Computing Homotopic Shortest Paths in the Plane

Sergei Bespamyatnikh

Department of Computer Science, University of Texas at Dallas, Box 830688, Richardson, TX 75083, USA.

Abstract

We address the problem of computing homotopic shortest paths in the presence of obstacles in the plane. Problems on homotopy of paths received attention very recently [5,11]. We present two output-sensitive algorithms, for simple paths and non-simple paths. The algorithm for simple paths improves the previous algorithm [11]. The algorithm for non-simple paths achieves $O(\log^2 n)$ time per output vertex which is an improvement by a factor of $O(n/\log^2 n)$ of the previous algorithm [14], where n is the number of obstacles. The running time has an overhead $O(n^{2+\varepsilon})$ for any positive constant ε. In the case $k < n^{2+\varepsilon}$, where k is the total size of the input and output, we improve the running to $O((n + k + (nk)^{2/3}) \log^{O(1)} n)$.

Key words: Shortest path, homotopy, output-sensitive algorithm.
PACS:

1 Introduction.

Finding shortest paths in a geometric domain is a fundamental problem [20]. Chazelle [6] and Lee and Preparata [15] gave a funnel algorithm that computes the shortest path between two points in a simple polygon. Hershberger and Snoeyink [14] simplify the funnel algorithm and studied various optimizations of a given path among obstacles under the Euclidean and link metrics and under polygonal convex distance functions. The funnel algorithm has been extended in addressing a classic VLSI problem, the continuous homotopic routing problem [9,16,12]. For this problem it is required to route wires with fixed terminals among

Email address: `besp@utd.edu` (Sergei Bespamyatnikh).
URL: `http://www.utd.edu/~besp` (Sergei Bespamyatnikh).

fixed obstacles when a sketch of the wires is given, i.e., each wire is given a specified homotopy class. If the wiring sketch is not given or the terminals are not fixed, the problem is NP-hard [17,22,23].

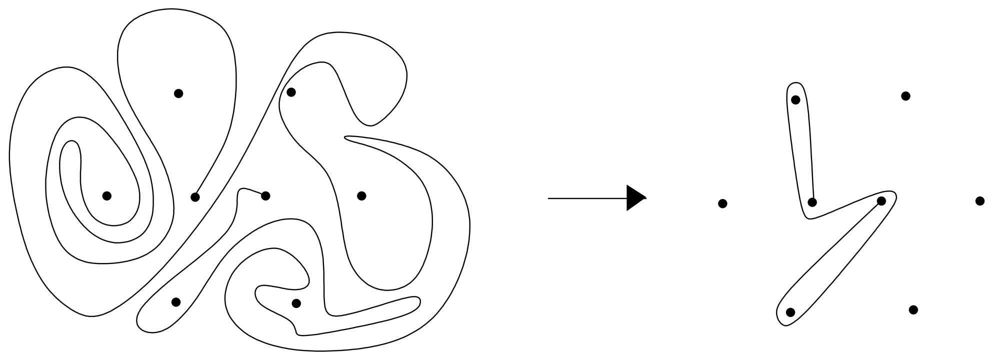


Fig. 1. Shortest path preserving homotopy type.

The topological concept of homotopy captures the notion of deforming paths [1,21]. A *path* is a continuous map $\pi : [0,1] \rightarrow \mathbb{R}^2$. Let $\alpha, \beta : [0,1] \rightarrow \mathbb{R}^2$ be two paths that share starting and ending endpoints, $\alpha(0) = \beta(0)$ and $\alpha(1) = \beta(1)$. Let $B \subset \mathbb{R}^2$ be a set of *barriers* such that the paths α and β avoid B. The paths α and β are *homotopic* with respect to the barrier set B if α can be continuously transformed into β avoiding B; more formally, if there exists a continuous function $\Gamma : [0,1] \times [0,1] \rightarrow \mathbb{R}^2$ with the following three properties:

(1) $\Gamma(0,t) = \alpha(t)$ and $\Gamma(1,t) = \beta(t)$ for $0 \leq t \leq 1$,
(2) $\Gamma(s,0) = \alpha(0) = \beta(0)$ and $\Gamma(s,1) = \alpha(1) = \beta(1)$ for $0 \leq s \leq 1$,
(3) $\Gamma(s,t) \notin B$ for $0 \leq s \leq 1$ and $0 < t < 1$.

The problem can be stated as follows.

Shortest Homotopic Path (SHP) Problem. Given n_p disjoint paths and n_b point barriers, find the shortest homotopic paths. We assume that the endpoints of the paths are barriers as well. Let $n = 2n_p + n_b$ be the total number of barriers. Let k_{in} (resp. k_{out}) be the total number of edges of the input paths (resp. output paths) and let $k = k_{in} + k_{out}$.

Hershberger and Snoeyink [14] gave a $O(nk_{in})$ algorithm for one path, $n_p = 1$, which is optimal in the worst case assuming that the running time is evaluated in terms of input parameters n and k_{in}. Recently Cabello *et al.* [5] studied the problem of testing whether two paths with common endpoints in the presence of obstacles are homotopic. They mentioned that computing shortest homotopic paths is expensive for testing homotopy since the shortest path can have $\Omega(nk_{in})$ edges which is quadratic in terms of the input size $O(n + k_{in})$ in the worst case. The key idea of the algorithm is to compute *canonical* paths that can be found by rectifying the paths and shortcutting them using *segment dragging* by Chazelle [7].

Very recently Efrat *et al.* [11] presented an output sensitive algorithm for computing the shortest homotopic paths. The algorithm runs in $O(n^{3/2} + k_{in} \log n + k_{out})$ time. In the first phase of the algorithm they apply shortcuts to compute x-monotone paths using an efficient algorithm for computing canonical paths [5]. The computation of the canonical paths is based on segment dragging [7,5]. The monotone paths are then bundled into $O(n)$ groups of homotopically identical paths. Then, the deterministic algorithm for computing shortest homotopic paths is based on recursive partition of the paths and routing procedure applied to the largest increasing (decreasing) sequence of paths. The running time of this phase is $O(n^{3/2} + k)$. Efrat *et al.* [11] also show that this bound can be improved to $O(n \log n + k)$ using randomization.

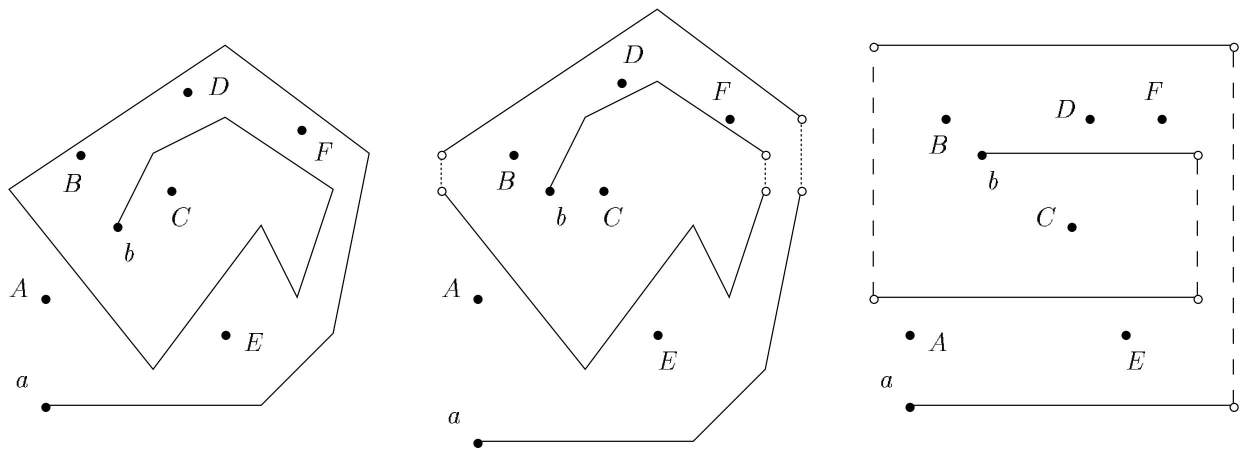
We focus on two versions of SHP problem: (1) simple input paths and (2) non-simple paths. For simple paths we show that the shortest homotopic paths can be computed in $O(n \log^{1+\varepsilon} n + k_{in} \log n + k_{out})$ time where $\varepsilon > 0$ is an arbitrarily small constant. For non-simple paths we design an *implicit funnel* algorithm that computes the shortest paths in $O(n^{2+\varepsilon} + k \log^2 n)$ time. To the best of our knowledge, this is the first output-sensitive algorithm for general paths. Note that the constants hidden in the notation depend on ε. We also show that for relatively small k the running time can be improved to $O((n + n^{2/3}k^{2/3} + k)\text{polylog}(n))$ where $\text{polylog}(x)$ denotes a polylogarithmic function $\log^{O(1)}(x)$. The improvement is based on hierarchical cuttings [19] and their space-time tradeoff. Note that if $k = \Theta(n)$ the algorithm runs in $O(n^{4/3}\text{polylog}(n))$ time. This matches up to a polylogarithmic factor to a lower bound of $\Omega(n^{4/3})$ obtained from testing homotopy [5].

The paper is organized as follows. We consider SHP for simple paths in Sections 2-4. In Section 2 we reduce the problem to the case of monotone paths by applying techniques from [5,11]. In Section 3 we show how to compute the shortest path in a simple polygon with barriers in linear time. In Section 4 we develop an algorithm for computing all the shortest paths. In Section 5 we consider SHP for non-simple paths. In Section 6 we discuss the algorithm for non-simple paths when k is relatively small.

2 Reduction to Monotone Paths.

In this Section we briefly describe the construction of the canonical rectified paths [5] and the bundling [11]. As in [11] we use the canonical paths to shortcut the given path and divide it into *x-monotone* paths, i.e., the paths that are monotone with respect to the direction OX. A path is *monotone* with respect to a direction d if any line orthogonal to d intersects the path at most once. One can partition a path into x-monotone pieces by exploring its vertices locally, i.e. we break the path at a vertex if two edges incident to the vertex do not form a x-monotone path as illustrated in Fig. 2 (b) (this partition might not capture the shape of the shortest path, see Fig. 1). We can treat these monotone paths as horizontal segments and obtain *rectified paths* [5], see Fig. 2 (c).

To rectify the paths one needs an “aboveness” relation between the monotone paths and the



(a) (b) (c)

Fig. 2. (a) Path ab, (b) Monotone paths, (c) Rectified paths.

barriers. A path is represented as a sequence of points that it passes above (overbar) and below (underbar). For example, the path ab depicted in Fig. 2 is recorded as the sequence $ab = \overline{ABCDEF}\underline{FEDCBAA}\overline{BCD}\underline{E}\overline{FF}\underline{E}\overline{D}\underline{C}$. An adjacent pair of repeated symbols can be deleted by deforming the path without changing the homotopy class. The deletion can be repeated until we obtain *canonical sequence*. The path shown in Fig. 2 has the canonical sequence $ab = \overline{ABCDEF}\underline{FEDCBA}\overline{BC}\underline{C}$, see Fig. 3.

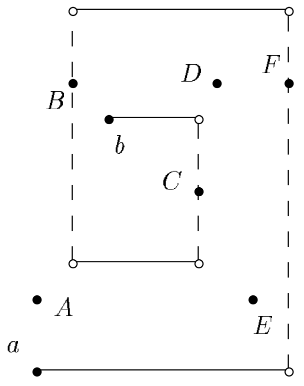


Fig. 3. Rectified canonical path.

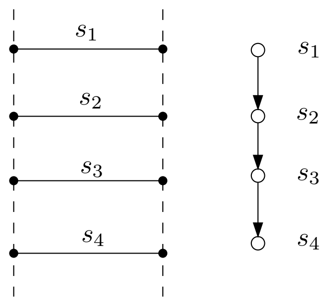


Fig. 4. Bundling homotopically equivalent segments $s_1, \ldots, s_4$. Segment $s_i, i > 1$ is the only child in the aboveness tree.

In order to generate the rectified paths we compute a triangulation or a trapezoidation [11] using an algorithm by Bar-Yehuda and Chazelle [2]. The running time of the triangulation algorithm is $O(n \log^{1+\varepsilon} n + k_{in})$ for any fixed $\varepsilon > 0$. The monotone paths induce an "aboveness" relation that is acyclic. The rectified paths and new positions of barriers are computed using their ranks as y-coordinates [5]. The rectified paths can be shortcut by vertical segments producing canonical rectified paths. Applying the segment dragging queries by Chazelle [7] shortcuts can be done in $O(k_{in} \log n)$ time using $O(n \log n)$ preprocessing time and $O(n)$ space.

The number of homotopically different paths produced by shortcutting is at most $2n$ [11]. The homotopic paths can be bundled reducing the problem to the case $n_p \leq 2n$. To bundle the paths we order the paths and the barrier points again. We even rectify the paths so that now each path in the rectified model is represented by a horizontal segment. For this we map the canonical paths back to the real paths, truncate them and retriangulate. We rank the paths and some barriers (not all!) – we exclude barriers that are path endpoints. Note that all path endpoints are the barriers, for example the paths in Fig 3 are aF, BF, BC and bC. We compute the bundles using the property that the ranks of paths homotopic to a path form a sequence of numbers $r, r+1, \ldots, r'$ for some integers r and r', see Fig. 4. The property follows from the fact that the homotopically equivalent segments form a chain in the aboveness tree [5] and the ranks are obtained from the inorder traversal of the tree. In Sections 3 and 4 we use this ranking and assume that $n_p \leq 2n$.

3 Shortest Monotone Paths.

We show how to compute the shortest path of a x-monotone path. The barrier points that are above or below the path are the only barrier points that may affect the shortest path. We reduce the problem of constructing the shortest homotopic paths to a problem of finding a shortest monotone path in a simple polygon with colored barriers so that the barriers are separated by the path according to their colors. The problem can be defined as follows.

Shortest monotone path. Let P be a simple x-monotone polygon[1] and let s, t be two points in P. Let S be a finite set of points (barriers) inside P so that each point is colored red or blue. We assume that all the points $P \cup S \cup \{s, t\}$ have distinct x-coordinates. Find the shortest path from s to t in P so that every point of S above (resp. below) the path is red (resp. blue).

In general, if the x-order of barriers is unknown, one can expect the worst case complexity $O(|S| \log |S| + |P|)$. We assume that the barriers are sorted.

Lemma 1 *The problem above can be solved in linear time assuming that the barriers are sorted by x-coordinates.*

Proof: We connect the barrier points inside P to the boundary of P according to the colors, see Fig. 5 for example. We draw vertical segments passing through the points s and t. We obtain a polygon that can be slightly perturbed to make a simple polygon. The segment endpoints on the boundary of P can be found by a simultaneous scan of the vertices of P and the barriers in the x-order. The shortest path in a simple polygon can be found in linear time [13,14]. Clearly, the produced path satisfies the color constraint. ■

[1] A polygon is x-monotone if its boundary is the union of two x-monotone paths.

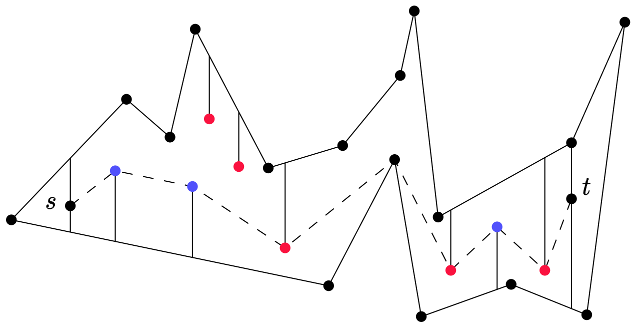


Fig. 5. Shortest path separating red and blue points.

It is straightforward to modify the algorithm above so that the general case where a red point and a blue point coincide. We can interpret them as points infinitesimally close to each other (the red point above the blue one). The case where many points have the same x-coordinate can be handled as well. The only constraint is that a blue point cannot be above any red point.

4 Shortest Paths.

We apply a divide-and-conquer approach based on the efficient algorithm from the previous section. Since the paths are given in the order according to the ranks we divide the problem into subproblems with approximately half of the paths. In what follows we discuss how to generate polygons, color barriers and compute the shortest paths efficiently.

Let $\Pi = \{\pi_1, \pi_2, \ldots\}$ be the set of monotone paths produced by shortcuts and bundling. The number of paths is $O(n)$ and we assume that their order is consistent with the ranking and aboveness relation. We even use above/below relation between paths according to the ranking. We assume that π_i is above π_{i+1}. Let $\overline{B}$ be the set of barriers that are not path endpoints. Let $\texttt{rank}(\pi_i)$ denote the rank of a path $\pi_i \in \Pi$. Also let $\texttt{rank}(p)$ denote the rank of a point $p \in \overline{B}$. For any barrier point p we define $\texttt{rank}_{\text{min}}(p)$ and $\texttt{rank}_{\text{max}}(p)$ as

$$\texttt{rank}_{\text{min}}(p) = \min\{\texttt{rank}(\pi_i) \mid \pi_i \in P \text{ and } p \text{ is an endpoint of } \pi_i\}$$
$$\texttt{rank}_{\text{max}}(p) = \max\{\texttt{rank}(\pi_i) \mid \pi_i \in P \text{ and } p \text{ is an endpoint of } \pi_i\}.$$

We start with the path π_i that is the median path of Π. Let s and t denote its start and target points. The shortest path corresponding to π_i can be constructed using Lemma 1 as follows. Let R be a rectangle such that all data points are contained inside R. It can be defined by enlarging the smallest bounding box of B and Π. We pick two points a and b slightly off R to make x-monotone polygon $P = conv(R \cup \{a, b\})$, see Fig. 6. Lemma 1 can be applied to the polygon P and the shortest path corresponding to π_i can be found. The barriers can be colored using ranks as follows.

Coloring Rule. A barrier point $p \in \overline{B}$ is colored red (resp. blue) if $\texttt{rank}(p) < \texttt{rank}(\pi_i)$ (resp. $\texttt{rank}(p) > \texttt{rank}(\pi_i)$). A barrier point $p \in B \setminus \overline{B}$ is colored red (resp. blue) if $\texttt{rank}_{\min}(p) < \texttt{rank}(\pi_i)$ (resp. $\texttt{rank}_{\max}(p) > \texttt{rank}(\pi_i)$). Note that, if $\texttt{rank}_{\min}(p) < \texttt{rank}(\pi_i) < \texttt{rank}_{\max}(p)$, the point p is colored in both red and blue colors.

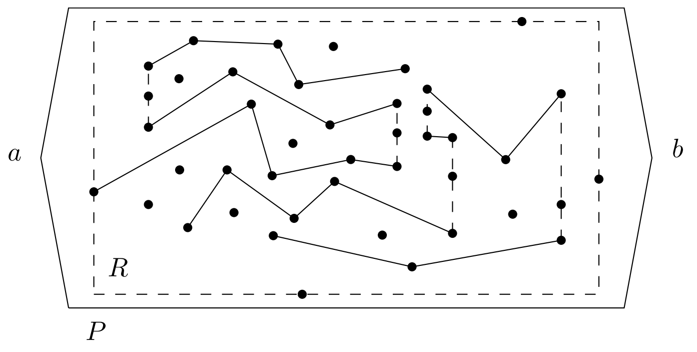


Fig. 6. Polygon P.

4.1 Splitting a Polygon.

In order to apply the algorithm from Lemma 1 recursively, the polygon P needs to be divided into two polygons. One possible idea is to connect the points a and b. If we apply the algorithm to find the shortest path ab among the barriers colored according to π_i, it will not necessarily contain the shortest path st, see Fig. 7. However we can still use this approach recursively and all paths produced in this way do not cross the path st. One can prove that the missing edges of st form at most two subpaths of st (in head and tail). These missing pieces can be found later.

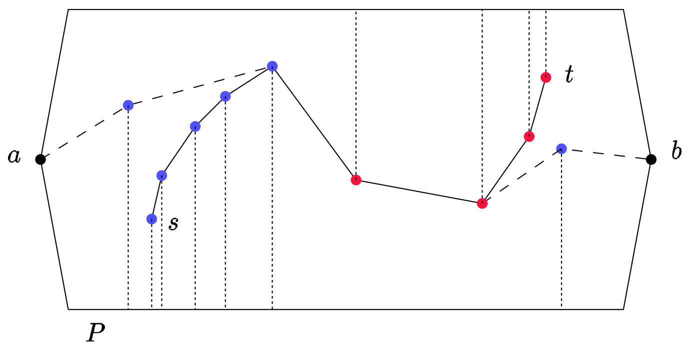


Fig. 7. The shortest paths ab and st.

We avoid the problem of fixing the paths. Instead we apply the algorithm three times and compute the shortest paths as, st, and tb. Notice that the vertices s and t contribute to both subproblems since they participate in the boundaries of two produced polygons. We call this procedure $\texttt{Split}(P, s, t)$. We have to be careful in defining the extensions as and tb so that they respect the vertical order of the paths $\pi_1, \pi_2, \ldots$ (i.e., if π_i is above (below) π_j then the extension of π_i is above (below) π_j. The same holds for relations between paths and barriers). Recall that the order is generated using the rectified paths. The rectified path model and the ranking of the rectified paths and the barriers is a nice (not just visual) tool to explain the order constraint and the partition of the barriers. We can double the rectified path endpoints and the barrier points that determine the vertical segments, see Fig. 8 (b) (note that a barrier point can support many vertical segments as the point E in Fig. 8 (a)). Then the extensions and corresponding partition of the points is naturally defined, see Fig. 8 (c).

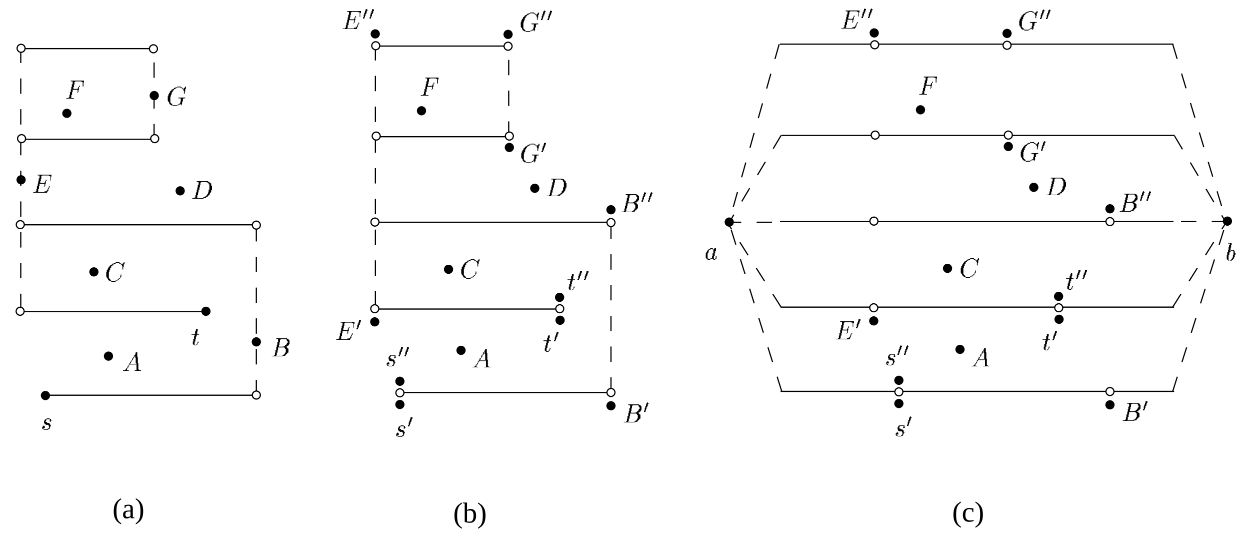


Fig. 8. (a) Rectified paths, (b) Doubling the path endpoints s, t and the support points B, E, G, (c) Extensions of the horizontal segments to ab.

Lemma 2 *The extension astb is correct, i.e., the shortest paths as and tb do not cross the shortest paths of* $\pi_1, \pi_2, \ldots$.

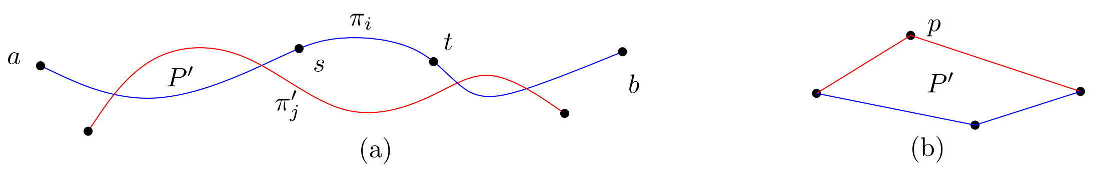


Fig. 9. The path π'_1 crosses the path $astb$. The convex vertex p of the polygon P'.

Proof: Suppose to the contrary that a path, say π_j, has the shortest path π'_j that crosses the shortest path as, the extension of a path π_i. Without loss of generality we assume that $\texttt{rank}(\pi_i) < \texttt{rank}(\pi_j)$. The endpoints of π_j cannot be above $astb$ by the definition of the extension as and the coloring rule. Then a cross point of as and π'_j is a vertex of a polygon P', see Fig. 9 (a). There are only two vertices of P' where π'_j and as cross, i.e., the leftmost vertex and the rightmost vertex.

The polygon P' has at least three convex vertices and at least one of them is not a cross point. Suppose that it is a point $p \in \pi'_j$, see Fig. 9 (b). The point p is the barrier point below π'_j (it is colored blue when the shortest path π_j found). Therefore p is blue when the shortest path as found. This contradicts the algorithm of Lemma 1. The case where the lower chain as of P' has a convex vertex is similar. The lemma follows. ∎

There is another potential problem. After we construct the path $astb$ and divide the current polygon by the path, the resulting polygons might be not simple, see Fig. 10 (a). The polygons on either side of the path $astb$ have the property that they are x-monotone and their projections on the x-axis are disjoint. Note that many polygons can be generated if the current polygon has red points on the upper side and blue points on the lower side (especially if they alternate). Our algorithm can deal with multiple polygons and divide them recursively. To do this efficiently we build the following data structure.

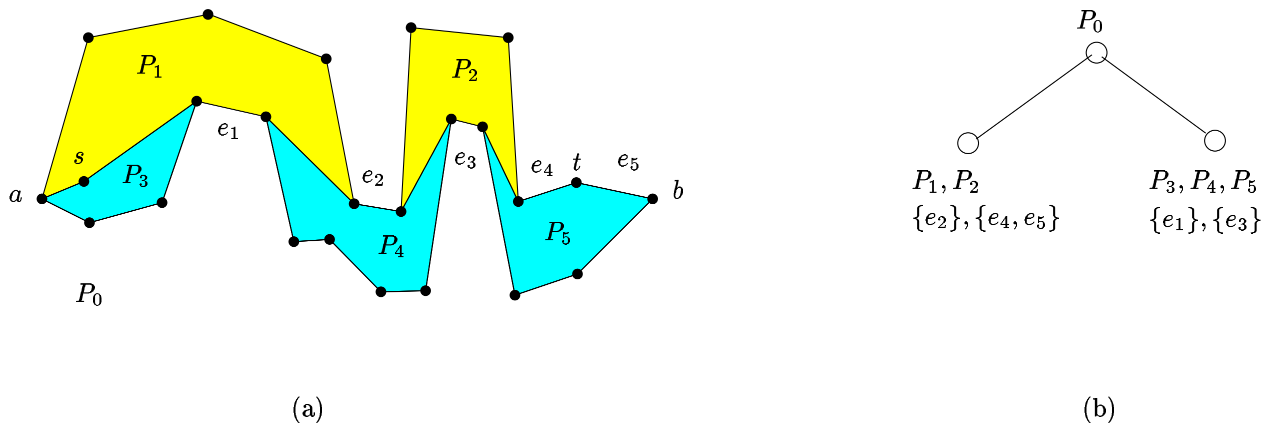


Fig. 10. (a) Multiple polygons. (b) Tree representation.

4.2 Data Structure.

Let T be a binary search tree of height $O(\log n)$ whose nodes correspond to the paths $\pi_1, \pi_2, \ldots$. The tree T is defined recursively: the root has the median path π_i and the subtrees of its children are constructed for the sets $\{\pi_j, j < i\}$ and $\{\pi_j, j > i\}$. At every node $v \in T$ we store a pointer $path(v)$ of its path. We associate the set of polygons with a node v of T denoted by $\rho_1(v), \rho_2(v), \ldots$ in the sorted order of x-projections. The root has one polygon P associated with it $\rho_1(v_{\text{root}}) = P$, see Fig. 6. With each polygon produced by `Split` we store its side, "top" or "bottom", that is generated. For example, the polygons P_1, P_2 in Fig. 10 (a) are marked "bottom" and P_3, P_4, P_5 are marked "top".

The polygons associated with the children of v are constructed by splitting the polygons $\rho_1(v), \rho_2(v), \ldots$ using the path $astb$ where s and t are endpoints of the path $path(v)$. Let $a_i(v)$ and $b_i(v)$ be the leftmost and rightmost vertices of a polygon $\rho_i(v)$. First we locate s

and t in the intervals $[a_i(v), b_i(v)]$. If they fall into the same interval, say $[a_i(v), b_i(v)]$, we apply the procedure $\texttt{Split}(\rho_i(v), s, t)$. If they fall into different intervals, say $s \in [a_i(v), b_i(v)]$ and $t \in [a_j(v), b_j(v)]$ then we split i-th and j-th polygon by calling $\texttt{Split}(\rho_i(v), s, b_i(v))$ and $\texttt{Split}(\rho_j(v), a_i(v), t)$. If one of the points s or t (or both) falls out the intervals we do not split polygons using this point (as a parameter in $\texttt{Split}()$). For each polygon $\rho_j(v)$ that misses both s and t we call $\texttt{Split}(\rho_j(v), a_j(v), b_j(v))$.

The barrier points. With each node v of T we store a barrier set $B(v)$. The barriers $B(v)$ are colored using the coloring rule when v is processed. The set $B(v)$ is defined as follows. The root has all the barriers, $B(v_{root}) = B$. Let π_i be a path stored at a node v. The barrier points of $\overline{B} \cap B(v)$ are partitioned into two sets and assigned to the children of v according to the colors. If a point of $B(v) \cap (B \setminus \overline{B})$ gets one color it goes to the corresponding child of v. A two-colored point of $B(v) \cap (B \setminus \overline{B})$ goes to a child u if it is a vertex of a polygon $\rho_j(u)$. This can be done in linear time for all two-colored barriers of v if the polygons of its children and the x-order of $B(v)$ are known. The x-order of the barriers for children can be maintained in linear time.

The polygons that are split go to the corresponding children nodes. We mark the corresponding sides of these polygons. We assign the other polygons according to their marked sides, i.e. if a polygon P_i has marked top (bottom) side it goes to the child node in the same way as P_i was created.

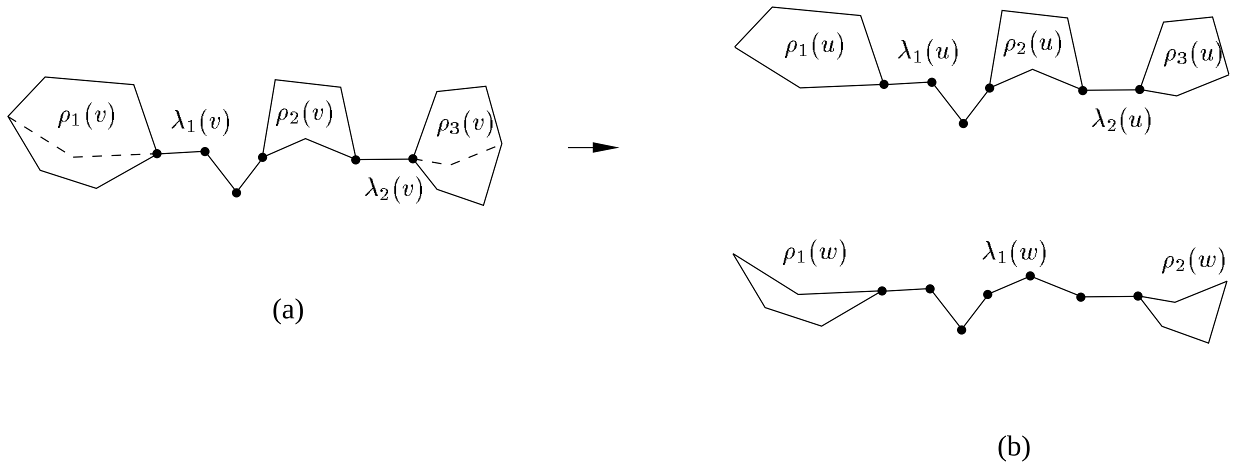


Fig. 11. Generating λ-paths. $\lambda_1(w)$ is combined from $\lambda_1(v)$, $a_2(v)b_2(v)$ and $\lambda_2(v)$.

The polygons $\rho_i(v)$ stored at a node v of T are connected by paths. We organize a hierarchical data structure Λ for these paths. A path connecting two polygons, say $\rho_i(v)$ and $\rho_{i+1}(v)$, is represented as a *λ-path* $\lambda_i(v)$ in Λ. A λ-path connecting two vertices, say c and d, is created when two polygons $\rho_i(v)$ and $\rho_{i+1}(v)$ are connected first time, i.e., $c = b_i(v)$ and $d = a_{i+1}(v)$ and the path cd is not λ-path of the parent of v. An elementary λ-path is a sequence of edges, for example the path (e_4, e_5) is λ-path of the left child in Fig. 10. In general, a λ-path is a list of λ-paths that are created earlier. This is illustrated in Fig. 11. The polygons and λ-paths of a node v are shown in Fig. 11 (a). The polygons and λ-paths of children of v are shown in Fig. 11 (b). The λ-paths of u are the same as ones of v. The node w has one λ-path that is combination of three paths $\lambda_1(v)$, $a_2(v)b_2(v)$ and $\lambda_2(v)$.

4.3 Point Location.

With each elementary path λ_i we store a binary search tree $T(\lambda_i)$ of its vertices so that point location can be done in $O(\log n)$ time. The binary tree $T(\lambda_i)$ can be constructed in linear time when the path λ_i is created.

Let λ_i be a λ-path. If λ_i has $O(1)$ λ-components, we create a binary search tree $T(\lambda_i)$ of size $O(1)$ whose leaves are the λ-components. If a λ-path λ_i has more than a constant number of λ-components we store a weighted binary search tree $T(\lambda_i)$ associated with λ_i. The leaves of the tree $T(\lambda_i)$ correspond to λ-components. The weight of a node is the total length (the number of edges) of the underlying path. We store the size of each λ-path in Λ to provide the weights of leaves in $T(\lambda_i)$. The tree $T(\lambda_i)$ can be constructed in linear time in terms of the number of leaves.

Lemma 3 *A point can be located in a λ-path in $O(\log n)$ time.*

Proof: Let λ_i be a λ-path and p be a point to locate in λ_i. The search procedure recursively processes a node of the tree $T(\lambda_i)$. If the tree $T(\lambda_i)$ has $O(1)$ size then $T(\lambda_i)$ is processed in $O(1)$ time. The tree $T(\lambda_i)$ can also be processed in $O(1)$ time even if $T(\lambda_i)$ is a weighted tree (when a leaf of $T(\lambda_i)$ with large λ-component is located in $O(1)$ time). The total number of these cases (over all trees) is $O(\log n)$ since the height of T is $O(\log n)$.

In the remaining cases the algorithm searches in weighted binary trees. Suppose that the search in $T(\lambda_i)$ visits $l = \Omega(1)$ vertices. The sequence of visited vertices has property that, among every $O(1)$ consecutive vertices, there is a vertex whose weight smaller the weight of its parent by at least a constant factor $\alpha > 1$. Therefore the weight of the last vertex (the size of the underlying path) dropped by a factor $O(c^l)$ where $c > 1$ is a constant. The total number (over all trees) of visited nodes is $O(\log n)$ since the total number of edges in the path λ_i is $O(n)$. ■

4.4 Shortest Path Retrieval.

We show how the shortest path homotopic to π_i can be computed. Let s and t be the endpoints of π_i. Consider the moment when π_i is processed at a node $v \in T$. If the points s and t lie in the same polygon then the procedure `Split` reports the shortest path. If the points s and t lie in different polygons, say $\rho_j(v)$ and $\rho_{j'}(v)$, then `Split` applied to them can output only two parts of the shortest path st. The interconnection can be computed as follows. For a polygon $\rho_l(v)$, $j < l < j'$, its contribution from $a_l(v)$ to $b_l(v)$ is computed in `Split` applied to $\rho_l(v)$. Connections between the polygons are λ-paths and can be extracted from Λ. An elementary λ-path is reported in linear time. If a λ-path is composed from other paths they are reported recursively.

Lemma 4 *The shortest path homotopic to π_i can found in $O(\log n + K)$ time using T where K is the number of edges of the shortest path.*

Proof: The existence of the path follows from the "aboveness" relation on the paths. The algorithm is correct since the shortest paths and its components, λ-paths, are connected.

The location of the points s and t requires $O(\log n)$ time by Lemma 3. The λ-paths of the shortest path st can be computed in $O(K + \log n)$ time. They can be reported in $O(K)$ time. ■

4.5 Space Reduction.

The above data structure takes $O(n \log n)$ space since an edge of a shortest homotopic path can participate in $O(\log n)$ polygons stored in T, one per level. In order to reduce the space we use the following trick. We construct T in top-bottom fashion and store the polygons only at one level. The nodes of T are processed level by level and the shortest paths are reported at the time when their nodes are traversed. The λ-paths are updated when T is processed.

We show that the space required for T is $O(n)$. The polygons stored at nodes of the same level are interior-disjoint. They take $O(n)$ space since every edge is stored at most twice. The edges of elementary λ-paths are stored in Λ so that an edge is stored at most twice.

Theorem 5 *The above algorithm computes the shortest homotopic paths in $O(n \log n + k)$ time.*

Proof: By Lemma 4 it suffices to show that T is constructed in $O(n \log n)$ time. By Lemma 1 the processing of the nodes at the same level takes $O(n)$ time since the total complexity of polygons is linear. The theorem follows. ■

Combining this with the preprocessing we obtain the main result for simple paths.

Theorem 6 *SHP problem can be solved in $O((n + k_{in}) \log n + k_{out})$ time assuming that a triangulation of the paths and the barriers is given.*

Corollary 7 *SHP problem can be solved in $O(n \log^{1+\varepsilon} n + k_{in} \log n + k_{out})$ time.*

5 Non-simple Paths.

If the paths are allowed to intersect themselves, see Fig. 12, an algorithm by Hershberger and Snoeyink [14] can be applied. The running time of the algorithm is $O(kn)$. As Efrat *et al.* [10] pointed out there exist non-trivial examples (even for simple paths!) where k can be much larger than n, for instance $k = \Omega(2^n)$. We are not aware of any output-sensitive algorithms for finding the shortest homotopic paths in the case of non-simple paths. We design an algorithm that computes the shortest paths in $O(\log^2 n)$ time per vertex.

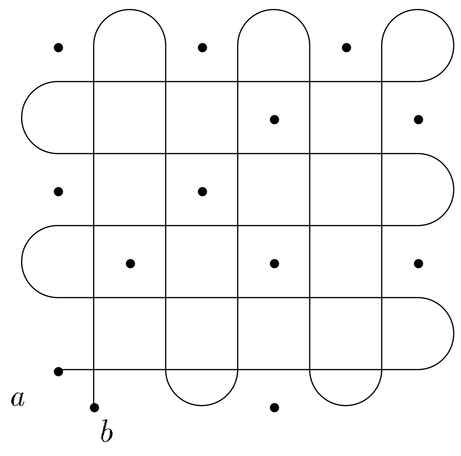


Fig. 12. Non-simple path ab is homotopic to the segment $[ab]$.

Our algorithm has two phases:

(i) we reduce the problem to x-monotone paths, and

(ii) we find the shortest homotopic paths for the monotone paths.

The algorithm of the first phase is a slight modification of the algorithm by Efrat *et al.* [10] that shortcuts the paths using the simplex range search [8]. A data structure of the simplex range search is constructed for the barrier points. Shortcuts applied for simple paths are illustrated in Fig. 13 (a) and (b). In order to shortcut the path using vertical lines the algorithm uses the range search to find the leftmost or the rightmost barrier point in the query triangle. The triangles of the range search are shaded in Fig. 13. Efrat *et al.* [10] distinguished *left* and *right* shortcuts that depends on what side of the new vertical segment the range search region lies. For example, a left shortcut is depicted on Fig. 13 (a) and three right shortcuts are depicted on Fig. 13 (b).

Lemma 8 (Efrat *et al.* [11]) *Let π be a path, and let μ be a result of performing left and right phases of elementary vertical shortcuts on π until no more are possible. Suppose that the local left and right extremes of μ are locked at the terminals $t_{i_1}, \ldots, t_{i_l}$ in that order. Let σ be a shortest path homotopic to π. Then the local left [right] extremes of σ are locked at* *exactly the same ordered list of terminals, and furthermore, the portion of σ between t_{i_j} and $t_{i_{j+1}}$ is a shortest path homotopic to the portion of μ between those same terminals.*

Lemma 9 (Efrat *et al.* **[11])** *The number of elementary vertical shortcuts that can be applied to a set of paths with a total of k_{in} segments is at most $2k_{in}$.*

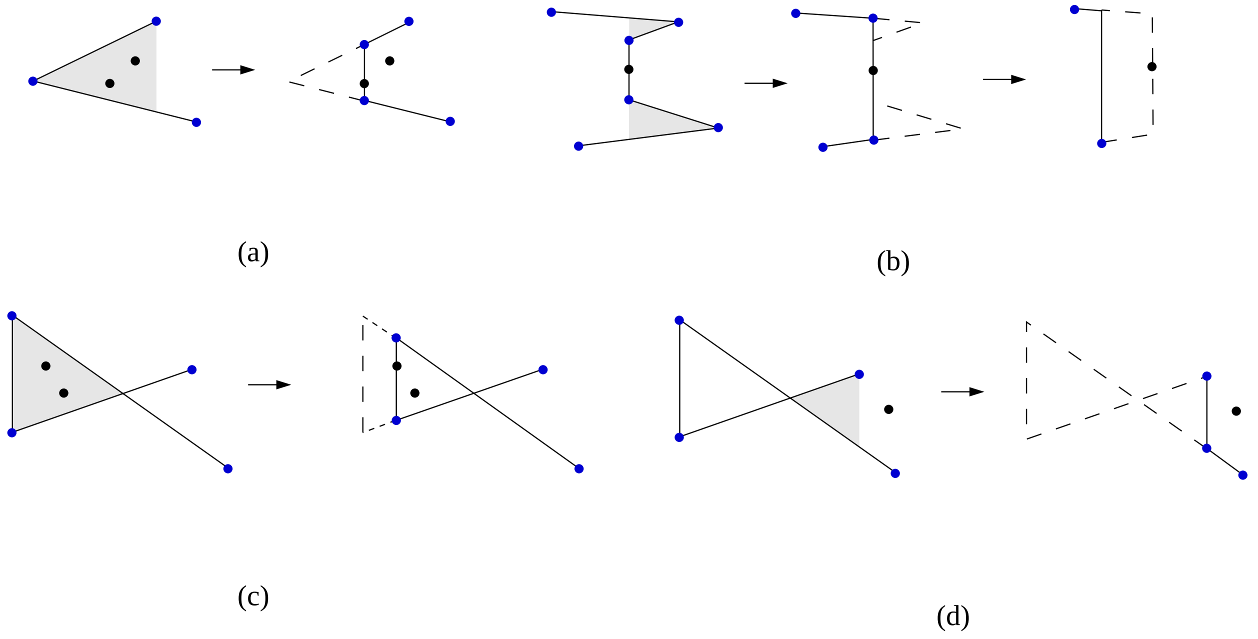


Fig. 13. (a) and (b) Shortcuts of simple path. (c) and (d) Shortcuts of non-simple paths. The regions for range search are shaded.

We show that if the paths are non-simple shortcuts can still be applied. If two edges of a simple path cross they can be shortcut by applying shortcut test twice: before the crossing point and after it. An example illustrated in Fig. 13 (c) shows a region for the first search (shaded triangle) and the shortcut found in the region (determined by the leftmost barrier point). Fig. 13 (d) illustrates an example where two searches are needed and the second region is shaded. The result of shortcuts is a set of x-monotone paths (possibly intersecting).

Lemma 10 *Let π be a path and let σ be the shortest path homotopic to π and let μ be a result of performing shortcuts on π until no more are possible. Let $\mu_1, \mu_2, \ldots$ be x-monotone components of μ and let $\sigma_1, \sigma_2, \ldots$ be x-monotone components of σ. Then*

(i) *each path σ_i has the same endpoints as μ_i and is the shortest homotopic path of μ_i, and*
(ii) *the number of shortcuts is at most two times the number of edges of π.*

Proof: (i) It follows by essentially the same "rubber band" argument as [11]. The endpoints of the paths μ_i are the terminals that must be present in σ even if π is non-simple path.

(ii) We use the counting scheme by Efrat *et al.* [11]. Each shortcut removes either a vertex or an edge of π, see Fig. 13. ■

The first phase takes $O(n^{2+\varepsilon})$ time for preprocessing, $O(k_{in} \log^2 n)$ time for shortcutting the paths, and $O(n^{2+\varepsilon})$ space.

The second phase is more difficult.

5.1 Implicit Funnel Algorithm.

Recall that the problem is to find the shortest homotopic paths in the presence of point obstacles. One can apply the algorithm from Lemma 1. As a result, the running time is at least n times the output size which is too expensive. Instead we use an idea of implicit representation of funnel. Let $\pi = p_1, p_2, \ldots$ be an x-monotone path. Let $l(p)$ denote the vertical line passing through a point p. For a two points p and q, let $l(p)l(q)$ denote the slab between two vertical lines $l(p)$ and $l(q)$. A *funnel with apex* p_1 *and base* $l(p)$ is defined as follows. Let B_U be the set of points that includes p_1 and the barrier points lying in the slab $l(p_1)l(p)$ and above π. Let U be the lower envelope of B_U. Let B_L be the set of points that includes p_1 and the barrier points lying in the slab $l(p_1)l(p)$ and below π. Let L be the upper envelope of B_L. The funnel is the area between U and L restricted by $l(p)$, see for example Fig. 14 (a) where $p = p_2$. We represent the funnel implicitly as the triangle defined by the extensions of edges incident to p_1 and $l(p_2)$, see Fig. 14 (b).

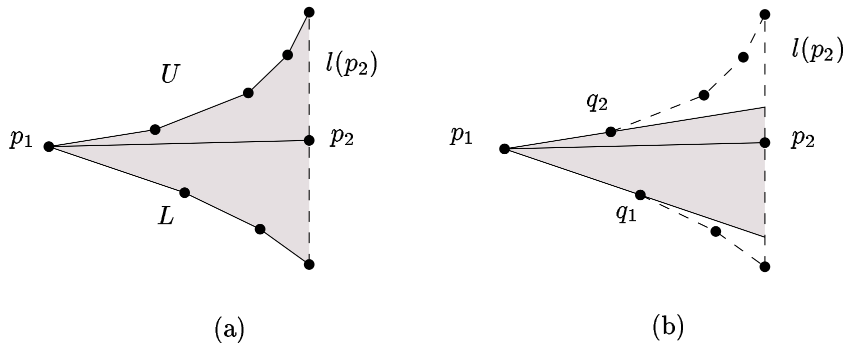


Fig. 14. (a) Funnel. (b) Implicit representation of funnel.

For every barrier point b we store all the remaining barrier points in a list $L(b) = (b_1, b_2, \ldots)$ sorted by the slopes $bb_i, 1 = 1, 2, \ldots$. The list $L(b)$ can be computed in $O(n \log n)$ time and n lists for all the barriers can be computed in $O(n^2 \log n)$ time.

We show how to compute the implicit funnel with base $l(p_2)$. Let B' be the set of barrier points in the slab $l(p_1)l(p_2)$. For a point $b \in B'$ let $\Delta(b)$ denote the triangle formed by the lines p_1b, p_1p_2 and $l(p_2)$. For any ray p_1b, $b \in B'$ we can test if $\Delta(b)$ contains a barrier point using the simplex range search. Using a binary search on $L(p_1)$ (see more details below) we can compute the upper side p_1q_2 and the lower side p_1q_1 of the implicit funnel that have the highest and the lowest slopes such that $\Delta(q_i)$ has no barrier points, see Fig. 14 (b).

Suppose that we have computed the implicit funnel F up to the vertical line through a point p_i. The funnel F is a triangle with vertices p_1, r_1 and r_2. Let q_1 and q_2 be the points of B determining the sides of F, see Fig. 15. We do not assume the points $p_2, p_3, \ldots, p_i$ lie inside the funnel F, for example the point p_i in Fig 15. In order to extend the funnel F to the next vertical line $l(p_{i+1})$ we apply binary search on $L(p_1)$ and compute

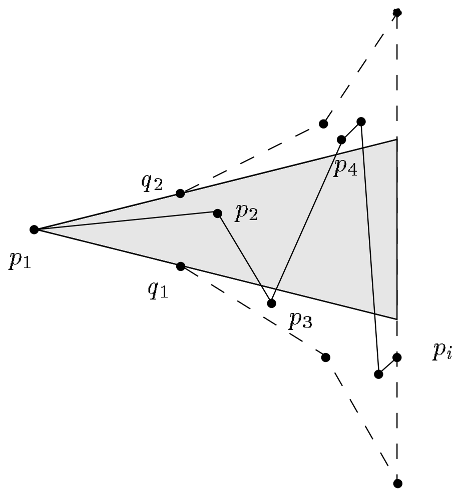


Fig. 15. The implicit funnel and the path $p_1 p_2 \ldots p_i$.

(i) q_2', the lowest point (point with the lowest slope $p_1 q_2'$) point visible from p_1 in the slab $l(p_i)l(p_{i+1})$ and above the segment $p_i p_{i+1}$, and

(ii) q_1', the highest point visible from p_1 in the slab $l(p_i)l(p_{i+1})$ and below the segment $p_i p_{i+1}$.

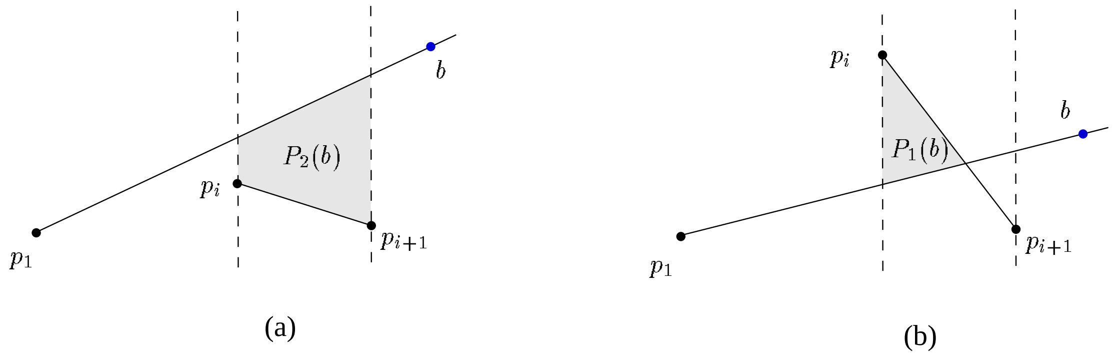


Fig. 16. (a) Polygon $P_2(b)$. (b) Polygon $P_1(b)$.

We describe the binary search in step (i). Let b be a barrier point. We define a polygon $P_2(b)$ as a locus of points in the slab $l(p_i)l(p_{i+1})$ above the segment $p_i p_{i+1}$ and below the line $p_1 b$, see Fig. 16 (a). A polygon $P_2(b)$ is either a trapezoid or a triangle. The test included in the binary search for q_2' is whether, for a point $b \in B - \{p_1\}$, the polygon $P_2(b)$ is empty. If $P_2(b)$ is a triangle then the test can be done using simplex range search in $O(\log n)$ time. If $P_2(b)$ is a trapezoid then it can be partitioned into two triangles and range search is applied to each triangle. Thus the test can be done in $O(\log n)$ time. The overall time for binary search is $O(\log^2 n)$.

The binary search in step (ii) is similar to the step (i). We define a polygon $P_1(b)$ as a locus of points in the slab $l(p_i)l(p_{i+1})$ below the segment p_ip_{i+1} and above the line p_1b, see Fig. 16 (b). The test included in the binary search for q_1' is whether, for a point $b \in B - \{p_1\}$, the polygon $P_1(b)$ is empty. Clearly, the polygon $P_1(b)$ can be tested in $O(\log n)$ time.

If the wedges $q_1p_1q_2$ and $q_1'p_1q_2'$ intersect then we update the funnel F to be the intersection of the wedges $q_1p_1q_2$ and $q_1'p_1q_2'$ and the slab $l(p_1)l(p_{i+1})$. Otherwise the funnel with apex at p_1 collapses. Suppose that the ray p_1q_2' is below the ray p_1q_1, see Fig. 17. The remaining case where the ray p_1q_1' is above the ray p_1q_2 is symmetrical. The funnel point q_1 is the next point of the shortest homotopic path. We report q_1 and update the funnel F as follows. We assign the apex of F to $p_1 = q_1$ and the base $l(p_i)$. The sides of F can be found similarly to the computation of the first funnel which is determined by the the segment p_1p_2 lying in F. The segment p_1r_1 lies in F and can be used to compute F. Then we continue the computation of the funnel up to the line $l(p_{i+1})$.

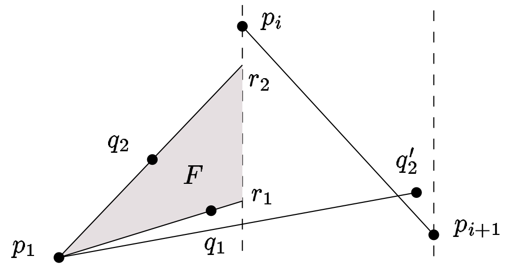


Fig. 17. Funnel collapse.

Theorem 11 *The shortest homotopic paths of polylines can be computed in* $O(n^{2+\varepsilon} + k \log^2 n)$ *time.*

Proof: The preprocessing includes the following steps

(i) the computation of the lists $L(b)$ for all barrier points $b \in B$,

(ii) the construction of data structure for simplex searching [8],

(iii) shortcutting the input paths and creating x-monotone paths.

These steps take $O(n^2 \log n)$, $O(n^{2+\varepsilon})$, and $O(k_{in} \log^2 n)$ time respectively. The total preprocessing time is $O(n^{2+\varepsilon} + k_{in} \log n)$. The total number of vertices of the produced x-monotone paths is $O(k_{in})$.

In the second phase the implicit funnel algorithm spends $O(\log n)$ tests in every binary search. Each test takes $O(\log n)$ time so the total time of each binary search is $O(\log^2 n)$.

We count the total number of binary search calls. There are two ways of applying a binary search. We apply the binary search when we promote the base of the implicit funnel. There are $O(k_{in})$ such calls of the binary search. The second type of binary search call happens when the implicit funnel collapses. In this case we report the apex of the funnel as a vertex of the shortest homotopic path and we charge it to the binary search call. The total number of these calls is at most k_{out}. Thus the total number of binary search calls is $O(k_{in}) + k_{out} = O(k)$. The theorem follows. ∎

Remark. One can apply Matoušek's data structure for range simplex searching [19] with quadratic space and $O(\log^3 n)$ query time. The implicit funnel algorithm with this data structure computes the shortest homotopic paths in $O(n^2 \log n + k \log^4 n)$ time using $O(n^2)$ space.

6 An Improvement for Small k.

The above algorithm is output-sensitive and efficiently finds the shortest homotopic paths if their size is $\Omega(n^2)$. Note that k can be arbitrarily large relative to n since it includes k_{in}. The complexity of the shortest homotopic paths is $\Theta(nk_{in})$ in the worst case which can dominate the preprocessing time $O(n^{2+\varepsilon})$. In order to reduce the total running time we can use standard space-time tradeoff techniques. Matoušek [19] applied the technique to the range searching with efficient hierarchical cuttings and obtained $O((N + M + (NM)^{2/3})\text{polylog}(N + M))$ running time for answering M range queries against N points in the plane.

Theorem 12 (Matoušek [19]) *Let P be an n-point set in $\mathbb{R}^d$ and let m be a parameter, $n \le m \le n^d$. The range searching problem with ranges being intersections of p half-spaces, $1 \le p \le d + 1$, can be solved with space $O(m)$, query time $O(\frac{n}{m^{1/d}} \log^{p-(d-p+1)/d} \frac{m}{n})$ and preprocessing time $O(n^{1+\varepsilon} + m(\log n)^{\varepsilon})$ where ε denotes an arbitrarily small positive constant.*

6.1 Optimization Queries

Our implicit funnel algorithm uses the simplex range searching structure [8]. The algorithm can be modified to work with other data structures. The problem is that the simplex range queries are not sufficient. We introduce a new type of queries and show how to modify data structures for simplex range searching to answer these queries. First, we define extreme points. Let p be a point in the plane and let A be a subset of the plane such p lies outside the convex hull of A. A point $q \in A$ is *extreme* with respect to p and A if pq is an edge of the convex hull of $A \cup \{p\}$. There can be at most two extreme points with respect to p and A.

We define an *optimization simplex query* and the problem of optimization simplex searching as follows. Let S be a set of n points in the plane. Preprocess S so that, given any query simplex σ and a query point p outside σ, the extreme points with respect to p and $S \cap \sigma$ can be computed efficiently. The optimization queries can be used to find the implicit funnel. For example, the point q_2' in the extension of the funnel to $l(p_{i+1})$ can be found using the optimization simplex query with respect to the point p_1 and the simplex formed by three lines $l(p_i), l(p_{i+1})$ and $p_i p_{i+1}$, see Fig. 16 (a).

We demonstrate how Matoušek's data structure [18] can be modified to answer optimization simplex queries. The data structure is based on the following Partition Theorem.

Theorem 13 (Matoušek [18]) *Let S be a set of n points in $\mathbb{R}^d, d \geq 2$, let s be an integer parameter, $2 \leq s < n$, and let $r = n/s$. There exists a simplicial partition Π for S, whose classes S_i satisfy $s \leq |S_i| < 2s$, and whose crossing number is $O(r^{1-1/d})$.*

As in [18] we use the simplicial partition recursively to create a partition tree. The leaves of the tree form a partition of S into subsets of size $O(1)$. Each internal node v of the tree corresponds to a subset $S_v \subset S$ and to a simplicial partition Π_v of S_v. Of that simplicial partition, we store simplices and the cumulative weights of the corresponding subsets in the node v. We also compute and store the convex hull of S_v. The simplicial partitions Π_v are constructed by applying Partition Theorem with $s = \lfloor \sqrt{|S_v|} \rfloor$ (since $d = 2$). The depth of the partition tree is $O(\log \log n)$.

An optimization query can be answered as follows. Let p be the query point and let σ be the query simplex. We start with the root of the partition tree. In general, a node v is processed by checking the simplicial partition Π_v. If a simplex σ' of Π_v is contained in σ we compute two lines passing through p and tangent to the convex hull of $\sigma' \cap S$. These tangent lines can be computed in $O(\log n)$ time. The points defining the lines are the extreme points with respect to p and $\sigma' \cap S$. The children nodes whose simplices are disjoint from σ can be ignored. We proceed with the remaining simplices. Let $r = n/s$. By Partition Theorem we recurse in $O(\sqrt{r})$ simplices only.

For the query time $T(n)$, we get a recurrence

$$T(n) \leq O(r) \log n + O(\sqrt{r}) T(2n/r), r = \sqrt{n},$$

with initial condition $T(n) = O(1)$ if $n = O(1)$. The solution is easily verified to be $T(n) = O(\sqrt{n} \cdot \text{polylog}(n))$.

We analyze the space requirement. Let $M(n)$ be the space needed for a modified partition tree with n points. The recurrence is

$$M(n) = O(n) + O(\sqrt{r}) T(2n/r), r = \sqrt{n}.$$

The solution of the recurrence is $M(n) = O(n)$.

We show that the preprocessing time is the same as the time needed for construction partition tree [18], $O(n \log n)$. We spend an additional time to construct convex hulls in each node. This gives the recurrence $A(n) = O(n \log n) + O(\sqrt{r})A(2n/r), r = \sqrt{n}$ where $A(n)$ is the additional time for a partition tree with n points. The solution is $A(n) = O(n \log n)$. Thus we proved the following lemma.

Lemma 14 *We can preprocess n points in the plane in $O(n \log n)$ time using $O(n)$ space so that optimization simplex queries can be answered in $O(\sqrt{n} \cdot \mathrm{polylog}(n))$ time.*

6.2 Guessing k and Space-Time Tradeoff

We want to apply range simplex searching depending on the parameter k. The problem is that k has the component k_{out} whose value is unknown in the beginning. We use standard logarithmic method that estimates k. We run our algorithm with guess that $k = 2k_{in}$. We stop the algorithm if it finds more than k_{in} vertices in the output path. Then we double the guess $k = 4k_{in}$ and repeat. If the guessed k is less than $\sqrt{n}/\mathrm{polylog}(n)$ we apply the algorithm from Lemma 14. If the guessed k exceeds $n^{2+\varepsilon}$ for some $\varepsilon > 0$, we run the algorithm from Theorem 11. This strategy involves at most $O(\log n)$ runs and the total time is proportional to the time of the last run. If k is in the range $[\sqrt{n}/\mathrm{polylog}(n), n^{2+\varepsilon}]$ the algorithm proceeds as follows.

We apply standard tradeoff technique, see for example [19]. The algorithm can occupy the space $m = O(n + k + (nk)^{2/3})$. As in Lemma 14 we allocate the partition tree for a suitable value of the parameter r and, for each its leaf, we create Matoušek's data structure with quadratic space (see remark in the previous Section). Since the size of partition tree is $O(r)$ we find r from $m = r^3$.

Theorem 15 *The shortest homotopic paths for paths that are not necessarily simple can be found in time $T(n,k)$ where*

$$T(n) = \begin{cases} O(n \log n) & \text{if } k < \sqrt{n}/\mathrm{polylog}(n), \\ O((n + k + (nk)^{2/3})\mathrm{polylog}(n)) & \text{if } \sqrt{n}/\mathrm{polylog}(n) < k < n^{2+\varepsilon}, \\ O(k \log^2 n) & \text{if } k > n^{2+\varepsilon}. \end{cases}$$

Proof: The algorithm is correct because it is correct in each of three cases. It should be noted that the value of k is always a guess and the algorithm halts if the implicit funnel algorithm discovers more than $k - k_{in}$ output edges.

The running time analysis for the second case is similar to [19]. ∎

7 Conclusion

We presented two output-sensitive algorithms for computing shortest homotopic paths in the plane. The algorithm for simple paths runs in $O(T(n, k_{in}) + (n + k_{in}) \log n + k_{out})$ time where $T(n, k_{in})$ is the running time for triangulation. The algorithm for non-simple paths takes $O(\log^2 n)$ time per vertex in the shortest homotopic paths and $O(n^{2+\varepsilon} + k_{in} \log^2 n)$ time depending on the input. For the case of relatively small k we showed that the running time can be improved to $O((n + k + (nk)^{2/3})\text{polylog}(n))$. In the case $k = O(n)$ the running time is $O(n^{4/3}\text{polylog}(n))$. The SHP problem is related to the range searching and Hopcroft's problem. One could not expect an improvement of $O(n^{4/3})$ bound for SHP problem unless this bound is improved for Hopcroft's problem. For example, Cabello *et al.* [5] show that Hopcroft's problem can be reduced to the problem of testing homotopy for non-simple paths.

Acknowledgements

I would like to thank the anonymous referees for many suggestions and comments.

References

[1] M. A. Armstrong. Basic Topology. McGraw-Hill, London, UK, 1979.

[2] R. Bar-Yehuda and B. Chazelle. Triangulating disjoint Jordan chains. *Internat. J. Comput. Geom. Appl.*, 4(4):475–481, 1994.

[3] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pp. 80–86, 1983.

[4] S. Bespamyatnikh. Computing Homotopic Shortest Paths in the Plane. In *Proc. 14th ACM-SIAM Sympos. Discrete Algorithms*, pp. 609–617, 2003.

[5] S. Cabello, Y. Liu, A. Mantler, and J. Snoeyink. Testing homotopy for paths in the plane. In *Proc. 18th Annu. ACM Sympos. Comput. Geom.*, pp. 160–169, 2002. `http://www.cs.uu.nl/people/sergio/publications /clms-thpp-02.pdf`

[6] B. Chazelle. A theorem on polygon cutting with applications. In *Proc. 23th Annu. IEEE Sympos. Found. Comput. Sci.*, pp. 339–349, 1982.

[7] B. Chazelle. An algorithm for segment-dragging and its implementation. *Algorithmica*, 3:205–221, 1988.

[8] B. Chazelle, M. Sharir, and E. Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. *Algorithmica*, 8:407–429, 1992.

[9] R. Cole and A. Siegel. River routing every which way, but loose. In *Proc. 25th Annu. IEEE Sympos. Found. Comput. Sci.*, pp. 65–73, 1984.

[10] A. Efrat, S. Kobourov, and A. Lubiw. Computing homotopic shortest paths efficiently. *manuscript*, April 29, 2002, 12 pages. `http://arxiv.org/ps/cs.CG/0204050`.

[11] A. Efrat, S. Kobourov, and A. Lubiw. Computing homotopic shortest paths efficiently. In *Proc. 10th Annu. European Sympos. Algorithms*, pp. 411–423, 2002. `http://link.springer.de/link/service/series/0558/bibs/2461/24610411.htm`

[12] S. Gao, M. Jerrum, M. Kaufmann, K. Mehlhorn, W. Rülling, and C. Storb. On continuous homotopic one layer routing. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pp. 392–402, 1988.

[13] L. 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.

[14] J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. *Comput. Geom. Theory Appl.*, 4:63–98, 1994.

[15] D. T. Lee and F. P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14:393–410, 1984.

[16] C. E. Leiserson and F. M. Maley. Algorithms for routing and testing routability of planar VLSI layouts. In *Proc. 17th Annu. ACM Sympos. Theory Comput.*, pp. 69–78, 1985.

[17] C. E. Leiserson and R. Y. Pinter. Optimal placement for river routing. *SIAM J. Comput.*, 12:447–462, 1983.

[18] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8(3), pp. 315–334, 1992.

[19] J. Matoušek. Range searching with efficient hierarchical cuttings. *Discrete Comput. Geom.*, 10(2), pp. 157–182, 1993.

[20] J. S. B. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pp. 633–701. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[21] J. R. Munkres. Topology: A first course. Prentice Hall, Englewood Cliffs, NJ, 1975.

[22] R. Pinter. River-routing: Methodology and analysis. In R. Bryan, editor, *Third Caltech Conference on VLSI*, Computer Science Press, Rockville, Maryland, 1983.

[23] D. Richards. Complexity of single layer routing. *IEEE Transactions on Computers*, 33:286–288, 1984.