Computing Homotopic Shortest Paths Efficiently*

Alon Efrat
Department of Computer Science
University of Arizona
alon@cs.arizona.edu

Stephen G. Kobourov†
Department of Computer Science
University of Arizona
kobourov@cs.arizona.edu

Anna Lubiw
Department of Computer Science
University of Waterloo
alubiw@uwaterloo.ca

January 27, 2003

Abstract

We give deterministic and randomized algorithms to find shortest paths homotopic to given collection Π of disjoint paths that wind amongst n point obstacles in the plane. Our deterministic algorithm runs in time $O(k_{out} + k_{in} \log n + n\sqrt{n})$, and the randomized algorithm runs in expected time $O(k_{out} + k_{in} \log n + n(\log n)^{1+\varepsilon})$. Here k_{in} is the number of edges in all the paths of Π, and k_{out} be the number of edges in the output paths.

*A preliminary version of this paper appeared in the *10th European Symposium on Algorithms (ESA)*, p.411-423, 2002.

†Partially supported by the NSF under grant ACR-0222920.

1 Introduction

Geometric shortest paths are a major topic in computational geometry; see the survey paper by Mitchell [16]. A shortest path between two points inside a simple polygon in the plane can be found in linear time using the "funnel" algorithm of Chazelle [4] and Lee and Preparata [14]. A more general problem is to find a shortest path between two points in a polygonal domain. In this case the "rubber band" solution is not unique, or, to put it another way, different paths may have different homotopy types. See definition below of homotopy. When the homotopy type of the solution is not specified, there are two main approaches, the visibility graph approach, and the continuous Dijkstra (or shortest path map) approach [16]. In this paper, we address the problem of finding a shortest path when the homotopy type is specified. Colloquially, we have a "sketch" of how the path should wind its way among the obstacles, and we want to shorten the path while preserving this sketch.

Homotopic shortest paths are used in VLSI routing [6, 12, 15]. A related problem is that of drawing graphs with "fat edges": given a planar weighted graph G, find a planar drawing such that all the edges are drawn as thickly as possible and proportional to the corresponding edge weights. Duncan *et al.* [8] and Efrat *et al.* [11] present an $O(kn + n^3)$ algorithm for this problem, where n is the number of edges and k is the maximum of their input and output complexities. Our paper improves this algorithm to roughly $O(n^3 + k)$.

Hershberger and Snoeyink [13] gave an algorithm for the homotopic shortest path problem. Their algorithm assumes a triangulation of size n of the polygonal domain, and finds a shortest path homotopic to a given path of k edges in time linear in k plus the number of triangles (counting repetition) visited by the input path. This can be nk in the worst case, and the purpose of our paper is to reduce it when output size permits.

Cabello, Liu, Mantler, and Snoeyink [3] consider the related problem of testing if two given paths are homotopically equivalent. Our original work [10] was done independently of theirs and the idea of the first step of the algorithms is the same. The current presentation of our work incorporates their more efficient implementation of this idea.

Bespamyatnikh's algorithm [2] improves the deterministic algorithm presented in section 4 to $O(n \log^{1+\varepsilon} n + k_{in} \log n + k_{out})$.

We now turn to a definition of homotopy, and a more precise description of the problem we solve. Let $\alpha, \beta : [0, 1] \longrightarrow \mathbb{R}^2$ be two continuous curves parameterized by arc-length. Then α and β are *homotopic* with respect to a set of obstacles $V \subseteq \mathbb{R}^2$ if α can be continuously deformed into β while avoiding the obstacles; more formally, if there exists a continuous function $h : [0, 1] \times [0, 1] \to \mathbb{R}^2$ with the following three properties:

1. $h(0, t) = \alpha(t)$ and $h(1, t) = \beta(t)$, for $0 \leq t \leq 1$

2. $h(\lambda, 0) = \alpha(0) = \beta(0)$ and $h(\lambda, 1) = \alpha(1) = \beta(1)$ for $0 \leq \lambda \leq 1$

3. $h(\lambda, t) \notin V$ for $0 \leq \lambda \leq 1, 0 < t < 1$

Let $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ be a set of disjoint, simple polygonal paths and let the endpoints of the paths in Π define the set T of at most $2n$ fixed points in the plane. Note that we allow a path to degenerate to a single point. We call the fixed points of T "terminals," and call the interior vertices of the paths "bends," and use "points" in a more generic sense, e.g. "a point in the plane," or "a point on a path." We assume that no two terminals/bends lie on the same vertical line.

Our goal is to replace each path $\pi_i \in \Pi$ by a shortest path σ_i that is homotopic to π_i with respect to the set of obstacles T; see Fig. 1 and Fig. 3. Note that σ_i is unique. Let $\Sigma = \{\sigma_1, \ldots \sigma_n\}$ be the set of resulting paths. Observe that these output paths may [self] intersect by way of segments lying on top of each other, but will be *non-crossing* in the sense that a slight perturbation of the bends makes the paths simple and disjoint.

Because we must always keep track of which side of a terminal a path goes on, we will find it convenient, both for pictures and for exposition, to regard each terminal as a small diamond. This actually changes the model of homotopy, because the endpoint of the path is now fixed at one corner of the diamond, and cannot swivel around the terminal. In other words, a path that begins at terminal t and spirals around t a few times before heading off is, with diamond terminals, no longer homotopic to the path that simply heads off. This is the only difference, however, and we can easily remove such spiraling from the final output.

Cabello *et al.* [3] point out the difference between the above two models, which they call the "pin" and "pushpin" models, respectively. They also describe the "tack" model, where the endpoints of the paths are not regarded as obstacles.

Let k_{in} be the number of edges in all the paths of Π. Let k_{out} be the number of edges in all the paths of Σ. We will measure the complexity of our algorithms in terms of n and $k = \max\{k_{in}, k_{out}\}$. Note that k may be arbitrarily large compared to n; for example, a path can wind around a set of terminals arbitrarily many times. Fig. 1 shows a less trivial example where k_{out} is $\Omega(2^n)$.

The algorithm of Hershberger and Snoeyink [13] finds homotopic shortest paths in time $O(nk_{in})$. Note that $k_{out} \leq nk_{in}$. The deterministic algorithm presented in this paper runs in time $O(k_{out} + k_{in} \log n + n\sqrt{n})$, and the randomized algorithm runs in expected time $O(k_{out} + k_{in} \log n + n(\log n)^{1+\varepsilon})$. These are improvements except when k is quite small compared to n. In this context, $\varepsilon > 0$ is an arbitrarily small parameter and the constant in the big-O notation depends on ε.

It is worth noting that our algorithm relies on the powerful algorithm of Bar-Yehuda and Chazelle [1] which uses linear time polygon triangulation and ideas of linear time Jordan sorting in order to find a trapezoidization of n disjoint polygonal chains with a total of k edges in time $O(k + n(\log n)^{1+\varepsilon})$. Replacing this by plane

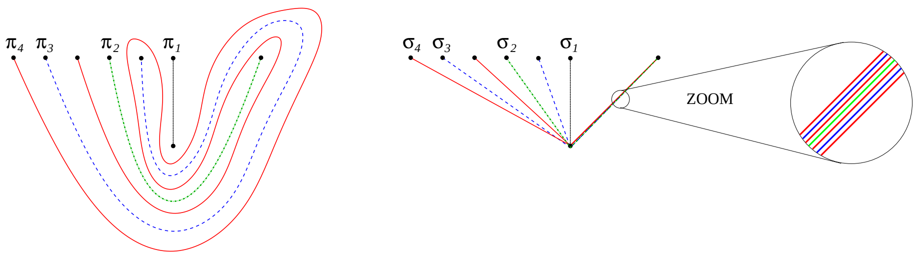

Figure 1: An example with exponential complexity: $k = \Omega(2^n)$: On the left is the initial wiring sketch, and on the right is the wiring after the shortest paths have been computed. The number of edge segments in the shortest paths $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ is $1, 2, 4, 8$, respectively. In general, wire σ_i has 2^{i-1} edge segments. Note that on the right many edge segments are parallel.

sweep makes our algorithm practical and yields a running time of $O(k_{out} + (n + k_{in}) \log(n + k_{in}))$ with an extra $O(n\sqrt{n})$ additive factor for the deterministic version.

The algorithm of Cabello *et al.* [3] to test whether two paths are homotopically equivalent has running time $O((n+k_{in}) \log(n+k_{in}))$ (they opt for the practical plane sweep approach). Their algorithm applies to the tack model of homotopies as well as to the pin and pushpin models. Note that, although an algorithm to find shortest homotopic paths can be used to test homotopic equivalence, there is a price to be paid, namely the size of the shortest paths, k_{out}.

In the remainder of this section we give the intuition and an outline of our algorithm. Although k_{in} can be arbitrarily large compared to n, one easily forms the intuition that, because the paths are simple and disjoint, k can be large in a non-trivial way only because path sections are repeated over and over. For example, a path may spiral arbitrarily many times around a set of points, but each wrap around the set is the same.

Our method makes essential use of this observation. We do not begin by explicitly searching for repeated path sections—this seems difficult modulo homotopic equivalence. Instead we begin in section 2 by applying vertical shortcuts to the paths (homotopically) so that each left and each right local extreme point occurs at a terminal. These terminals must then be part of the final shortest paths, and we have decomposed the paths into "essential" x-monotone pieces with endpoints at terminals. This idea is independently developed by Cabello *et al.* [3]. In section 3 we argue that the number of homotopically distinct x-monotone pieces is at most $2n$. We will bundle together all the homotopically equivalent pieces. Routing one representative from each such bundle using the straightforward "funnel" technique takes $O(k_{in}+n^2)$ time total. In section 3 we reduce this using a "shielding technique" where we again exploit the fact that the paths are disjoint and use the knowledge gained in routing one shortest path to avoid repeating work when we route subsequent paths. The final step of the algorithm is to unbundle, and recover the final paths by putting together

Main Algorithm

1. shortcut paths to divide into essential monotone pieces
2. bundle homotopically equivalent pieces
3. find the shortest path for each bundle
4. unbundle to recover final paths

Figure 2: Summary of the algorithm.

the appropriate pieces. This is straightforward, and we say no more about it. We summarize the main steps of the algorithm in Fig. 2.

2 Shortcutting to Find Essential Monotone Pieces

We begin by applying vertical shortcuts to reduce each path to a sequence of essential x-monotone path sections; see Fig. 3. A *vertical shortcut* is a vertical line segment ab joining a point a on some path π with a point b also on π, and with the property that the subpath of π joining a and b, π_{ab}, is homotopic to the line segment ab.

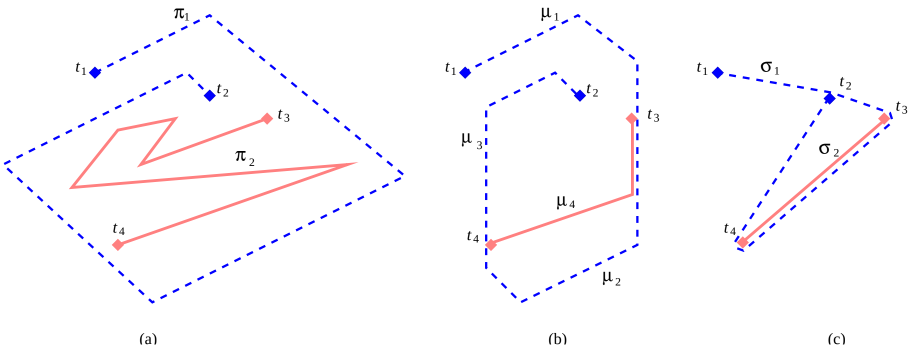

Figure 3: (a) Two paths π_1 and π_2 joining terminals t_1 to t_2 and t_3 to t_4, respectively. (b) The two paths after performing vertical shortcuts. Note that path π_1 now consists of 3 monotone pieces: μ_1 from t_1 to t_3, μ_2 from t_3 to t_4, and μ_3 from t_4 to t_2; path π_2 now consists of one x-monotone piece, μ_4, homotopically equivalent to μ_2; (c) The final homotopically equivalent shortest paths, σ_1 and σ_2.

We will only do *elementary vertical shortcuts* where the subpath π_{ab} consists of [portions of] 2 line segments or 3 line segments with the middle one vertical, and the other two non-vertical. We distinguish *left shortcuts* which are elementary vertical shortcuts where π_{ab} contains a point to the left of the line through ab; *right shortcuts* where π_{ab} contains a point to the right of the line through ab; and *collinear shortcuts*

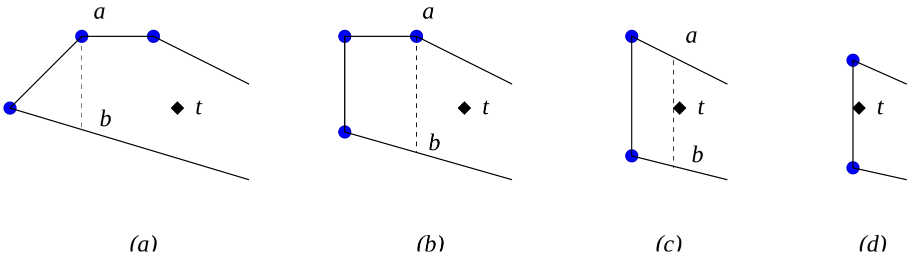

Figure 4: Maximal left shortcuts (bends are represented by circles and the terminals t by a diamond): (a) a 2 segment shortcut, maximal because a is a bend; (b) a 3 segment shortcut, maximal for the same reason; (c) a 3 segment shortcut, maximal because ab hits a terminal; (d) the result of the shortcut in (c).

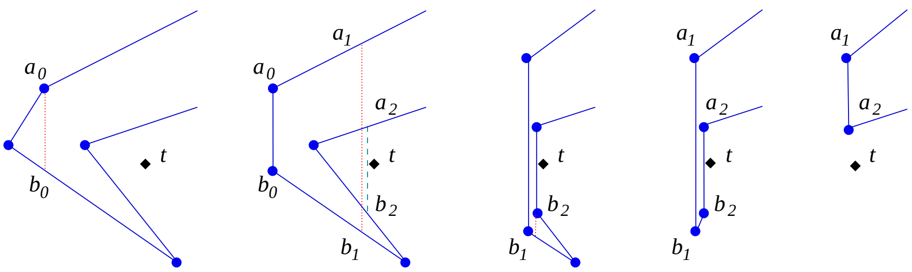

Figure 5: A sequence of maximal elementary vertical shortcuts (bends are represented by circles and the terminal t by a diamonds). Shortcuts a_0b_0, a_1b_1 and a_2b_2 are left shortcuts; b_2b_1 is a right shortcut; and a_2b_1 and a_1a_2 are collinear shortcuts.

where π_{ab} lies in the line through ab; see Fig. 4 and Fig. 5. Collinear shortcuts are always applied after left/right shortcuts (and only then), and eliminate consecutive vertical segments.

We will in fact only apply *maximal* elementary vertical shortcuts, where the subpath π_{ab} cannot be increased. In particular, this means that for left and right shortcuts, either a or b is a bend, or the line segment ab hits a terminal.

A *local left [right] extreme* of a path is a point or, more generally, a vertical segment, where the x-coordinate of the path reaches a local minimum [maximum]. Observe that every left [right] shortcut "cuts off" a local left [right] extreme. Conversely, every left [right] local extreme provides a left [right] shortcut *unless* the local extreme *is locked at* a terminal, meaning that the left [right] extreme contains the left [right] point of the terminal's diamond. The first two pictures in Fig. 5 show left extremes that provide left shortcuts; the middle picture shows left extremes locked at a terminal.

To detect the maximal left shortcut for a local left extreme, l, we use range searching. Let the segments preceding and following l along the path be l_p and l_f.

Let their right endpoints be $r(l_p)$ and $r(l_f)$, and let t be the leftmost of these two points. Then t determines the maximal potential shortcut that can be performed at l in the absence of terminals. We must query the triangle/trapezoid bounded by l, l_p, l_f and the vertical line through t to find the leftmost terminal inside it. This determines the actual maximal left shortcut that can be performed at l. As described in section 2.2, we use an idea from Cabello *et al.* [3] to reduce the range queries to orthogonal range queries, which can be answered more efficiently.

We perform elementary vertical shortcuts until none remain, at which time all local left and right extremes are locked at terminals. We claim that the final shortest paths must be locked in the same way at the same terminals, so we have divided the paths into their *essential monotone pieces*.

Claim 2.1 *Let π be a path, and let μ be the result of performing elementary vertical shortcuts on π until no more left nor right shortcuts 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.*

Proof: Because π and μ are homotopic, σ is the shortest path homotopic to μ. We can thus go from μ to σ using "rubber band" deformations that only shorten the path, and such deformations cannot loosen a left [right] extreme from the terminal it is locked at.

■

Doing elementary vertical shortcuts in an arbitrary order may result in crossing paths; see Fig. 6. To guarantee non-crossing paths we do the elementary vertical shortcuts in alternating *left* and *right phases*. Throughout the algorithm we maintain the sets L and R of local left [right] extremes not locked at terminals. A left phase continues until L is empty. At each step we remove a local left extreme from L, perform the maximal left shortcut there, perform the consequent collinear shortcuts, and update the sets L and R. The new shortcut may destroy one or two elements of R, or create one new element of R or L. These changes occur only at the newly added shortcut and are easy to detect. It is crucial for the correctness of this approach that a left shortcut does not destroy any local left extremes, nor change the shortcuts that will be performed at them.

Observe that more than one left and one right phase may be required, since the right phase may add new members to L; see Fig. 7.

Two things remain: to prove that the final paths do not cross, and to consider the implementation and analysis of the algorithm. We devote a subsection to each.

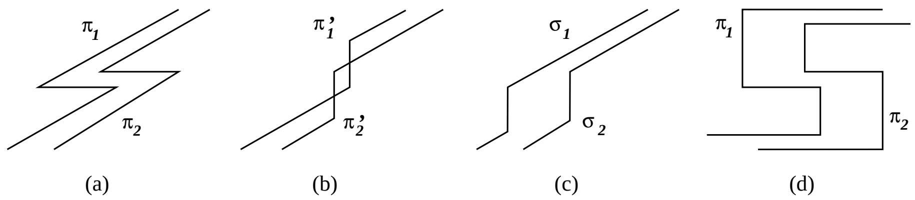

Figure 6: Performing shortcuts in arbitrary order may result in crossings. (a) Disjoint paths π_1 and π_2; (b) Crossing after a left shortcut in π_1 and a right shortcut in π_2. No further shortcuts are possible, hence the crossing cannot be removed; (c) Disjoint paths output by step 1 after two right shortcuts; (d) Rectified version of (a).

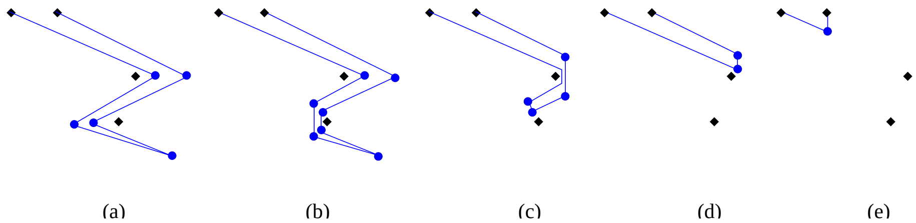

Figure 7: A path through alternating left and right shortcutting phases: (a) the original path; (b) after a left phase; (c) after a right phase; (d) after a left phase; (e) after a right phase. Terminals are represented by diamonds

2.1 Non-crossing paths

Lemma 2.2 *Each left [right] phase of shortcuts preserves the property that paths are non-crossing.*

It is crucial for steps 2 and 3 of our algorithm that the paths remain non-crossing. By symmetry, we can concentrate on a left phase. Note that a left phase, as described, is non-deterministic. At each stage we choose one member l from the set L of current local left extremes, and use it to perform a left shortcut. To prove Lemma 2.2 we will first show that for each left phase there is *some* sequence of choices that preserves the property that the paths are non-crossing. We will then argue that the end result of a phase does not depend on the choices made. The correctness of Claim 2.3 and Lemma 2.4 below imply the proof of Lemma 2.2

We say that a sequence of left shortcuts is done in a *rightmost order* if the shortcut chosen at each iteration has its locally leftmost point to the right of the locally leftmost point of any other possible shortcut.

Claim 2.3 *Performing left shortcuts in rightmost order leaves the paths non-crossing.*

Observe that we could complete a left phase of shortcuts using this "rightmost"

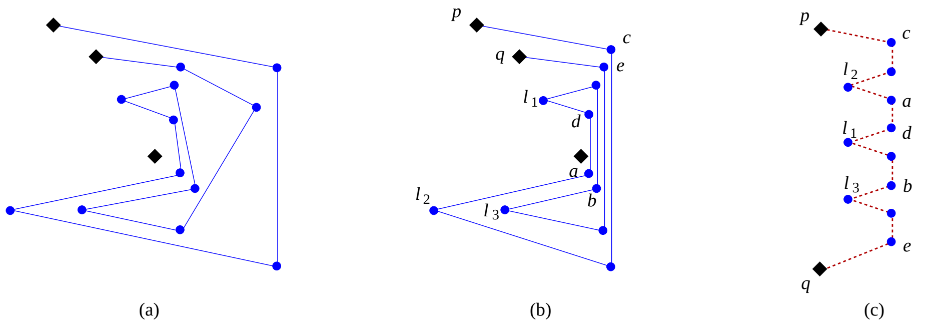

Figure 8: (a) A given path undergoes a right phase; (b) Handling local left extreme l_1 first yields the shortcut ab. Handling the local left extreme l_2 first yields shortcut cd. The final result of handling l_1, l_2 and l_3 in any order is shortcut ce; (c) An "unfolded" version of part (b).

order, at a cost of an extra $\log k_{in}$ factor for maintaining a heap. There is no practical advantage to doing this.

Lemma 2.4 *The end result of a left phase does not depend on the sequence of choices made during the phase.*

Proof: We begin with the claim that the set $\mathcal{L}$ of all local left extremes that appear in L over the course of the phase is independent of the choices made during the phase.

This implies that the set of left shortcuts performed during the course of the phase is also independent of the choices made during the phase. However, the set of collinear shortcuts is *not* independent. In particular, the order in which we perform left shortcuts affects the set of collinear shortcuts; see Fig. 8.

Consider one left phase. Let L_0 be the initial set of local left extremes not locked at terminals. Let $\mathcal{L}$ be the union of L over the course of the phase. Suppose two sequences of choices C_1 and C_2 during a left phase yield sets $\mathcal{L}_1$ and $\mathcal{L}_2$. We would like to show that $\mathcal{L}_1 = \mathcal{L}_2$. Consider $l \in \mathcal{L}_1$. We will prove $l \in \mathcal{L}_2$ by induction on the number of left shortcuts performed in the phase before l enters $\mathcal{L}_1$. If this number is 0 then $l \in L_0$ and we are done. Otherwise, l is added to L as a result of some left shortcut and consequent collinear shortcuts. Any vertical segment used in the collinear shortcuts may, in its turn, have arisen as a result of some left shortcut and consequent collinear shortcuts. Tracing this process, we find that l is formed from a set of left shortcuts linked by vertical segments, all in the same vertical line as l; see Fig. 8. All these left shortcuts arose from local left extremes that entered $\mathcal{L}_1$ before l did, and thus, by induction, are in $\mathcal{L}_2$. They only leave $\mathcal{L}_2$ when we perform their shortcuts, and thus the choice sequence C_2 includes these shortcuts—though possibly in a different order than in C_1. The consequent collinear shortcuts will merge all the verticals forming l, and cannot merge more than that because the segments attached before and after l are not vertical (l is a local left extreme). Thus l is in $\mathcal{L}_2$.

This proves that the set of local left extremes, and thus the set of left shortcuts is independent of choices made during the phase. Any vertical segment that is in the final set of paths output by the phase arises through left shortcuts plus consequent collinear shortcuts. Since any set of choices leads to the same set of left shortcuts, though possibly in different order, the consequent collinear shortcuts will arrive at the same final vertical shortcuts—i.e. the same final paths.

■

2.2 Implementation and run-time analysis

We use range queries to identify the maximal shortcut that can be performed at a local extreme of the path. Our original version of this paper [10] used general simplex range searching, which is time-consuming. Cabello *et al.* [3], in their paper on testing homotopy, independently developed this same idea of shortcutting to divide a path into essential monotone pieces. However, they were able identify shortcuts using *orthogonal* range queries, which can be done much more efficiently. We briefly describe this clever idea.

The main insight is that shortcuts depend only on the *aboveness* relation among terminals and x-monotone path sections, and that the aboveness relation forms a partial order. This partial order can be preserved while each monotone path section is modeled as a horizontal line segment, and the whole picture becomes "rectified"; see Fig. 6(d). Shortcuts can now be identified using orthogonal range queries.

We find the aboveness relation from a trapezoidization of the input, which can be computed in time $O(k_{itin} + n(\log n)^{1+\varepsilon})$ using the algorithm of Bar-Yehuda and Chazelle [1]. The partial order can be extended to a total order in linear time. We now assign to each terminal and x-monotone path section, a new y-coordinate equal to its rank in the total order. As justified in [3], shortcuts after this transformation correspond to shortcuts in the original.

Each shortcut now requires an orthogonal range query: given an axis-aligned query rectangle, certify that it is empty or else return the rightmost [leftmost] terminal inside it. Chazelle's data structure for segment dragging [5] solves this with $O(n \log n)$ preprocessing time, $O(n)$ space, and $O(\log n)$ query time. Simpler algorithms require an extra logarithmic factor in space [9, 17].

We need one more ingredient to bound the running time of step 1 of our algorithm by $O(k_{in} \log n + n(\log n)^{1+\varepsilon})$.

Claim 2.5 *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 $2k_{in}$.*

Proof: Assume that no two terminals and/or bends line up vertically. Consider the set of vertical lines through bends and through the left and right sides of each terminal's diamond. An elementary shortcut operates between two of these vertical lines. If a left [right] shortcut has its leftmost [rightmost] vertical at a bend, then after the shortcut, this vertical disappears forever; see Fig. 5. There are thus at most k_{in} such elementary shortcuts. Consider, on the other hand, a left shortcut that has its leftmost vertical at a terminal. This only occurs when two previous left shortcuts are stopped at the terminal, and then combined in a collinear shortcut. See the right hand pictures of Fig. 5. Thus an original edge of the path has disappeared. Note that elementary shortcuts never fragment an edge of the path into two edges, but only shorten it from one end or the other. Thus there are at most k_{in} elementary shortcuts of this type. Altogether, we obtain a bound of $2k_{in}$ elementary shortcuts. ∎

Hence we observe that Step 1 of the algorithm runs in time $O(k_{in} \log n + n(\log n)^{1+\varepsilon})$.

3 Bundling Homotopically Identical Paths

Let $M = \{\mu_1 \ldots \mu_m\}$ be the set of essential x-monotone paths obtained from step 1 of our algorithm. In the second step of the algorithm we bundle homotopically equivalent paths in M. More precisely, we take one representative path for each equivalence class of homotopically equivalent paths in M. This is justified because the paths in each equivalence class have the same homotopic shortest path. Because the paths in M are non-crossing and x-monotone, it is easier to detect homotopic equivalence: two paths are homotopically equivalent if they have the same endpoints, and, between these endpoints no terminal lies vertically above one path and vertically below the other.

In order to perform the bundling we use a trapezoidization of M, see Fig. 9. We apply the trapezoidization algorithm of Bar-Yehuda and Chazelle [1] to the paths obtained after the shortcuts of step 1, but before these paths are chopped into monotone pieces.

Claim 3.1 *We can perturb the paths output from step 1 so that they become disjoint and the Bar-Yehuda and Chazelle algorithm can be applied.*

Constructing the trapezoidization requires time $O(k_{in} + n(\log n)^{1+\varepsilon})$. Once we have a trapezoidization of M, we can in linear time bundle homotopically equivalent paths as follows. While scanning each $\mu \in M$, we check if it is homotopically equivalent to the path "below" it, by examining all the edges of the trapezoidization that are incident to μ from below. If all these trapezoidization edges reach the same path μ_j and none pass through a terminal on the way to μ_j, and μ_i and μ_j have the same terminals as endpoints, then we mark μ_i as a duplicate. Let $R = \{\rho_1 \ldots \rho_r\}$ be the paths of M that are not marked as duplicates.

Lemma 3.2 *The number of paths in R is bounded by $2n$.*

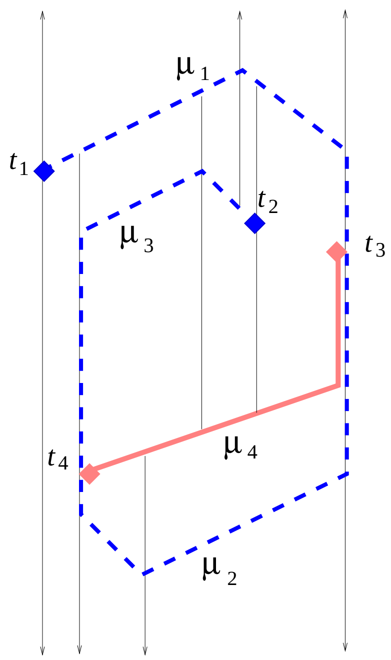

Figure 9: A trapezoidization enables us to identify that μ_2 and μ_4 are homotopically equivalent.

Proof: Note that every terminal t is either a right endpoint of paths in R or a left endpoint of paths in R, but not both. This is because a terminal cannot have local left extremes and local right extremes locked at it without the paths crossing.

For each terminal $t \in T$, associate its bottommost incident path $\phi(t)$ (its "floor"), and the first path hit by a vertical ray going up from t, $\gamma(t)$ (its "roof"). We claim that every path in R is $\phi(t)$ or $\gamma(t)$ for some $t \in T$. This proves that the number of paths in R is at most $2n$.

Consider a path $\rho \in R$ with left and right terminals s and t, respectively. Suppose that ρ is not a roof. Then every vertical ray extending downward from a point of ρ must hit the same path σ. (If two rays hit different paths, then in the middle some ray must hit a terminal.) Furthermore, since ρ is not the bottommost path incident to s, σ must hit s. Similarly σ must hit t. But then σ and ρ are homotopically equivalent.

■

4 Shortest Paths

In this section we find shortest paths homotopic to the monotone paths $R = \{\rho_1, \ldots, \rho_r\}$ produced in the previous section. Note that r is $O(n)$. Let ρ_i' denote the shortest path homotopic to ρ_i. We route each path using a funnel technique. The funnel algorithms [4, 14] operate on a triangulation of the n points (the terminals in our

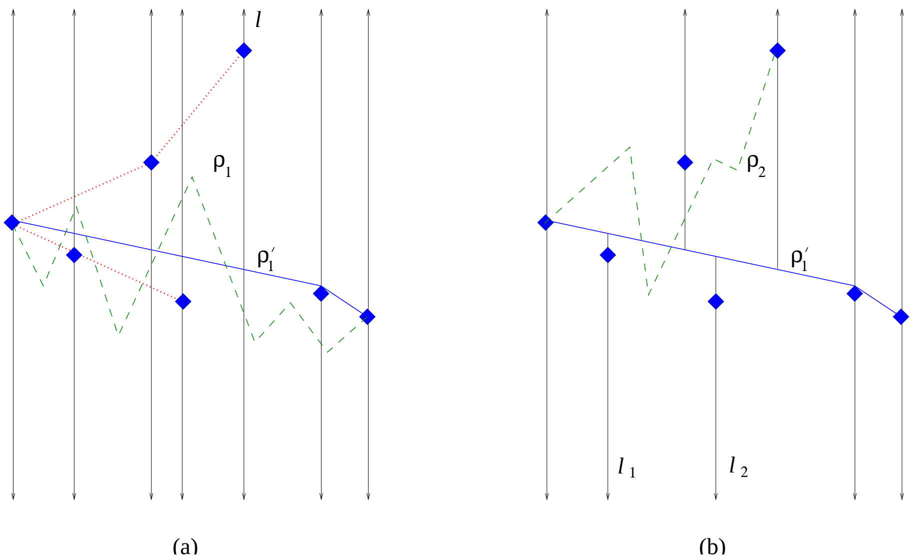

Figure 10: (a) A path ρ_1 (dashed), its funnel up to line l (dotted) and the final homotopic shortest path ρ_1'; (b) After the shortest path ρ_1' has been found, shielding allows us to route another path ρ_2 without examining vertical lines l_1 and l_2.

case), and follows the path through the triangulation maintaining a current "funnel" containing all possible shortest paths thus far. The algorithm takes time proportional to the number of edges in the path plus the number of intersections between the triangulation edges and the path.

Rather than a triangulation, we will use a trapezoidization formed by passing a vertical line through each of the n terminals; see Fig. 10(a). Then, since each path ρ_i is x-monotone, it has $O(n)$ intersections with trapezoid edges, and the funnel algorithm takes time proportional to n plus the number of edges in ρ_i. Since the total number of edges in R is bounded by k_{in}, this gives a total over all paths of $O(n^2 + k_{in})$. (Note that the number of edges in the output is $O(n^2)$, so it doesn't appear explicitly here.)

We will improve this by making use of the fact that the paths are disjoint, and re-using information gained when each path is routed. We give a randomized algorithm to route the $O(n)$ monotone paths of R in time $O(n \log n + k)$, and a deterministic algorithm with expected $O(n\sqrt{n} + k)$ running time. Recall that $k = k_{in} + k_{out}$.

Both methods use a "shielding" technique. We begin by describing this idea intuitively. First note that the ρ_i's can be routed independently, since none affects the others. The initial paths ρ_i are non-crossing, and so are the final shortest paths, ρ_i'. If ρ_j is below ρ_i, and we have already computed ρ_j', then ρ_j' behaves as a barrier that "shields" ρ_i from terminals that are vertically below ρ_j'; see Fig. 10(b).

To utilize shielding we will modify the basic trapezoidization described above as we discover shortest paths ρ_i'. In particular, the upward vertical ray through terminal

t, $u(t)$, will be truncated at the lowest shortest path ρ'_i that is strictly above t and does not bend at t. The downward vertical ray through terminal t, $d(t)$, will be truncated in an analogous way.

The shortest paths found so far, together with the truncated vertical rays $u(t)$ and $d(t)$ for each terminal t, partition the plane into trapezoids, each bounded from left and right by the vertical rays, and from above and below by shortest paths. To route a new path ρ_i through this modified trapezoidization we use the following algorithm.

Routing ρ_i with shielding

1. Identify the first trapezoid that ρ'_i will traverse. This can be done in $O(\log n)$ time because the shortest paths observe the same vertical ordering as the original ρ_j's.

2. Traverse from left to right the sequence of trapezoids that ρ'_i will pass through. (Note that ρ_i itself may pass through different trapezoids; see Fig. 10(b).) We construct the funnel for ρ'_i as we do this traversal. Suppose that our path enters trapezoid τ. To leave τ on the right we have two cases. If the right side of τ is a point, then it is a terminal t, and we are locked between two paths that terminate or bend at t. Then the funnel collapses to this point, and we proceed to the next trapezoid if ρ_i continues. Otherwise the right side of τ is a vertical through some terminal t, and (unless ρ_i ends at t) we have a choice of two trapezoids to enter, the upper one with left side $u(t)$ or the lower one with left side $d(t)$. We follow path ρ_i until it crosses the infinite vertical line through t. If it passes above t then we enter the upper trapezoid, and otherwise we enter the lower trapezoid. We update the funnel to include t.

3. When we reach the right endpoint of ρ_i, the funnel gives us the shortest homotopic path ρ'_i.

4. We update the trapezoidization as follows. For each vertical segment $u(t)$ or $d(t)$ that is intersected by ρ'_i we chop the segment at its intersection point with ρ'_i, provided that the intersection point is not t (i.e. that ρ'_i does not bend at or terminate at t).

Without yet discussing the order in which we route the paths, we can say a bit about the timing of this shielding method. As we traverse a path ρ_i we spend time proportional to the size of ρ_i plus the number of trapezoids traversed by ρ'_i. When ρ'_i leaves a trapezoid, say at the line segment $u(t)$ above terminal t, it may happen that ρ'_i will bend at t or terminate at t. In this case t is part of the output path ρ'_i, and we can charge the work for this trapezoid to the output size k_{out}. If, on the other hand, ρ'_i does not bend or terminate at t, then it crosses $u(t)$ and we chop $u(t)$ there. In this case we charge the work for this trapezoid to the chop. Thus the total time spent by the algorithm is $O(k + C)$ where C is the total number of chops performed at the n verticals.

For shielding to be effective we need to route paths in an order that makes C grow more slowly than n^2. We first analyze the randomized algorithm where we choose the next path to route at random with uniform probability from the remaining paths.

Claim 4.1 *If paths are routed in random order then the expected number of times we chop a segment $u(t)$ or $d(t)$ is $O(\log r)$, and thus C is $O(n \log n)$, and the routing takes time $O(k + n \log n)$.*

Proof: This follows from a standard backward analysis. See for example [7] for other proofs along these lines. Let l be a vertical line through a point t, and let u denote a ray emerging vertically from t. Assume that m paths intersecting u have been inserted up to now, and we are about to insert a new one. Then u will be chopped if and only if the new path creates an intersection point below all existing intersection points on u, but above t itself. Since the order of the insertions is random, the probability of the new intersection point being below all other intersection points is $1/m$. Summing over all paths yields the claimed bound.

■

Finally, we describe how to achieve a routing time of $O(k+n\sqrt{n})$ deterministically. Suppose that the paths $\{\rho_1, \ldots, \rho_r\}$ are in order from top to bottom. (We get this for free from step 2 of our algorithm.) Partition the paths into blocks $B_1, \ldots, B_{\sqrt{r}}$ each of size $\sqrt{r}$, and route them one block at a time from $B_{\sqrt{r}}$ to B_1. Within each block we process the paths in order. The largest increasing [decreasing] sequence with this ordering has size $\sqrt{r}$, therefore the number of chops at each vertical is $\sqrt{r}$. Thus C is $O(n\sqrt{n})$, and this routing method takes time $O(k + n\sqrt{n})$.

5 Conclusion and Open Problems

Given n terminal points in the plane, and a set of disjoint paths joining pairs of the terminals, we showed in this paper how to find shortest paths homotopic to the originals in time $O(k \log n + n\sqrt{n})$, or with randomization, in expected time $O(k \log n + n(\log n)^{1+\varepsilon})$, where k is the sum of input and output sizes of the paths. Unless k grows quite slowly compared to n, this is better than the Hershberger-Snoeyink algorithm which runs in time $O(nk)$.

An interesting open problem is to extend our efficient shortest homotopic path algorithm to the "tack" model, as described in [3]. Under this model the endpoints of paths need not be obstacles; they act as tacks that are pushed all the way into the backing, allowing the rubber bands of the paths to pass freely over them. Cabello *et al.* successfully use the shortcut technique to find the essential x-monotone pieces so they can test equivalence of two paths under the tack model. However, the problem of finding shortest paths seems more difficult, because the final shortest paths will cross in general, and the later steps of our algorithm fail.

Acknowledgments

We would like to thank S. Cabello, J. Snoeyink for pointing us to the possibility of using an orthogonal range searching data structure. We would also like to thank the anonymous referees for their helpful ideas and comments.

References

[1] R. Bar-Yehuda and B. Chazelle. Triangulating disjoint Jordan chains. *International Journal of Computational Geometry & Applications*, 4(4):475–481, 1994.

[2] S. Bespamyatnikh. Computing Homotopic Shortest Paths in the Plane. In *14th ACM-SIAM Symposium on Discrete Algorithms,* pages 609-617, 2003.

[3] S. Cabello, Y. Liu, A. Mantler, and J. Snoeyink. Testing homotopy for paths in the plane. In *18th Annual Symposium on Computational Geometry*, pages 160–169, 2002.

[4] B. Chazelle. A theorem on polygon cutting with applications. In *23rd Annual Symposium on Foundations of Computer Science*, pages 339–349, 1982.

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

[6] R. Cole and A. Siegel. River routing every which way, but loose. In *25th Annual Symposium on Foundations of Computer Science*, pages 65–73, 1984.

[7] M. de Berg, M. van Kreveld, M. H. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, 2nd edition, 2000.

[8] C. A. Duncan, A. Efrat, S. G. Kobourov, and C. Wenk. Drawing with fat edges. In *9th Symposium on Graph Drawing* , pages 162–177, September 2001.

[9] H. Edelsbrunner. A note on dynamic range searching. *Bulletin of the European Association for Theoretical Computer Science*, 15:34–40, Oct. 1981.

[10] A. Efrat, S. Kobourov, and A. Lubiw. Computing homotopic shortest paths efficiently. Technical report, April 2002. `http://www.arXiv.org/abs/cs/0204050`.

[11] A. Efrat, S. G. Kobourov, M. Stepp, and C. Wenk. Growing fat graphs. Video Presentation. In *18th Annual Symposium on Computational Geometry*, pages 277–278, 2002.

[12] S. Gao, M. Jerrum, M. Kaufmann, K. Mehlhorn, W. Rülling, and C. Storb. On continuous homotopic one layer routing. In *4th Annual Symposium on Computational Geometry*, pages 392–402, 1988.

[13] J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. *Computational Geometry: Theory and Applications*, 4:63–97, 1994.

[14] D. T. Lee and F. P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14(3):393–410, 1984.

[15] C. E. Leiserson and F. M. Maley. Algorithms for routing and testing routability of planar VLSI layouts. In *17th Annual ACM Symposium on Theory of Computing*, pages 69–78, 1985.

[16] J. S. B. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*. 1998.

[17] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1989.