Computing Homotopic Shortest Paths in the Plane

Sergei Bespamyatnikh*

Abstract

We address the problem of computing homotopic shortest paths in presence of obstacles in the plane. The problems on homotopy of the paths received attention very recently [3, 8]. We present two output-sensitive algorithms, for simple paths and non-simple paths. The algorithm for simple paths improves the previous algorithm [8]. 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 [13] where n is the number of obstacles.

1 Introduction.

Finding shortest paths in a geometric domain is a fundamental problem [18]. Chazelle [4] and Lee and Preparata [14] gave a funnel algorithm that computes the shortest path between two points in a simple polygon. Hershberger and Snoeyink [13] 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 [7, 15, 9]. For this problem it is required to route wires with fixed terminals among 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 [16, 20, 21].

The topological concept of homotopy captures the notion of deforming paths [1, 19]. A *path* is a continuous map $\pi : [0,1] \to \mathbb{R}^2$. Let $\alpha, \beta : [0,1] \to \mathbb{R}^2$ 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 avoids B. 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 $\Gamma : [0,1] \times [0,1] \to \mathbb{R}^2$ with the following three properties:

1. $\Gamma(0,t) = \alpha(t)$ and $\Gamma(1,t) = \beta(t)$ for $0 \le t \le 1$,
2. $\Gamma(s,0) = \alpha(0) = \beta(0)$ and $\Gamma(s,1) = \alpha(1) = \beta(1)$ for $0 \le s \le 1$,
3. $\Gamma(s,t) \notin B$ for $0 \le s \le 1$ 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 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}, k_{out} be the total number of edges of the input/output paths and let $k = k_{in} + k_{out}$.

Hershberger and Snoeyink [13] 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.* [3] studied the problem of testing whether two paths with common endpoints in 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 input size $O(n + k_{in})$ (of their problem) in the worst case. The key idea of the algorithm is to compute *canonical* paths and they can be found by rectifying the paths and shortcutting them using *segment dragging* by Chazelle [5].

Very recently Efrat *et al.* [8] presented output sensitive algorithm for computing the shortest homotopic paths. The algorithm runs in $O(n^{3/2} + k \log n)$ time where $\varepsilon > 0$ is arbitrary small constant. In the first phase of the algorithm they apply shortcuts to compute x-monotone paths using efficient algorithm for computing canonical paths [3]. The computation of the canonical paths is based on the segment dragging [5, 3]. The monotone paths are 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.* [8] also show that this bound can be improved to $O(n \log n + k)$ using randomization.

*Department of Computer Science, University of Texas at Dallas, Box 830688, Richardson, TX 75083, USA. E-mail: besp@utdallas.edu

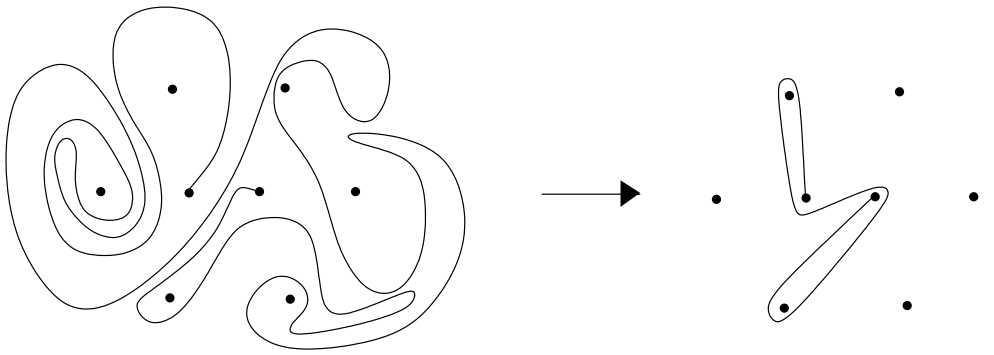

Fig. 1: Shortest path preserving homotopy type.

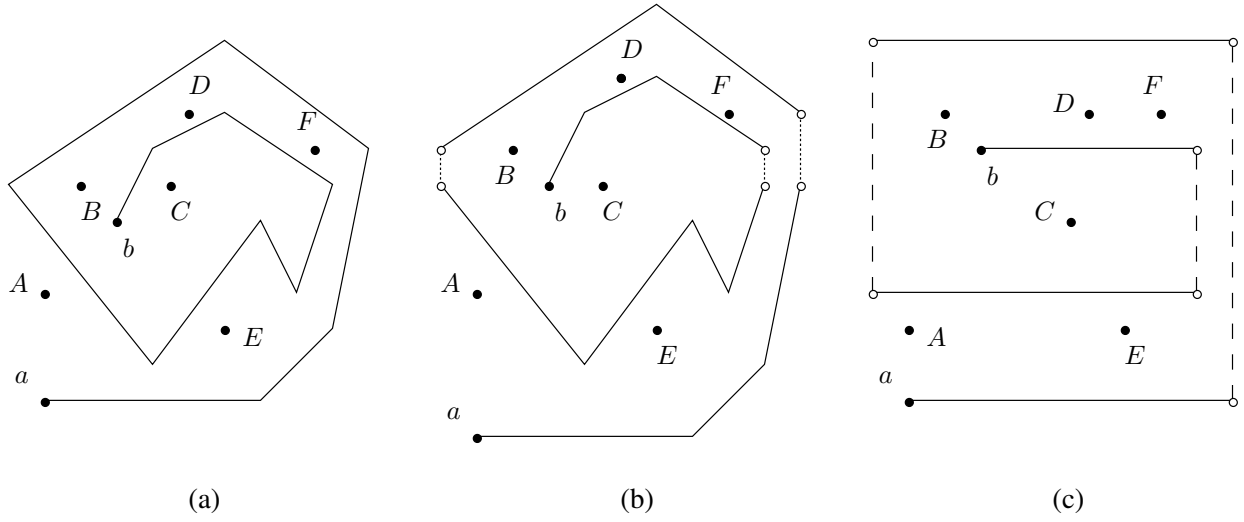

Fig. 2: (a) Path ab, (b) Monotone paths, (c) Rectified paths.

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. For non-simple paths we design an *implicit funnel* algorithm that computes the shortest paths in $O(n^{2+\varepsilon} + k_{in} \log^2 n + k_{out})$ time. We also show that for relativly small k the running time can be improved to $O(n \cdot polylogn + n^{2/3}k^{2/3}polylogk + k \cdot polylogk)$ using hierarchical cuttings [17].

We mention a related work on finding homotopic paths amidst semi-algebraic obstacles [10, 11].

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 [3, 8]. 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.

2 Reduction to Monotone Paths.

In this Section we briefly describe the construction of the canonical paths [3] and the bundling [8]. The canonical paths are *x-monotone*, i.e. 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 one time. One can partition a path into x-monotone pieces by exploring its vertices locally, i.e. we break the path in 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. **??**). We can treat these monotone paths as horizontal segments and obtain *rectified paths* [3], see Fig. 2 (c).

To rectify the paths one needs "aboveness" relation between the monotone paths and the 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. 4.

In order to generate the rectified paths we compute a triangulation or a trapezoidation [8] using algorithm of 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 "aboveness" relation that is acyclic. The rectified paths and new positions of barriers are computed using their ranks as y-coordinates [3]. The rectified paths can be shortcut by vertical segments producing canonical rectified paths. Applying the segment dragging queries by Chazelle [5] shortcuts can be done in $O(k_{in} \log n)$ time using $O(n \log n)$ preprocessing time and $O(n)$ space.

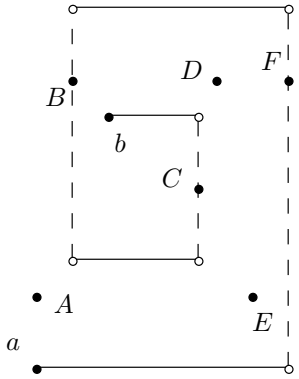

Fig. 3: Rectified canonical path.

The number of homotopically different paths produced by shortcutting is at most $2n$ [8]. The homotopic paths can be bundled reducing the problem to the case $k_{in} \leq 2n$. 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'.

3 Shortest Monotone Paths.

Let Π be the set of paths produced by shortcuts and bundling. The number of paths is $O(n)$. There is an order $\pi_1, \pi_2, \ldots$ of the paths that is consistent with the aboveness relation. We assume that π_i is "below" π_{i+1}. The order of the paths $\pi_i, i = 1, 2, \ldots$ is determined by the *aboveness tree* [3]. Recall that B is the set of all barriers. Let B_i be the set of barriers below $\pi_i, i = 1, 2, \ldots$. We reduce the problem of constructing the shortest homotopic paths to the finding of a shortest monotone path in a simple polygon with two colored barriers so that the barriers are separated by a 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 s, t are 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/below the path is red/blue respectively.

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

LEMMA 3.1. *The problem above can be solved in linear time assuming that the barriers are sorted by x-coordinates.*

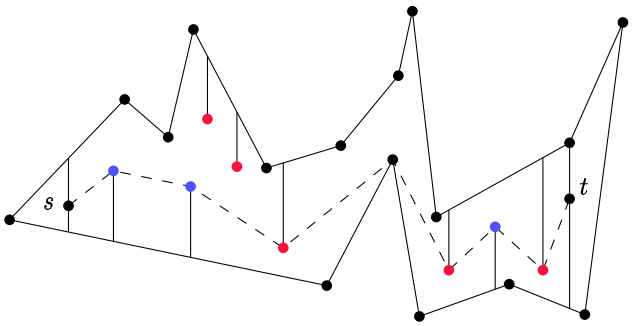

Fig. 4: Shortest path separating red and blue points.

Proof. We connect the barrier points inside P to the boundary of P according to the colors, see Fig. 4 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 shortest path in a simple can be found in linear time [12, 13]. Clearly, the produced path satisfies the color constraint.

4 Shortest Paths.

Consider a path $\pi_i \in \Pi$. Let s and t denote its start and target points. The shortest path corresponding to

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

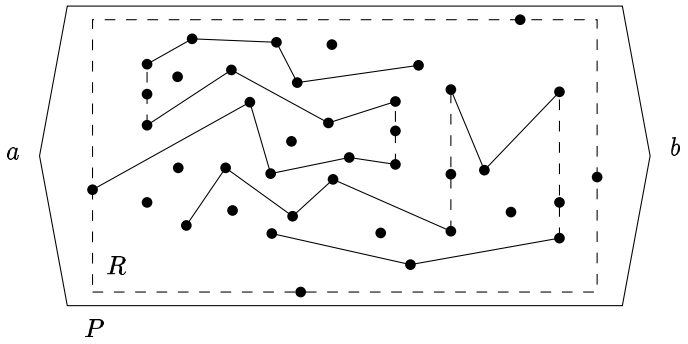

Fig. 5: Polygon P.

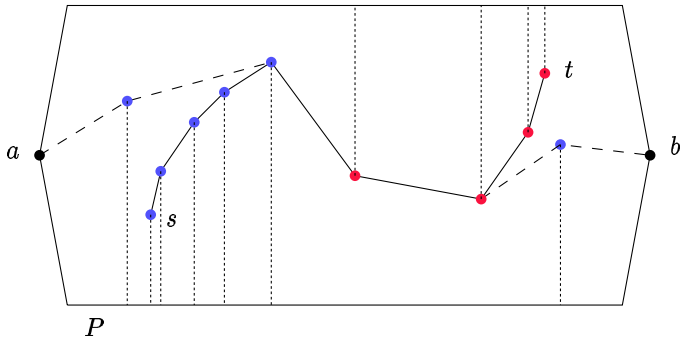

Fig. 6: The shortest paths ab and st.

π_i can be constructed using Lemma 3.1 as follows. Let R be a rectangle containing all data points inside. 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. 5. The path π_i generates a partition of the set B into two sets B_i and $B - B_i$. We color the points of S_i in blue and the points of $B - B_i$ in red. Applying Lemma 3.1 to the polygon P we obtain the shortest path corresponding to π_i.

4.1 Splitting a Polygon. In order to apply the algorithm from Lemma 3.1 recursively the polygon P needs to be divided into two polygons. The idea is to connect the points a and b. If we apply the algorithm to find the shortest path ab among barriers colored according to π_i, it will not necessarily contain the shortest path st, see Fig. 6. 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.

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 `Split`(P, s, t).

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 and contains multiple polygons, see Fig. 7 (a). The polygons on either side of the path $astb$ have the property that they are x-monotone and their projections on x-axis are disjoint. Our algorithm can deal with multiple polygons and divide them recursively. In order to make efficient computation we build the following data structure.

4.2 Data Structure. Let T be a binary search tree of height $O(\log n)$ whose nodes correspond to the paths $\pi_1, \pi_2, \ldots$ in this order. This can be done 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

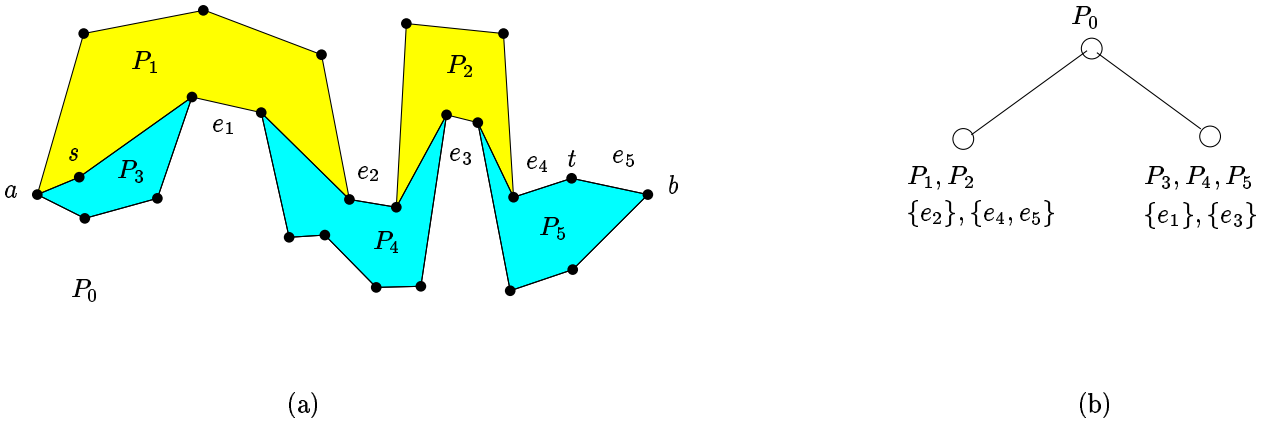

Fig. 7: (a) Multiple polygons. (b) Tree representation.

$\{\pi_j, j > i\}$. 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 sorted order of x-projections. 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. 7 (a) are marked "bottom" and P_3, P_4, P_5 are marked "top".

The polygons associated with its children are constructed by splitting these polygons 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 `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 `Split`$(\rho_i(v), s, b_i(v))$ and `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. 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.

If new polygons are generated on either side we store the paths connecting them at the corresponding children nodes, see Fig. 7 (b). If a or b disappears from generated polygons we also store the paths from it to polygons, for example the path e_4e_5 connecting P_2 and b in Fig. 7 (b). We denote these paths $\lambda_1(v), \lambda_2(v), \ldots$. A path $\lambda_i(v)$ of a node v can be connected (share a vertex) with a path $\lambda_j(u)$ of its parent node u (for example, a point b in Fig. 7 (b) can be the leftmost point of a λ-path on the above level). We store pointers between connected λ-paths. Note that a λ-path can be connected to at most two λ-paths, one λ-path per endpoint.

4.3 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 P_i and P_j, then `Split` applied to them can output only two parts of the shortest path st. The interconnection can be computed as follows. If $P_i$and P_j are connected by a λ-path of v we output it. Otherwise we find all λ-paths of v between P_i and P_j and, for every two consecutive λ-paths, we follows their pointers to λ-paths reducing the gap. Repeat this procedure until all gaps are closed. Note that these pointers lead to nodes of different levels and the levels may go up and down.

LEMMA 4.1. *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 its paths are λ-paths that are connected.

The location of the points s and t requires $O(\log n)$ time and every edge of the shortest path is retrieved in $O(1)$ time.

4.4 Space Reduction. The above data structure takes $O(n \log n)$ since

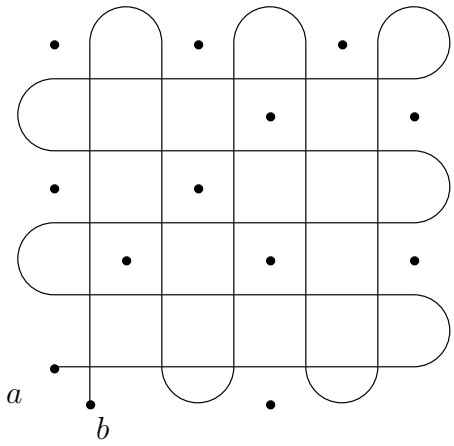

Fig. 8: Simple path ab is homotopic to the segment $[ab]$.

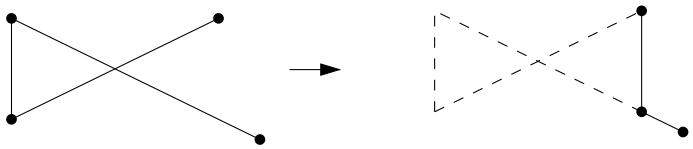

Fig. 9: Shortcut of non-simple path.

(i) every edge of a shortest homotopic path can participate in $O(\log n)$ polygons stored in T, one per level, and

(ii) there are $O(n)$ possible edges stored in T since they form a planar graph with n vertices.

Unfortunately the both above bounds are tight. In order to reduce the space we use the following trick. We store the polygons only at leaves of T when T is constructed. The remaining information suffices to recurse the construction and retrieve the shortest paths. 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 λ-paths are stored once in T.

THEOREM 4.1. *The above algorithm computes the shortest homotopic paths in $O(n \log n + k)$ time.*

Proof. By Lemma 4.1 it suffices to show that T is constructed in $O(n \log n)$ time. By Lemma 3.1 the processing of the nodes at the same level takes $O(n)$ time since the total complexity of polygons is linear. The theorem follows.

5 Non-simple Paths.

If the paths are allowed to intersect themselves, see Fig. 8, an $O(kn)$ algorithm by Hershberger and Snoeyink [13] can be applied. As Efrat *et al.* [8] 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 *polylogn* time per vertex.

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.* [8] that shortcuts the paths using the simplex range search [6]. If two edges of a simple path cross they can be cut by applying shortcut test twice: before the crossing point and after, see Fig. 9. Clearly, the result of shortcuts is a set of x-monotone paths (possibly intersecting). The second phase is more tricky.

5.1 Implicit Funnel Algorithm. The problem is to find the shortest homotopic path in presence of obstacles. One can apply the algorithm from Lemma

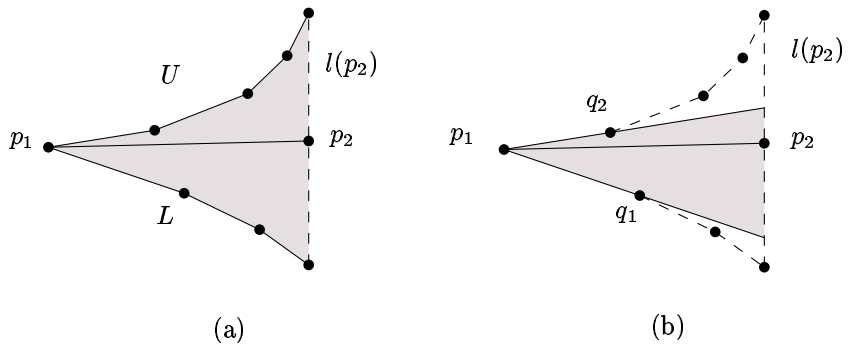

Fig. 10: (a) Funnel. (b) Implicit representation of funnel.

3.1. This leads to the factor of n to 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 $l(p_1)l(p)$. Let L be the lower envelope of B_L. The funnel is the area between U and L restricted by $l(p)$, see for example Fig. 10 (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. 10 (b).

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. 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. Using binary search on $L(p_1)$ 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. 10 (b).

Suppose that we have computed the implicit funnel F up to the vertical line through a point p_i. F is a triangle with vertices p_1, r_1 and r_2. Let q_1 and q_2 be the points of B defined the sides of F, see Fig. 13. 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 12. 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

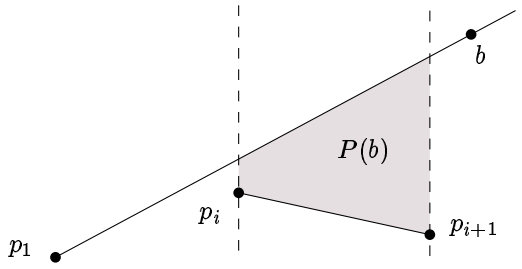

Fig. 11: Polygon $P(b)$.

(i) q_2', the lowest point (pont with the lowest slope p_1q_2') point visible from p_1 in the slab $l(p_i)l(p_{i+1})$ and above the segment p_ip_{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_ip_{i+1}.

The test included in the binary search for q_2' is whether, for a point $b \in B - \{p_1\}$, the polygon $P(b)$ bounded by the lines $l(p_i)$, $l(p_{i+1})$, p_ip_{i+1} and p_1b is empty, see Fig. 11. The polygon $P(b)$ is either a triangle or a trapezoid. A trapezoid can be partitioned into two triangles. Thus the test can be done using simplex range searching 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 their intersection. 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. 13. 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, assign $p_1 = q_1$ and $F = q_1r_1r_2$ and continue the computation of the funnel up to $l(p_{i+1})$.

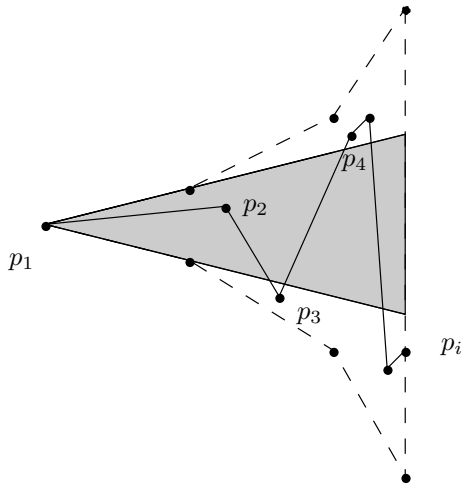

Fig. 12: The implicit funnel and the path $p_1p_2 \dots p_i$.

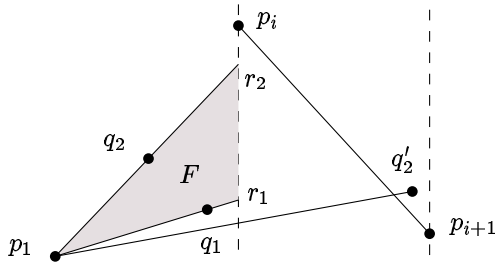

Fig. 13: Funnel collapse.

THEOREM 5.1. *The shortest homotopic paths of polylines can be computed in $O(n^{2+\varepsilon} + k_{in} \log^2 n)$ time.*

Proof. The preprocessing step includes

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

(ii) the construction of data structure for simplex searching [6],

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

These steps take $O(n^2 \log n)$, $O(n^{2+\varepsilon})$, and $O(k_{in} \log n)$ 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 the 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 times when we apply binary search. There are two ways of applying a binary search. We apply 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 for 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.

5.2 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 be dominated by the preprocessing time $O(n^{2+\varepsilon})$. In order to reduce the total running time we can use standard space-time tradeoff techniques. Matoušek [17] applied the technique to the range searching with efficient hierarchical cuttings and obtained $O(N^{2/3}M^{2/3})$ running time for processing M range queries against N points in the plane. There are two difficulties that need to be overcome if we apply this approach. One is that the preprocessing step of the algorithm from Theorem 5.1 takes $\Omega(n^2)$ time and space for computing the lists $L(b), b \in B$. The second difficulty is that the queries are not known beforehand. In the final version of the paper we show that the approach can still be applied.

THEOREM 5.2. *The shortest homotopic paths for paths that are not necessarily simple can be found in $O(n \cdot polylogn + n^{2/3}k^{2/3}polylogk + k \cdot polylogk)$ time.*

Acknowledgments. I would like to thank the anonymous referees for their valuable comments and the suggestion to apply the hierarchical cuttings.

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] 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`

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

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

[6] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9(2):145–158, 1993.

[7] 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.

[8] 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`

[9] 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.

[10] D. Grigoriev and A. Slissenko. Computing minimum-link path in a homotopy class amidst semi-algebraic obstacles in the plane. In T. Mora and H. F. Mattson, editors, *AAECC*, volume 1255 of *Lecture Notes in Computer Science*, pp. 114–129. Springer, 1997.

[11] D. Grigoriev and A. Slissenko. Polytime algorithm for the shortest path in a homotopy class amidst semi-algebraic obstacles in the plane. In *Proc. 9th Annu. Internat. Sympos. Algorithms Comput.*, pp. 17–24, 1998.

[12] 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.

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

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

[15] 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.

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

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

[18] 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.

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

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

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