time complexity remains linear.

Lemma 2 *The collection of non-crossing shortest paths between k source-destination pairs of points on ∂P can be found in linear time if the pairs are in series.*

Proof. Assuming that P is arbitrarily triangulated, the *sleeve* of a pair (s_i,t_i) is the subpolygon of P consisting of the triangles that $\pi(s_i,t_i)$ passes through.[8] It has been shown that the total time required to compute $\pi(s_i,t_i)$ is proportional to the size of the sleeve.[8] The sleeves associated with the given source-destination pairs $(s_i,t_i), 1,\ldots,k$ are either disjoint, or they share some triangles. Since the source-destination pairs are in series, a triangle can be intersected by the shortest paths of at most three such pairs (otherwise some pairs would be in parallel, see Fig. 5). Thus, each triangle can be shared by at most 3 sleeves. Therefore, the total number of triangles in all k sleeves is $O(n+k)$. □

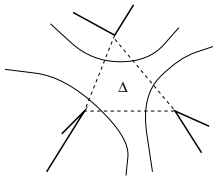

Fig. 5. Triangle Δ may be intersected by at most three paths between pairs in series.

In the general case, where pairs are both in series and in parallel, we compute the collection of shortest paths, $\mathcal{E}$, in a bottom-up fashion using the tree of slices, T_{sl}, as a guide. In phase 1, we compute the geodesic decomposition of P induced by R. This reveals the collection of shortest paths for every pair at a leaf node of T_{sl}. For any pair (s_i,t_i) at the bottom level of T_{sl}, we add $\pi(s_i,t_i)$ to $\mathcal{E}$ and we color all edges along $\pi(s_i,t_i)$ red. The remaining edges of the geodesic decomposition are not colored and are said to be *white*. Let R^q, for $1 \leq q \leq h$, where h is the height of T_{sl}, denote the set of points appearing at levels 1 to $(h-q+1)$ in T_{sl}. Note that $R^1 = R$ and $R^h = \{s_1,t_1\}$. In phase q, $1 < q \leq h$, we compute the geodesic decomposition of P induced by R^q which reveals the collection of shortest paths for every pair (s_i,t_i) at level $(h-q+1)$ of T_{sl}. Edges of the decomposition at phase q are either red or white; red edges are those that have been output to $\mathcal{E}$ in some previous phase. At the end of phase q, we output to $\mathcal{E}$ all white edges along $\pi(s_i,t_i)$

for every pair (s_i, t_i) at level $(h - q + 1)$ of T_{sl} and color them red.

At every phase $q, 1 \leq q \leq h$, the geodesic decomposition of R^q is maintained. Geodesic polygons and links are kept as doubly linked lists of their vertices. In general, an apex is adjacent to two vertices in its geodesic polygon and one vertex in the incident geodesic link. An exception are apexes that are common to two geodesic polygons (e.g., apex a in Fig. 4) which are incident to two vertices in each of the incident geodesic polygons. Edges of geodesic polygons and geodesic links are marked as red or white, where red edges have been output to $\mathcal{E}$. In more detail, the geodesic decomposition is kept as follows: Every vertex v along a geodesic polygon g has a pointer $next(v)$ and a pointer $prev(v)$ to the vertex following and preceding v in g respectively. Every apex α, other than a main apex, has an additional pointer $nlink(\alpha)$ to the vertex following α along the incident geodesic link. The main apex of g, $\alpha(g)$, has a pointer $plink(\alpha(g))$ to the vertex preceding $\alpha(g)$ along the incident geodesic link. In other words, every apex has a total of three pointers. Every vertex u along a geodesic link has a pointer $nlink(u)$ and a pointer $plink(u)$ to the vertex following and preceding u respectively along the geodesic link. For consistency, if a geodesic link consists of a single vertex u we can introduce a dummy vertex u' representing u in the incident geodesic polygon, and let $nlink(u) = u'$ and $plink(u') = u$.

To achieve linear time complexity we need the ability to extract the white edges of $\pi(s_i, t_i)$ for some pair (s_i, t_i) at level $(h - q + 1)$ of T_{sl} without visiting the red edges along $\pi(s_i, t_i)$. For this purpose we maintain a doubly linked list of the polygon vertices incident to white edges, referred to as the *white-list*. The white-list includes $\alpha(r_j)$ for every $r_j \in R^q$. Two vertices v and u are joined by a link in the white-list if and only if v and u belong to the shortest path between two consecutive apexes $\alpha(r_j)$ and $\alpha(n_{R^q}(r_j))$ and either $\overline{vu}$ is a white edge or $\pi(v, u)$ is a maximal red subpath of $\pi(\alpha(r_j), \alpha(n_{R^q}(r_j)))$. Every link of the white-list corresponds either to a white edge of the decomposition, referred to as a *white link*, or to a maximal sequence of red edges, referred to as a *red link*. Note that a white edge along a geodesic link appears twice in the white-list and thus each of the incident vertices also appears twice.

4. The Algorithm

Let's first consider the geodesic decomposition at phase 1, i.e., the geodesic decomposition of R. Pairs $(r_i, r_{i+1}), 1 \leq i \leq 2k$, are clearly in series and therefore the collection of $\pi(r_i, r_{i+1})$ can be computed in $O(n + k)$ time (Lemma 2). Once $\pi(r_i, r_{i+1})$ for every $i, 1 \leq i \leq 2k$, have been computed, the geodesic decomposition of P can be easily obtained in linear time. For every pair of points (s_i, t_i) at the bottom level of T_{sl}, we color edges along $\pi(s_i, t_i)$ red and output them to $\mathcal{E}$. We also build the white list by a simple scan starting at $\alpha(r_1)$.

To compute the geodesic decomposition of $R^q, q > 1$, we use the geodesic decomposition of R^{q-1}. To facilitate updating we use the shortest path tree from s_1 to all points in R. Computing the shortest path tree from s_1 can be done in linear time using the algorithm of Guibas et al.[5] or the simpler to implement algorithm

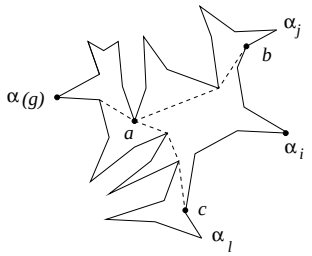

Fig. 6. $\gamma(\alpha(g), \alpha_j, \alpha_i, \alpha_l)$

of Hershberger and Snoeyink.[6] Given two points $r_i, r_j \in R$, the nearest common ancestor of r_i and r_j in the shortest path tree from s_1 is denoted as $nca(r_i, r_j)$. In the following, we will use the term nearest common ancestor of r_i and r_j to denote $nca(r_i, r_j)$ without explicitly referring to the shortest path tree from s_1.

Suppose that the geodesic decomposition of P induced by R^{q-1} has been computed, and that we wish to compute the geodesic decomposition induced by R^q. For brevity, we will use γ to denote the geodesic decomposition during phase q, $n(r_j)$ to denote $n_{R^q}(r_j)$, the point following r_j in R^q, and $p(r_j)$ to denote $p_{R^q}(r_j)$, the point preceding r_j in R^q. At the end of phase q, $\gamma = \gamma(R^q)$.

Consider a point $r_j \in R^q$ such that $n(r_j) \neq n_{R^{q-1}}(r_j)$. In phase q, every point $r_i \in R^{q-1}$ between r_j and $n(r_j)$ must be deleted from the geodesic decomposition. For this purpose, we incrementally compute $\pi(\alpha(r_j), \alpha(r_i))$ for every $r_i \in R^{q-1}$ between r_j and $n(r_j)$ (including $n(r_j)$), updating at the same time the geodesic decomposition. At the end, we have the geodesic decomposition updated according to $\pi(\alpha(r_j), \alpha(n(r_j)))$.

Suppose now that $\pi(\alpha(r_j), \alpha(r_i))$ has been computed for some $r_i \in R^{q-1}, r_j < r_i < n(r_j)$. Suppose also that the geodesic decomposition has been updated accordingly i.e., $\gamma = \gamma(R^q \cup \{r \in R^{q-1}, r \geq r_i\})$. Let r_l be the point following r_i in R^{q-1} i.e., $r_l = n_{R^{q-1}}(r_i)$. Note that r_j, r_i, and r_l are currently consecutive points. Let g be the geodesic polygon containing $\alpha(r_i)$, the apex of r_i. For brevity, let α_j, α_i, and α_l denote $\alpha(r_j, g), \alpha(r_i)$, and $\alpha(r_l, g)$ respectively. Note that α_j or α_l may coincide with the main apex of g. To update the geodesic decomposition, it is enough to update the geodesic polygon g according to $\pi(\alpha_j, \alpha_l)$.

Consider the geodesic decomposition induced by $\{\alpha(g), \alpha_j, \alpha_i, \alpha_l\}$, denoted as $\gamma(\alpha(g), \alpha_j, \alpha_i, \alpha_l)$ (see Figs. 6, 7). If $\alpha(g)$ coincides with α_j or α_l, $\gamma(\alpha(g), \alpha_j, \alpha_i, \alpha_l)$ must form a geodesic triangle (i.e., have three apexes). Otherwise, $\gamma(\alpha(g), \alpha_j, \alpha_i, \alpha_l)$ forms either a geodesic quadrilateral (a geodesic polygon of four apexes, see Fig. 6) or two geodesic triangles (see Fig. 7). In any case, let $g(a_i)$ denote the geodesic polygon of $\gamma(\alpha(g), \alpha_j, \alpha_i, \alpha_l)$ containing α_i. Let a, b and c denote the apexes of $\gamma(\alpha(g), \alpha_j, \alpha_i, \alpha_l)$, other than α_i, such that a is the last common vertex along $\pi(\alpha(g), \alpha_j)$ and $\pi(\alpha(g), \alpha_l)$, b is the last common vertex along $\pi(\alpha_j, \alpha(g))$ and $\pi(\alpha_j, \alpha_i)$, and c is the last common vertex along $\pi(\alpha_l, \alpha_i)$ and $\pi(\alpha_l, \alpha(g))$ (see Fig. 6). If the main apex of g coincides with a_j (resp. a_l) then a coincides with b (resp. c). Note that a is the nearest common ancestor of r_j and r_l. If $g(\alpha_i)$ is a geodesic triangle, let a' denote the main apex of $g(\alpha_i)$ (see Fig. 7). Note that a' is

either the nearest common ancestor of r_j and r_i or the nearest common ancestor of r_i and r_l. To update the geodesic polygon g, we need to compute $\pi(\alpha_j, \alpha_l)$ i.e., it is enough to compute $\pi(b, c)$.

Suppose $g(\alpha_i)$ is a geodesic quadrilateral (see Fig. 6). To compute $\pi(b, c)$ we basically need to compute segment $\overline{yz}$ where $y \in \pi(\alpha_i, b) \cup \pi(b, a)$ and $z \in \pi(\alpha_i, c) \cup \pi(c, a)$ such that y and z are *supporting* and $\overline{yz}$ lies entirely in P (see Fig. 8). Then $\pi(b, c)$ consists of $\pi(b, y)$, segment $\overline{yz}$, and $\pi(z, c)$. Since $\pi(b, y)$ and $\pi(z, c)$ are both known we only need to compute segment $\overline{yz}$. There are four possible cases for segment $\overline{yz}$:

Case 1: $y \in \pi(\alpha_i, b)$ and $z \in \pi(\alpha_i, c)$ (Fig. 8(a)).
Case 2: $y \in \pi(b, a)$ and $z \in \pi(\alpha_i, c)$, $y \neq b$ (Fig. 8(b)).
Case 3: $y \in \pi(\alpha_i, b)$ and $z \in \pi(c, a)$, $z \neq c$ (Fig. 8(c)).
Case 4: $y \in \pi(b, a)$ and $z \in \pi(c, a)$, $y \neq b$, $z \neq c$ (Fig. 8(d)).

If $g(\alpha_i)$ is a geodesic triangle then $\pi(\alpha_j, \alpha_l)$ is known (see Fig. 7). (Recall that $\pi(\alpha(g), a_j)$ and $\pi(\alpha(g), a_l)$ belong to the shortest path tree from s_1). If a', the main apex of $g(\alpha_i)$, is the nearest common ancestor of r_j and r_i (see Fig. 7(a)), let z be a' and let y be the vertex following z along $\pi(a', b)$. Segment $\overline{yz}$ is supporting to both $\pi(a', b)$ and $\pi(a', \alpha_i)$. Since $y \in \pi(a, b)$ and $z \in \pi(\alpha_i, c)$, this can be regarded as case (2) for $y \neq b$ or case (1) for $y = b$. If a' is the nearest common ancestor of r_i and r_l (see Fig. 7(b)) let y be a' and z be the following vertex along $\pi(a', c)$. Since $y \in \pi(\alpha_i, b)$ and $z \in \pi(c, a)$ this can be regarded as case (3) for $z \neq c$ or case (1) for $z = c$. Thus, there is no need to differentiate between updating a geodesic quadrilateral and a geodesic triangle.

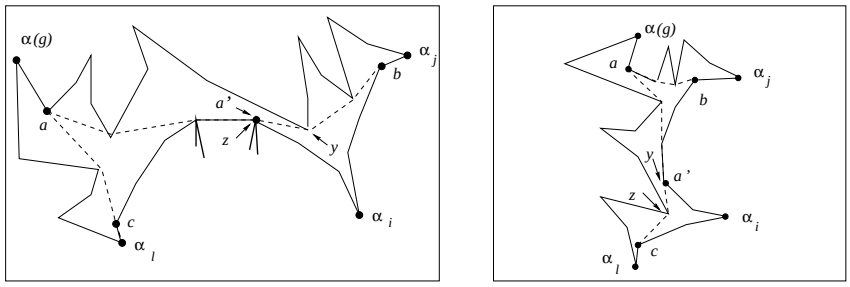

Fig. 7. $g(\alpha_i)$ is a geodesic triangle.

To determine which of the four cases is the one occurring and determine vertices y and z, we advance a variable vertex v along $\pi(\alpha_i, b) \cup \pi(b, a)$ and a variable vertex u along $\pi(\alpha_i, c) \cup \pi(c, a)$ until v and u reach y and z respectively (v and u start at α_i). The problem is that vertices a, b and c as well as apexes α_j and α_l are not known in advance. However, useful information can be obtained while advancing v and u using the shortest path tree from s_1. The following properties which can be easily derived from the definitions, simplify the advancement of v and u.

Property 1: For any vertex v along $\pi(\alpha_i, \alpha_j)$, v coincides with b if and only if the predecessor of v on the shortest path from s_1, denoted as $pred(v)$, is supporting

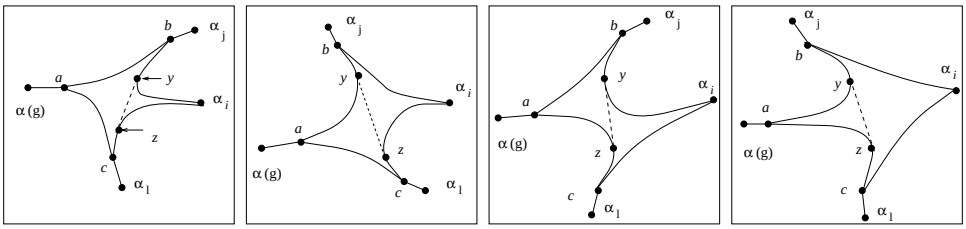

Fig. 8. The four cases for segment $\overline{yz}$.

with respect to $\overline{pred(v)v}$ and $pred(v) \notin \pi(\alpha_i, \alpha_j)$. Similarly, for any vertex u along $\pi(\alpha_i, \alpha_l)$, u coincides with c if and only if $pred(u)$ is supporting with respect to $\overline{pred(u)u}$ and $pred(u) \notin \pi(\alpha_i, \alpha_l)$.

Property 2: Segment $\overline{vu}$ intersects $\pi(a, c)$ (resp. $\pi(a, b)$) if and only if $pred(u)$ (resp. $pred(v)$) lie at the same side of $\overline{vu}$ as α_i.

Due to property 1 we can easily determine when v and u reach b and c respectively during the advancement. Furthermore, due to property 2, segment $\overline{vu}$ can be advanced so that it always lies entirely in the interior of $g(\alpha_i)$. If at any point during the advancement it is determined that $\overline{vu}$ intersects $\pi(a, c)$ (resp. $\pi(a, b)$), we can conclude that we have case 3 or case 4 (resp. case 2 or case 4). In case 4, the advancement of $\overline{vu}$ can be done similarly to the construction of a U-hull[9] (see Ref. [10], page 128). In case 4, vertex a is the nearest common ancestor of r_j and r_l, and thus a can be easily determined. The details of the algorithm to advance segment $\overline{vu}$ are given in the appendix.

Once segment $\overline{yz}$ is determined the update of g can be briefly stated as follows:

Case 1: Remove $\pi(y, \alpha_i) \cup \pi(\alpha_i, z)$ and add $\pi = \overline{yz}$.
Case 2: Remove $\pi(b, \alpha_i) \cup \pi(\alpha_i, z)$ and add $\pi = \pi(b, y) \cup \overline{yz}$.
Case 3: Remove $\pi(c, \alpha_i) \cup \pi(\alpha_i, y)$ and add $\pi = \pi(c, z) \cup \overline{yz}$.
Case 4: Remove $\pi(b, \alpha_i) \cup \pi(\alpha_i, c)$ and add $\pi = \pi(b, y) \cup \overline{yz} \cup \pi(z, c)$.

To update g in any of the four cases we walk along the corresponding path $\pi \subseteq \pi(b, c)$ starting at vertices b or c. (In case 4, we walk along $\pi_1 = \pi(b, y)$ and $\pi_2 = \pi(c, z)$ and then consider segment $\overline{yz}$). Fig. 9 illustrates the update of g for case 3 where $\pi = \pi(c, y)$. Every edge along π induces a new geodesic polygon or becomes part of a new geodesic link. In Fig. 9, edge $\overline{y_2 y_3}$ and vertex y_4 become geodesic links; the remaining edges induce new geodesic polygons which are shown shaded. To do the actual update we simply need to update the pointers $prev(y_i), next(y_i), plink(y_i)$ and $nlink(y_i)$ for every vertex y_i along π. The details are given in the appendix.

The edges visited by the algorithms to determine $\overline{yz}$ and update g are either deleted from g or they are part of the new path $\pi \subseteq \pi(b, c)$. For each visited edge only constant time is spent. Edges along π are either new white edges to be added to the geodesic decomposition or they become part of a geodesic link. Once an edge becomes part of a geodesic link, it is never visited again until it gets output or deleted from the geodesic decomposition. Thus, over all phases, the time spent for

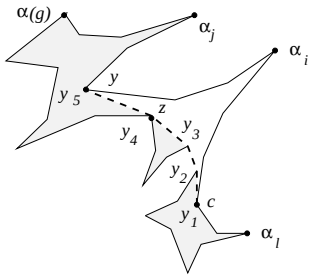

Fig. 9. The update of g in case 3.

updates of the geodesic decomposition is proportional to the total number of edges appearing in the geodesic decomposition throughout the algorithm. Since no two such edges intersect and since they are always incident to vertices or points in R, their number is bounded by $O(n+k)$.

To update the white list, let π' be the path obtained from π by substituting any maximal sequence of red edges by a single red link. Let h_1 and h_2 be the vertices in the white-list preceding and following (or coinciding with) the endpoints of π respectively. (Recall that during the update, all vertices in the white-list between h_1 and h_2 were deleted from the geodesic decomposition). Delete the part of the white list between h_1 and h_2 and merge π'. The time spent is proportional to the number vertices deleted plus the size of π. Thus,

Lemma 3 *The total time spent for updating the geodesic decomposition and the white-list in every phase of the algorithm is $O(n+k)$.*

To complete the update, we need to compute the nearest common ancestor of r_j and r_l because r_j and r_l become neighbors after the deletion of r_i. Clearly $nca(r_j, r_l)$ is either $nca(r_j, r_i)$ or $nca(r_i, r_l)$. In particular, $nca(r_j, r_l)$ must be the one appearing first along $\pi(s_1, r_i)$ i.e., the one nearest to the root. Assigning a level number to all nodes of the shortest path tree from s_1 allows to compute $nca(r_j, r_l)$ from $nca(r_j, r_i)$ and $nca(r_i, r_l)$ in constant time.

After the geodesic decomposition induced by R^q has been computed, we need to extract white edges of $\pi(s_i, t_i)$ for every pair (s_i, t_i) at level $(h-q+1)$ of T_{sl}, color them red, output them to $\mathcal{E}$, and update the white list. Starting at $\alpha(s_i)$, we follow the white list until $\alpha(t_i)$ adding at the same time the white edges that we encounter to $\mathcal{E}$. Note that white edges along $\pi(s_i, t_i)$ that are part of geodesic links appear twice along the white-list and that both occurrences must become red. Finally, substitute $\pi(\alpha(s_i), \alpha(t_i))$ by a single red link between $\alpha(s_i)$ and $\alpha(t_i)$. The time spent is proportional to the number of white edges that get output to $\mathcal{E}$ i.e., the number of edges that become red.

We therefore conclude that $\mathcal{E}$ can be computed in $O(n+k)$ time.

5. A Simple Polygon With One Hole

Similarly to Refs. [11,12], our algorithm can be extended to the case of a simple polygon with one hole. Let ∂P and ∂Q denote the outer and inner boundary

respectively i.e., ∂Q is the boundary of the hole. Source-destination pairs in $\mathcal{ST}$ may appear on both the outer and the inner boundary.

In the presence of one hole the dual graph of an arbitrary triangulation contains a cycle. Thus, for every pair of points in the polygon we have two kinds of "pull taut" paths: one clockwise and one counterclockwise around ∂Q. Let $\pi_c(s_i, t_i)$ and $\pi_{cc}(s_i, t_i)$ denote the shortest paths from s_i to t_i that pass clockwise and counterclockwise around Q respectively, where $(s_i, t_i) \in \mathcal{ST}$. The shortest path from s_i to t_i is the shortest between $\pi_c(s_i, t_i)$ and $\pi_{cc}(s_i, t_i)$. Note that if any of the two paths has no common part with ∂Q then that path must be the shortest one.

Let's first assume that there is a pair (s_i, t_i) with $s_i \in \partial P$ and $t_i \in \partial Q$. Then once a path between s_i and t_i has been routed there is only one way (if any) to route the rest of the pairs in a non-crossing fashion. Furthermore, a path can be routed in only two ways: clockwise or counterclockwise around the hole Q. Thus, we can compute $\pi_c(s_i, t_i)$ and $\pi_{cc}(s_i, t_i)$, solve the problem separately for each case, and chose the solution of minimum total cost. In the following, we concentrate in determining non-crossing shortest paths assuming that $\pi_c(s_i, t_i)$ has been routed. The counterclockwise case can be treated in the same way.

Suppose that we *cut* the polygon along $\pi_c(s_i, t_i)$. *Cutting* corresponds to treating every diagonal along $\pi_c(s_i, t_i)$ as a pair of boundary edges. By cutting along $\pi_c(s_i, t_i)$ we produce a number of simple subpolygons. The problem can then be reduced to finding non-crossing shortest paths for each subpolygon. Fig. 10 illustrates two of the polygons obtained by cutting along $\pi(s_i, t_i)$. The lightly shaded subpolygon is the one containing the first and the last diagonal along $\pi(s_i, t_i)$ with endpoints on both ∂P and ∂Q. We shall refer to the latter subpolygon as the *main* subpolygon produced by the cutting operation.

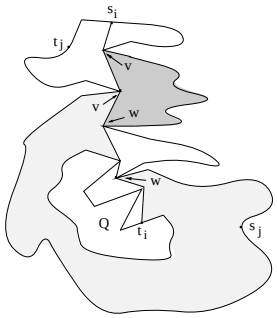

Fig. 10. Cutting P along $\pi_c(s_i, t_i)$.

Let X be a subpolygon produced by the cutting operation and let (s_j, t_j) be a source-destination pair such that $s_j \in \partial X$ and $t_j \notin \partial X$. The shortest non-crossing path from s_j to t_j must share some common part with the boundary of X along $\pi_c(s_i, t_i)$. Let v and w be the first and last vertex respectively along $\pi_c(s_i, t_i) \cap \partial X$; if X is the main subpolygon, let w be the last vertex along $\pi_c(s_i, t_i) \cap \partial P$. In Fig. 10, vertices v and w for each subpolygon are indicated by arrows contained entirely within the respective subpolygons. The shortest path from s_j to t_j which is non-crossing to $\pi(s_i, t_i)$ (if any) must clearly pass through vertex v or w. In

particular, it is not hard to see that $\pi(s_j, t_j)$ must pass through v if and only if: 1) vertex $v \in \partial P$ and t_j is encountered before s_i as we walk on ∂P starting at v and moving away from ∂X or 2) vertex $v \in \partial Q$ and $t_j \in \partial P$. Otherwise, $\pi(s_j, t_j)$ must pass through w. According to this observation we can map t_j to either v or w and consider pair (s_j, v) or (s_j, w) respectively as a source-destination pair in X. Using this transformation for any source-destination pair with terminals in different subpolygons we can reduce the problem into computing non-crossing shortest paths within every subpolygon X.

Let's now assume that there is no source-destination pair in $\mathcal{ST}$ with terminals on both ∂P and ∂Q. If all pairs appear on ∂Q then the problem is easy and we skip the discussion. Let's assume that all source-destination pairs in $\mathcal{ST}$ appear on ∂P. Then $\mathcal{ST}$ admits a set of non-crossing shortest paths under exactly the same conditions as in the simple polygon case. This collection of paths has minimum total cost if and only if every path is itself shortest. The clockwise paths $\pi_c(s_i, t_i)$ for every pair $(s_i, t_i) \in \mathcal{ST}$, can be computed similarly to the simple polygon case. For this purpose we build the geodesic decomposition by computing the clockwise shortest path between every two consecutive points in $\mathcal{ST}$. Since the paths are all clockwise, the computation can be done as in the ordinary case by breaking the cycle of the dual graph. For pair (s_1, t_1) both $\pi_c(s_1, t_1)$ and $\pi_c(t_1, s_1) = \pi_{cc}(s_1, t_1)$ get computed and thus at the end we can chose the shortest among the two. Let's assume without loss of generality that the shortest path from s_1 to t_1 is $\pi_c(s_1, t_1)$. Then for every descendent of (s_1, t_1) in T_{sl}, $\pi_c(s_i, t_i)$ must be the shortest. Let $\mathcal{ST}'$ denote the remaining pairs i.e., the descendents of (t_1, s_1) in T_{sl}. For pairs in $\mathcal{ST}'$ we also need to compute counterclockwise shortest paths and chose the shortest. Note however, that if $\pi_{cc}(s_j, t_j)$ is the shortest path for a pair $(s_j, t_j) \in \mathcal{ST}'$ then the counterclockwise path must be the shortest for any ancestor of (s_j, t_j) in T_{sl}.

We can compute counterclockwise shortest paths for pairs in $\mathcal{ST}'$ as follows. Let r be any source or destination between t_1 and s_1 and let $\pi_{cc}(r, r)$ be the shortest counterclockwise path from r to itself ($\pi_{cc}(r, r)$ wraps around Q, see Fig. 11). Let $\mathcal{ST}''$ be the subset of $\mathcal{ST}'$ that consists of pairs with sources between t_1 and r and destinations between r and s_1 i.e., $\mathcal{ST}'' = \{(s_j, t_j), t_1 < s_j \leq r \text{ and } r \leq t_j < s_1\}$. We can compute $\pi_{cc}(s_j, t_j)$ for every pair in $\mathcal{ST}''$ by computing shortest non-crossing paths in the simple polygon produced by cutting P along $\pi(s_1, t_1)$ and $\pi_{cc}(r, r)$. For the remaining pairs in $\mathcal{ST}' - \mathcal{ST}''$ let v and w be the first and last vertex respectively in ∂Q along $\pi_{cc}(r, r)$. Clearly the counterclockwise shortest path for any pair in $\mathcal{ST}' - \mathcal{ST}''$ must pass through v or w. Thus we can compute the remaining counterclockwise shortest paths using the shortest path trees from v and w to all points in $\mathcal{ST}' - \mathcal{ST}''$. Note that if $\pi_c(s_1, t_1)$ passed through ∂Q we could compute counterclockwise shortest paths for pairs in $\mathcal{ST}'$ using the shortest path trees from the first and last vertex in ∂Q along $\pi_c(s_1, t_1)$.

6. A note on the k-pairs non-crossing shortest path problem in a rectangular domain

In this section we make an observation on the algorithm of Takahashi *et al.*[12]

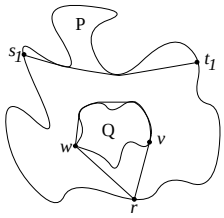

Fig. 11. Computing counterclockwise shortest paths in $\mathcal{ST}'$.

for the k pairs non-crossing shortest path problem in a domain of rectangles. All rectangles are enclosed within a single outer rectangle and source-destination pairs may appear on the outer and one inner rectangle. Distances are measured in the L_1 metric. Takahashi *et al.*[12] showed that the collection of non-crossing paths of minimum total length between the k source-destination pairs can be computed in $O((n+k)\log(n+k))$ time where n is the number of rectangles. The idea is to reduce the problem for a plane region to a problem for a series of plane graphs of $O(n+k)$ vertices in total. Our observation is that k need not play an important role in the time complexity of this algorithm. The problem for a plane region can be reduced to a problem for plane graphs of total size $O(n)$. Then a solution for $O(n)$ non-crossing paths can be modified to provide the set of non-crossing shortest paths for the original k source-destination pairs. In this manner the algorithm of Takahashi *et al.*[12] can achieve $O(k+n\log n)$ time, assuming that the source-destination pairs are given sorted around the boundary. If pairs are not sorted an additional $O(k\log k)$ time is needed initially for sorting.

A simpler problem whose solution is used throughout Ref. [12] is one where source-destination pairs appear on two parallel boundary edges in a sorted order. To solve the general problem, Takahashi *et al.*[12] perform a case analysis and define subsets of source-destination pairs for which they determine shortest non-crossing paths within subregions of the rectangular domain. Within each subregion the non-crossing shortest path problem simplifies to one where source-destination pairs appear on two parallel boundary edges. The algorithmic parts that lead to the derivation of these simpler subproblems do not depend on the number of source-destination pairs, k, and can be performed in $O(n\log n)$ time. These parts involve the computation of axis parallel paths between a constant number of boundary points and the *cutting*[a] of the domain along those paths. Thus, by modifying the algorithm of Ref. [12] for the simpler subproblem to run in $O(k+n\log n)$ time, the time complexity of the general algorithm in Ref. [12] becomes also $O(k+n\log n)$ assuming that source-destination pairs are given sorted. In the sequel we focus on this simpler problem (see Fig. 12).

Let's assume, without loss of generality, that all sources appear on the upper horizontal edge of the outer boundary and that all destinations appear on the lower horizontal edge. The outer boundary is assumed to be an axis parallel polygon

[a] *Cutting* a plane region is equivalent to *slitting* a plane graph as defined in Ref. [12]. Recall that *cutting* a region along a path corresponds to treating every segment along the path as a pair of parallel boundary edges.

which need not always be a rectangle. As with the simple polygon case, the source-destination pairs admit a collection of non-crossing paths if they appear in a certain order along the boundary. If non-crossing paths exist the solution is simply a collection of shortest paths between the given source-destination pairs. We assume that the given set of source-destination pairs, $\mathcal{ST}$, admits a set of non-crossing paths and that (s_1, t_1) appears on the boundary as the left-most pair. Note that a set of non-crossing paths exists if and only if $s_1, \ldots, s_k$ and $t_1, \ldots, t_k$ appear on the boundary from left to right in this order.

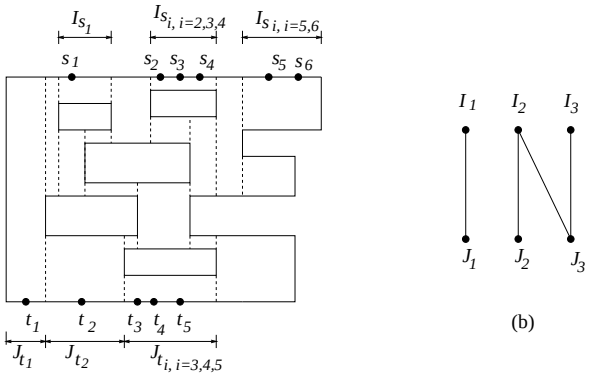

Fig. 12. The vertical decomposition of a rectangular domain and the corresponding bipartite graph.

Consider the *vertical* decomposition of the plane region obtained by drawing vertical lines through the vertical edges of the boundary (see Fig. 12). It partitions the horizontal boundary edges into intervals which are assumed to be ordered from left to right. An interval containing at least one source (resp. destination) is referred to as a *source interval* (resp. *destination interval*). Consider the bipartite graph G whose vertex set consist of *source nodes* corresponding to source intervals and *destination nodes* corresponding to destination intervals. There is an edge connecting a source node I to a destination node J if and only if there is a pair (s_i, t_i) such that s_i belongs to interval I and t_i belongs to interval J. Fig. 12b illustrates the bipartite graph for the problem in Fig. 12. Since $\mathcal{ST}$ is assumed to admit a set of non-crossing paths, the edges of G must be non-crossing.

Given a pair (s_i, t_i), let I_{s_i} and J_{t_i} be the intervals of the boundary where s_i and t_i belong respectively. Nodes I_{s_i} and J_{t_i} must be adjacent in G. If I_{s_i} and J_{t_i} bound the same rectangular region in the vertical decomposition, $\pi(s_i, t_i)$ can be computed trivially and thus we ignore this case. Otherwise, let l_{s_i} and r_{s_i} (resp. l_{t_i} and r_{t_i}) denote the left and right endpoint of I_{s_i} (resp. J_{t_i}). It is easy to see that $\pi(s_i, t_i)$ must be the shortest among the following four paths: $P_1 = \overline{s_i l_{s_i}} \cup \pi(l_{s_i}, l_{t_i}) \cup \overline{l_{t_i} t_i}$, $P_2 = \overline{s_i l_{s_i}} \cup \pi(l_{s_i}, r_{t_i}) \cup \overline{r_{t_i} t_i}$, $P_3 = \overline{s_i r_{s_i}} \cup \pi(r_{s_i}, r_{t_i}) \cup \overline{r_{t_i} t_i}$, $P_4 = \overline{s_i r_{s_i}} \cup \pi(r_{s_i}, l_{t_i}) \cup \overline{l_{t_i} t_i}$. This observation implies that no matter how many sources there are within interval I_{s_i} or how many destinations within interval J_{t_i}, the problem of finding $\pi(s, t)$ for $s \in I_{s_i}$ and $t \in J_{t_i}$ can be reduced to computing $\pi(l_{s_i}, l_{t_i}), \pi(r_{s_i}, l_{t_i}), \pi(l_{s_i}, r_{t_i})$, and $\pi(r_{s_i}, r_{t_i})$. Note that if $\pi(s_i, t_i)$ passes through

r_{s_i} (resp. l_{s_i}) then for any $s_j \in I_{s_i}, j > i$, (resp. $s_j \in I_{s_i}, j < i$), path $\pi(s_j, t_j)$ must also pass through r_{s_i} (resp. l_{s_i}). Similarly for r_{t_i} and l_{t_i}. Thus, we can define a new set of source-destination pairs, $\mathcal{ST}'$, where sources are the endpoints of intervals containing at least one source, and destinations are the endpoints of intervals containing at least one destination. In particular, $\mathcal{ST}' = \cup\{(I_{s_i}, J_{t_i}) \mid (s_i, t_i) \in \mathcal{ST}\}$, where $(I_{s_i}, J_{t_i}) = \{(l_{s_i}, l_{t_i}), (l_{s_i}, r_{t_i}), (r_{s_i}, l_{t_i}), (r_{s_i}, r_{t_i})\}$. Set $\mathcal{ST}'$ can be regarded as a set of source-destination pairs of intervals where each interval pair contributes four source-destination pairs of points. The source-destination pairs of intervals correspond to pairs of adjacent vertices in the bipartite graph G and can be regarded as pairs of super-nodes between which we need to obtain shortest paths. The size of $\mathcal{ST}'$ is $O(\min\{n, k\})$.

The paths between pairs of points in $\mathcal{ST}'$ are crossing each other, however $\mathcal{ST}'$ can be partitioned into at most four subsets of non-crossing pairs (assuming that the original set $\mathcal{ST}$ admits a set of non-crossing paths). In particular, let $\mathcal{ST}'_1 = \{(l_{s_i}, l_{t_i}) \mid (s_i, t_i) \in \mathcal{ST}\}$, $\mathcal{ST}'_2 = \{(l_{s_i}, r_{t_i}) \mid (s_i, t_i) \in \mathcal{ST}\} - \mathcal{ST}'_1$, $\mathcal{ST}'_3 = \{(r_{s_i}, l_{t_i}) \mid (s_i, t_i) \in \mathcal{ST}\} - \{\mathcal{ST}'_1 \cup \mathcal{ST}'_2\}$, and $\mathcal{ST}'_4 = \{(r_{s_i}, r_{t_i}) \mid (s_i, t_i) \in \mathcal{ST}\} - \{\mathcal{ST}'_1 \cup \mathcal{ST}'_2 \cup \mathcal{ST}'_3\}$. Since the size of each subset is $O(\min\{n, k\})$, shortest paths can be computed in $O(n \log n)$ time using the algorithm of Takahashi *et al.*[12] Alternatively, instead of repeating the algorithm[12] for each subset, we can slightly modify the divide and conquer technique of Ref. [12] by using the bipartite graph G as a guide. Let I_p be the middle source interval i.e., $p = \lfloor m/2 \rfloor$ where m is the number of source intervals. Let J_q be the left-most destination interval which is adjacent to I_p in G. We first compute $\pi(I_p, J_q) = \{\pi(l_p, l_q), \pi(r_p, l_q), \pi(l_p, r_q), \pi(r_p, r_q)\}$, where l_p, r_p, l_q and r_q denote the left and right endpoints of I_p and I_q respectively. Assuming that all horizontal line segments of rectangles have been pre-sorted in a list $\mathcal{L}$ in descending order,[12] each path in $\pi(I_p, J_q)$ can be computed in $O(n)$ time by procedure SHORTESTPATH of Ref. [12]. Note that J_q is the left-most destination interval adjacent to I_p and thus, no shortest path for $\mathcal{ST}$, other than $\pi(s_j, t_j)$, $s_j \in I_p$ and $t_j \in J_q$, need to intersect $\pi(l_p, r_q)$. Thus, we can divide the plane region into two subregions by *cutting* the region along $\pi(l_p, r_q)$. Then we recursively compute shortest paths for interval pairs with sources to the left of I_p in the left subregion, and paths for interval pairs with sources to the right of I_p (including I_p if I_p is adjacent to J_{q+1} in G) in the left subregion. In this manner, the depth of the recursive calls is $O(\log m)$ and the time required for all recursive calls at the same depth is $O(n)$. (Note that $\mathcal{L}$ can be updated for the two cut subregions in $O(n)$ time.[12]) Thus, the modified recursive algorithm takes $O(n \log n)$ time in total.

Now for each pair $(s_i, t_i) \in \mathcal{ST}$ we simply need to select the shortest among paths P_1, P_2, P_3 and P_4. Assuming that ties are broken in a consistent way when computing shortest paths, the resulting collection of shortest paths for $\mathcal{ST}$ must be non-crossing. (Note that crossing paths between pairs in $\mathcal{ST}$ must cross at least twice. Thus, the only way for shortest paths between two pairs to be crossing is to have subpaths of equal length between the crossing points). Thus, the solution for shortest paths in $\mathcal{ST}'$ can be easily modified to derive the set of non-crossing paths for the original set $\mathcal{ST}$ in additional $O(k)$ time, resulting in an $O(k + n \log n)$-time

algorithm.

7. Conclusion

We have given a simple linear time algorithm for the k-pairs non-crossing shortest path problem in a simple polygon. The algorithm is easy to implement and can be used to obtain useful geodesic decompositions of simple polygons by including in the decomposition shortest paths between pairs of boundary points. An example is the *balanced geodesic decomposition*.[2,3] Similarly to Takahashi *et al.*,[11,12] this result can be extended to a simple polygon with one hole where source-destination pairs may appear on both the outer and the inner boundary. We also made an observation for the k-pairs non-crossing shortest path problem in a rectangular polygonal domain, as defined in Ref. [12], that k need not play a main role in the algorithm other than initial sorting.

Acknowledgments

The author wishes to thank professor D.T. Lee for valuable discussions and comments.

References

1. B. Chazelle, "A theorem on polygon cutting with applications," *Proc. 23rd Annu. IEEE Sympos. Found. Comput. Sci.*, 1982, pp. 339–349.
2. B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink, "Ray shooting in polygons using geodesic triangulations", *Algorithmica*, 12, 1994, 54-68.
3. M. T. Goodrich and R. Tamassia, "Dynamic ray shooting and shortest paths via balanced geodesic triangulations", *In Proc. 9th Annu. ACM Sympos. Comput. Geom*, 1993, 318-327.
4. L.J. Guibas and J. Hershberger, "Optimal shortest path queries in a simple polygon", *J. Comput. Syst. Sci.*, 39, 1989, 126-152.
5. L.J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R.E. Tarjan, "Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons". *Algorithmica*, 2, 209-233, 1987.
6. J. Hershberger and J. Snoeyink, "Computing minimum length paths of a given homotopy class", *Comput. Geometry: Theory and Applications*, 4, 1994, 63-97.
7. D.T. Lee, "Non-crossing path problems", Manuscript, Dept. of EECS, Northwestern University, 1991.
8. D. T. Lee and F. P. Preparata, "Euclidean shortest paths in the presence of rectilinear barriers", *Networks*, 14 1984, 393-410.
9. M.H. Overmars and J. van Leeuwen, "Maintenance of configurations in the plane", *Journal of Computer and System Science* 23 1981, 166-204.
10. F.P. Preparata and M. I. Shamos, *Computational Geometry: an Introduction*, Springer-Verlag, New York, NY 1985.
11. J. Takahashi, H. Suzuki, and T. Nishizeki, "Shortest non-crossing paths in plane graphs", *Algorithmica*, 16(3) 1996, 339-357.

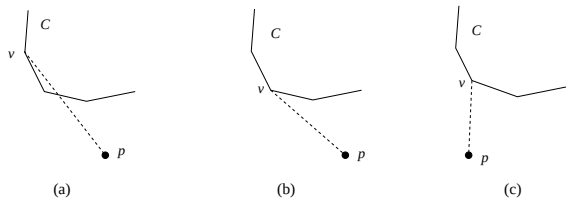

Fig. A.1. (a) v is concave. (b) v is supporting. (c) v is reflex.

12. J. Takahashi, H. Suzuki, and T. Nishizeki, "Shortest non-crossing rectilinear paths in plane regions", *Int. J. of Computational Geometry and Applications*, Vol. 7, No. 5, 1997, 419-436.

Appendix A:

In the following, the algorithms to determine segment $\overline{yz}$ and update the geodesic polygon g are given in detail. A segment $\overline{uv}$, where $u, v \in \partial P$, is said to be *valid* if it lies entirely within P.

Let's first give some useful terminology.[10] Given a convex chain C, a vertex $v \in C$ and a point $p \notin C$, we say that vertex v is *concave* (with respect to segment $\overline{pv}$) if segment $\overline{pv}$ intersects the polygon derived by drawing the segment joining the endpoints of C (see Fig. A.1(a)). Otherwise, if the line containing $\overline{pv}$ is tangent to C (i.e., the two vertices adjacent to v lie on the same side of the line), we say that v is *supporting* (with respect to $\overline{pv}$) (see Fig. A.1(b)). Otherwise, we say that v is *reflex* (with respect to $\overline{pv}$) (see Fig. A.1(c)).

Algorithm Determine-$\overline{yz}$
Input: The geodesic decomposition γ and α_i.
Output: Vertices y, z, and the case id $(1 \ldots 4)$.
Begin

(i) Initialize v and u to α_i.

(ii) Check for case (1) as follows: (Recall that $v \in \pi(\alpha_i, b)$ and $u \in \pi(\alpha_i, c)$.) While either v or u is reflex with respect to $\overline{uv}$, and segment $\overline{uv}$ is valid do: (To determine whether $\overline{uv}$ is valid, discriminate $pred(v)$ and $pred(u)$ against $\overline{uv}$. Segment $\overline{uv}$ is valid if and only if both $pred(v)$ and $pred(u)$ lie to the opposite side of $\overline{uv}$ as α_i.)

(a) Advance v along $\pi(\alpha_i, \alpha_j)$ until v becomes supporting or $\overline{uv}$ becomes invalid. (Note that v cannot advance beyond b in this step. For $v = b$, v must be supporting with respect to $\overline{vpred(v)}$ (property 1), and thus v must be supporting with respect to any valid $\overline{uv}$ with $u \in \pi(\alpha_i, \alpha_l)$. Note also that v cannot advance beyond $a' = nca(r_i, r_l)$, since for $v = nca(r_i, r_l)$, v must be supporting with respect to any valid $\overline{uv}$ with $u \in \pi(\alpha_i, \alpha_l)$ (see Fig. 7).)

(b) Advance u along $\pi(\alpha_i, \alpha_l)$ until u becomes supporting or $\overline{uv}$ becomes invalid. (Note that u cannot advance beyond c or a' in this step).

If segment $\overline{uv}$ is valid we have case (1); return. (Note that if segment $\overline{uv}$ is valid both v and u must be supporting, otherwise the previous step would not

have terminated.)

Otherwise, if $\overline{uv}$ is invalid because $pred(v)$ falls to the same side of $\overline{uv}$ as α_i i.e., $\overline{uv}$ intersects $\pi(a,b)$, we have case (2) or case (4). Determine b by advancing v along $\pi(\alpha_i,\alpha_j)$ until v becomes supporting with respect to $\overline{vpred(v)}$; let $v=b$; go to step (iii).

Otherwise, segment $\overline{uv}$ must be invalid because $pred(u)$ falls to the same side of $\overline{uv}$ as α_i i.e., $\overline{uv}$ intersects $\pi(a,c)$. We have case (3) or case (4). Determine c by advancing u along $\pi(\alpha_i,\alpha_l)$ until u becomes supporting with respect to $\overline{upred(u)}$; let $u=c$; go to step (iv).

(iii) Check for case (2) as follows: (Recall that $v\in\pi(a,b)$ and $u\in\pi(\alpha_i,c)$).
While v is concave or u is reflex, and $\overline{uv}$ does not intersect $\pi(a,c)$ do: (Note that segment $\overline{uv}$ intersects $\pi(a,c)$ if and only if $pred(u)$ is to the same side of $\overline{uv}$ as α_i.)

- (a) Advance v along $\pi(b,\alpha(g))$ until v becomes supporting. (Note that v starts as concave).
- (b) Advance u along $\pi(\alpha_i,\alpha_l)$ until either u becomes supporting, or v becomes concave, or $\overline{uv}$ intersects $\pi(a,c)$.

If segment $\overline{uv}$ is valid we have case (2); return.

Otherwise we have case (4). In case (4), determine vertex c by advancing u along $\pi(\alpha_i,\alpha_l)$ until u becomes supporting with respect to $\overline{upred(u)}$; let $u=c$; go to step (v).

(iv) Check for case (3) by advancing segment $\overline{uv}$ similarly to case (2). (Recall that $v\in\pi(\alpha_i,b)$ and $u\in\pi(a,c)$). If at the end of the advancement segment $\overline{uv}$ is valid we have case (3); return.

Otherwise we have case (4). In case (4), determine b by advancing v along $\pi(\alpha_i,\alpha_j)$ until v becomes supporting with respect to $\overline{vpred(v)}$; let $v=b$; go to step (v).

(v) Case (4): Advance v along $\pi(b,a)$ and u along $\pi(c,a)$ until both v and u become supporting. Deciding whether to advance v or u at each step is done as in the U-hull construction of Overmars and van Leeuwen[9] (see page 128 of Ref. [10]). In particular do the following:

Let l be a line containing vertex $a=nca(r_j,r_l)$ with slope between the slopes of edges along $\pi(a,b)$ and $\pi(a,c)$ that are incident to a. Let l be oriented away from a.

While v or u is concave do

- (a) Let l_1 be the line containing v and its successor along $\pi(b,a)$ and let l_2 be the line containing u and its successor along $\pi(c,a)$. Let I be the intersection point of l_1 and l_2.
- (b) If I is to the right of l then u has not reached its final position because u must be concave with respect to $\overline{uv}$ for any $v\in\pi(b,a)$; advance u.
- (c) If I is to the left of l then v has not reached its final position because v must be concave with respect to $\overline{uv}$ for every $u\in\pi(c,a)$; advance v. (See page 128 of Ref. [10] for more details).

End

The following algorithm updates a geodesic polygon g according to path $\pi \subseteq \pi(r_j, r_l)$ (in case 4, $\pi = \pi_1$ and $\pi = \pi_2 \cup \overline{yz}$). For the definition of π, π_1, and π_2, see section 4. Let $y_1, \ldots, y_m$ denote the vertices along π (resp. π_1 and $\pi_2 \cup \overline{yz}$ for case 4) where y_1 is the starting point of π i.e., $y_1 = b$ or $y_1 = c$ in cases 2,3,4. Fig. 9 shows the update of g for case 3, where $\pi = \pi(c, y)$. The shaded parts are the geodesic subpolygons of g produced by the update.

Algorithm Update-γ

Input: The geodesic decomposition γ, vertices y, z, path $\pi = y_1 \ldots y_m$, and the case id $(1 \ldots 4)$.

Output: The updated geodesic decomposition.

Begin

Let $i = 1$. While $i < m$ do

(i) If $\overline{y_i y_{i+1}} \neq \overline{yz}$ and $\overline{y_i y_{i+1}}$ is not a boundary edge of g then $\overline{y_i y_{i+1}}$ introduces a geodesic subpolygon whose main apex is y_{i+1} (e.g., edges $\overline{y_1 y_2}$ and $\overline{y_3 y_4}$ in Fig. 9). Edge $\overline{y_i y_{i+1}}$ is white.

(a) If $y_i < y_{i+1}$ (i.e., y_{i+1} follows y_i in a clockwise traversal of ∂P starting at s_1), let $\mathit{old\text{-}next}(y_{i+1}) = \mathit{next}(y_{i+1})$ and $\mathit{next}(y_{i+1}) = y_i$. Otherwise, let $\mathit{old\text{-}prev}(y_{i+1}) = \mathit{prev}(y_{i+1})$ and $\mathit{prev}(y_{i+1}) = y_i$. (Recall that geodesic polygons are assumed to be ordered clockwise.)

(b) If y_i has become the main apex of a geodesic polygon then y_i corresponds to a geodesic link of a single vertex. For consistency, we represent this link using a dummy node y_i'. Let $\mathit{prev}(y_i') = \mathit{prev}(y_i)$, $\mathit{next}(y_i') = \mathit{next}(y_i)$, $\mathit{plink}(y_i') = y_i$, and $\mathit{nlink}(y_i) = y_i'$. If $y_i < y_{i+1}$, let $\mathit{next}(y_i) = \mathit{old\text{-}next}(y_i)$. If $y_i > y_{i+1}$, let $\mathit{prev}(y_i) = \mathit{old\text{-}prev}(y_i)$.

(c) If $y_i < y_{i+1}$, let $\mathit{prev}(y_i) = y_{i+1}$. Otherwise, let $\mathit{next}(y_i) = y_{i+1}$.

(ii) If $\overline{y_i y_{i+1}}$ is a boundary edge of g then $\overline{y_i y_{i+1}}$ becomes part of a geodesic link (e.g., edge $\overline{y_2 y_3}$ in Fig. 9). Edge $\overline{y_i y_{i+1}}$ may be either red or white. If $y_i = \alpha_j$ or $y_i = \alpha_l$ then update the apexes of r_j or r_l respectively to y_{i+1} and add edge $\overline{y_i y_{i+1}}$ to the output $\mathcal{E}$. Otherwise, let $\mathit{nlink}(y_{i+1}) = y_i$ and $\mathit{plink}(y_i) = y_{i+1}$.

(iii) If $\overline{y_i y_{i+1}}$ is segment $\overline{yz}$ (e.g., segment $\overline{y_4 y_5}$ in Fig. 9) do the following. If vertex y or vertex z has become the main apex of a geodesic polygon, introduce a dummy node as explained in step (b). Let $\mathit{next}(y) = z$ and $\mathit{prev}(z) = y$. Edge $\overline{yz}$ is white.

End