Touring a Sequence of Polygons

July 30, 2025

Abstract

Given a sequence of k polygons in the plane, a start point s, and a target point, t, we seek a shortest path that starts at s, visits in order each of the polygons, and ends at t. If the polygons are disjoint and convex, we give an algorithm running in time $O(kn \log(n/k))$, where n is the total number of vertices specifying the polygons. We also extend our results to a case in which the convex polygons are arbitrarily intersecting and the subpath between any two consecutive polygons is constrained to lie within a simply connected region; the algorithm uses $O(nk^2 \log n)$ time. Our methods are simple and allow shortest path queries from s to a query point t to be answered in time $O(k \log n + m)$, where m is the combinatorial path length. We show that for nonconvex polygons this "touring polygons" problem is NP-hard.

The touring polygons problem is a strict generalization of some classic problems in computational geometry, including the safari problem, the zoo-keeper problem, and the watchman route problem in a simple polygon. Our new results give an order of magnitude improvement in the running times of the safari problem and the watchman route problem: We solve the safari problem in $O(n^2 \log n)$ time and the watchman route problem (through a fixed point s) in time $O(n^3 \log n)$, compared with the previous time bounds of $O(n^3)$ and $O(n^4)$, respectively.

1 Introduction

A number of problems in computational geometry—including the zookeeper problem, the safari problem, and the watchman route problem—involve finding a shortest path, within some constrained region, that visits a set of convex polygons in an order that is given, or can be deduced. When region constraints permit it, a locally shortest path may travel straight through a polygon; otherwise, it "reflects" off edges or vertices of the polygons. The three problems mentioned above have in common that a path is globally shortest if it is locally shortest—i.e., it "reflects" appropriately whenever it changes direction.

In this paper we give the first polynomial-time algorithm for the general touring polygons problem: to find a shortest path, between two specified points, that visits in order a sequence of k possibly intersecting convex polygons while lying in some constrained regions. We crucially use the property that local optimality of a path implies global optimality. One consequence is that we never need to compute distances; we only need to compute reflections of points with respect to lines. If the given polygons are not convex, this property fails, and we prove that the touring polygons problem is NP-hard.

Many geometric shortest path problems, e.g., shortestpaths among obstacles, are solved using the general technique of computing a shortest path map with respect to a fixed starting point s, which is a subdivision of the plane into regions such that the shortest paths from s to all points t in one region are combinatorially equivalent. One cannot directly apply this technique to our problem since the number of combinatorially distinct shortest paths in the touring polygons problem can be exponential in the input size, as we show in Theorem 3. We use instead a new technique based on subdividing the plane only according to the combinatorial type of the last step of the path. We call this a last step shortest path map. Our algorithms are based on computing the last step shortest path maps iteratively, one for each of the k polygons that must be visited, in the order they are given. Given the sequence of such maps, a query for the shortest path from s to any query point t can be answered in time $O(k \log(n/k))$: a query for t in the last shortest path map tells us whether the path to t goes through, or reflects from P_k, thus determining a query point for the previous shortest path map.

The general touring polygons problem (TPP) is as follows. We are given a start point $s = P_0$, a target point $t = P_{k+1}$, and a sequence of polygons, $P_1, \ldots, P_k$, possibly intersecting, each considered to be a closed region in the plane (i.e., each includes interior plus boundary). For each $i = 0, \ldots, k$, we are given a simply connected polygonal region F_i, called a "fence", with F_i containing P_i and P_{i+1}. We say that a path π_i visits polygon P_i if $\pi_i \cap P_i \neq \emptyset$. We say that π_i visits the sequence $P_1, \ldots, P_k$ if there exist points $q_1 \in P_1, q_2 \in P_2, \ldots, q_k \in P_k$ such that $q_1, \ldots, q_k$ appear in order along π_i. We let $p_0 = s$ and let p_i denote the first point of π_i (i.e., the point closest to s along the path) that lies within P_i and comes after p_{i-1} along π_i; such points exist if π_i visits the sequence $P_1, \ldots, P_k$. Our goal is to compute a shortest path that begins at s, visits $P_1, P_2, \ldots, P_k$, in that order, and arrives at t, with each subpath of π_i from p_{i-1} to p_i lying within the fence F_{i-1}. In the unconstrained TPP, each of the fences F_i is the entire plane, $\mathbb{R}^2$.

An i-path is a path that starts at s and visits the sequence of polygons $P_1, \ldots, P_i$. A fenced i-path to a point $q \in F_i$ is an i-path satisfying the relevant fence constraints, i.e., having the subpath from p_{j-1} to p_j lying within F_{j-1}, for $j = 1, \ldots, i$ and the subpath from p_i to q in F_i. Using this notation, the TPP asks for a shortest fenced k-path to t. For a point x, let $\pi_i(x)$ denote a shortest fenced i-path that ends at x. We let n denote the total number of vertices in the polygons $P_1, \ldots, P_k$ and the fences $F_0, \ldots, F_k$.

The formulation of the TPP allows for both the cycle and the path versions of the problem: If the destination point t is the same as the starting point s, we are computing a shortest cycle ("tour") through s visiting the P_i's. If the order in which the polygons P_i must be visited is not specified, then the touring polygons problem becomes the Traveling Salesperson Problem with Neighborhoods, which is NP-hard. See [20].

The touring polygons problem can be modeled as a special kind of 3-dimensional shortest path problem among polyhedral obstacles. Imagine k very large sheets of paper stacked up in parallel planes orthogonal to the z-axis. The i-th sheet has a hole, P_i, cut out of it; each sheet of paper is a polyhedral obstacle. The point s is placed at a z-coordinate just below the first sheet and the point t at a z-coordinate just above the kth sheet. Projecting this configuration to the (x, y) plane yields the original touring polygons input. A solution to the touring polygons problem is the projection of a shortest path in 3-dimensions that starts from s, threads its way through the polygonal holes, and arrives at t.

1.1 Summary of Results

- We give an $O(kn \log(n/k))$ time ($O(n)$ space) algorithm for the unconstrained TPP with disjoint convex polygons P_i. No polynomial-time algorithm was previously known for this problem. For fixed s, the data structure supports $O(k \log(n/k))$-time shortest path queries to t.

- We give an $O(nk^2 \log n)$ time ($O(nk)$ space) algorithm for the general TPP with possibly intersecting convex polygons P_i and arbitrary fences F_i. Actually, we only require convexity of the part of P_i's boundary from which fenced rays may bounce.

- We show that the full combinatorial shortest path map for the TPP has worst-case size $\Theta((n-k)2^k)$. It can be computed in output-sensitive time and supports $O(k+\log n)$-time shortest path queries.
- We show that the unconstrained TPP is NP-hard in general, if the polygons P_i are allowed to be nonconvex. A fully polynomial time approximation scheme follows from the application of results for shortest paths among obstacles in $\mathbb{R}^3$.
- We give substantional improvements in the running time for two classical problems in computational geometry, namely (a)
- We give substantial improvements in the running time for two classical problems in computational geometry, namely (a) the safari problem, for which our results imply an $O(n^2 \log n)$ algorithm (improving upon $O(n^3)$); and (b) the (fixed-source) watchman route problem, for which our results imply an $O(n^3 \log n)$ algorithm (improving upon $O(n^4)$). (Our results imply also an improved bound - $O(n^4 \log n)$ instead of $O(n^5)$ - for the "floating" watchman route problem [26].) One of the main significances of our new method is not just that it yields a substantially faster algorithm for the watchman route problem, but also that it avoids using dynamic programming and complicated path "adjustments" arguments (which were the source of the original erroneous claims of the problem's polynomial-time solvability).
- Our results apply also to give the first polynomial-time solution to the parts cutting problem (below).

2 APPLICATIONS AND RELATED WORK

2.1 The Parts Cutting Problem

Suppose we have a sheet of some material such as wood or sheet metal, and a collection $P_1, \ldots, P_k$ of disjoint polygons on the sheet that must be cut out with a laser or saw as efficiently as possible. We limit the options by requiring that the edges of each polygon P_i be cut in one continuous cycle, starting at some point $p_i \in \partial P_i$. The cutter starts at point s and the objective is to determine the points P_i that minimize the length of the polygonal chain $(s, P_1, p_2, \ldots, P_k)$, since the total length of the cutter path is the length of this chain plus the sum of the perimeters of the parts P_i (and this sum is a constant). We thus want a minimum length path that visits the boundaries of all the polygons. We suppose, furthermore, that we are given the order in which the polygons must be visited. (Certain applications specify the order in which parts must be cut; without a given order, the problem is obviously NP-hard.) This problem is known as the parts cutting problem [16]. The parts cutting problem is exactly the unconstrained TPP with disjoint P_i's and $F_i = \mathbb{R}^2, \forall i$. Our algorithm solves the parts cutting problem in time $O(kn \log(n/k))$; see Section 3.

2.2 The Safari and Zookeeper Problems

For the safari and zookeeper problems, we are given a simple polygon P, and disjoint convex polygons ("cages") $P_1, \ldots, P_k$ inside P, each sharing exactly one edge with P. In the zookeeper problem we seek a shortest tour inside P that visits each of the P_i's but never enters any of them. In the safari problem we seek a shortest tour inside P that visits each of the P_i's, and is allowed to enter them. In this paper we consider only the "fixed-source" versions of these problems in which a starting point, s, for the tour is given. Although these problems seem different from the TPP in that no ordering of the P_i's has been specified, it is easy to argue that a shortest tour must visit the P_i's in the same order as they meet the boundary of P ([8, 21]). The zookeeper problem, introduced by Chin and Ntafos [8], has an $O(n \log n)$ time algorithm [3] utilizing the full shortest path map. The safari problem was introduced by Ntafos[21], who claimed an $O(n^3)$ time algorithm, which was then improved to $O(n^2)$ [24]. However, Tan and Hirata [28] found an error in the earlier analysis and presented the current best algorithm with running time $O(n^3)$. Our algorithm solves a much more general problem, and improves the running time for the safari problem to $O(kn \log n)$.

2.3 The Watchman Route Problem

In the watchman route problem (WRP) we are given a simple polygon P, and the goal is to find a shortest tour inside P so that every point in P is seen from some point of the tour. We consider the "fixed-source" case of the WRP, in which a point $s \in \partial P$ is specified as the starting point of the tour; the "floating" WRP, with no point s specified, requires time $O(n)$ times that of the fixed WRP, as shown recently by Tan [26]. The WRP was introduced by Chin and Ntafos [9, 7], who claimed in [7] an $O(n^4)$ time algorithm for the fixed WRP in a simple polygon. However, years later, there was a flaw discovered in their algorithm, and several attempts were made to correct it (some of which were themselves flawed). The best current algorithm for the fixed WRP, based on a relatively complex dynamic programming algorithm, is due to Tan, Hirata, and Inagaki [25] and runs in time

$O(n^4)$. Our results imply a new, simpler algorithm for the fixed WRP that runs in time $O(n^3 \log n)$. In order to see that the WRP is a special case of the TPP, we recall the notion (e.g., from [7]) of essential cuts and the essential pockets they define: A cut, c_i, with respect to s is a chord defined by extending the edge e incident on a reflex vertex, v_i, where e is chosen to be that edge incident on v whose extension creates a convex vertex at v in the piece containing s. The portion of P on the side of c_i not containing s is the pocket, P_i, induced by the cut c_i. A pocket is essential if no other pocket is fully contained within it. A tour sees all of P if and only if it visits all essential pockets. Let $P_1, \ldots, P_k$ denote the sequence of essential pockets, clockwise around ∂P starting from s. It is known that the shortest watchman tour visits this sequence of essential pockets in order. We then get an instance of the TPP. by defining each fence F_i to be the bounding polygon P.

3 THE UNCONSTRAINED TPP FOR DISJOINT CONVEX POLYGONS

3.1 Local Optimality Conditions

Clearly we only need to consider k-paths that are polygonal chains. Such a path is locally shortest if bends occur only at points on the boundaries of the P_i's, and for each such bend point $b \in \partial P_i$, moving b slightly in either direction on the boundary of P_i while keeping other bends fixed makes the path longer. This implies that for a bend point b on the interior of an edge e of P_i, the angle between the incoming segment $(\overline{ab})$ and e must be equal to the angle between the outgoing segment $(\overline{bc})$ and e — i.e., the path must reflect on e, so that $b \in \overline{ac'}$, where c' is the reflection of c with respect to the line through e.

For a bend point at a vertex $v = e_1 \cap e_2$ the condition of being locally shortest is that the outgoing path segment from v must leave v in the cone, γ, bounded by the reflected rays, r_1' and r_2', corresponding to the two edges.

The following is a special case of Lemma 8.

Lemma 1. For any $p \in \mathbb{R}^2$ and any $i \in 0, \ldots, k$, there is a unique locally shortest i-path, $\pi_i(p)$, to p. Thus, local optimality is equivalent to global optimality.

Proof. Let T_i be the first contact set of P_i, i.e., the points where a shortest $(i-1)$-path first enters P_i after visiting $P_1, \ldots, P_{i-1}$. Since the P_j's are disjoint, $T_i \subset \partial P_i$. One can show that T_i is connected:

Lemma 2. Each first contact set T_i is a (connected) chain on the boundary of P_i.

3.2 The Last Step Shortest Path Map

Locally shortest i-paths may leave points of T_i only in certain directions, either continuing in a straight line, or properly reflecting. For p in T_i let $r_i^s(p)$ be the set of rays of locally shortest i-paths going straight through p, and let $r_i^b(p)$ be the set of rays of locally shortest i-paths properly reflecting at p. (The mnemonic is "s" for straight, "b" for bounce.) Lemma 1 implies that for p in an edge of T_i, $r_i^s(p)$ and $r_i^b(p)$ each contain a single ray, and for p a vertex of T_i, $r_i^s(p)$ contains a single ray and $r_i^b(p)$ consists of a cone.

Let $r_i(p) = r_i^s(p) \cup r_i^b(p)$ and $R_i = \cup_{p \in T_i} r_i(p)$. The set R_i is an infinite family of rays with the property that each point in the plane is reached by exactly one ray; we say that R_i is a starburst with source T_i. Associated with R_i is a subdivision, S_i, of the plane that groups together points reached by rays of R_i that leave from the same vertex of T_i, or leave from the same side of the same edge of T_i. We call the polygonal subdivision S_i the last step shortest path map (with respect to s) associated with P_i, since it encodes the information about the last segment in a shortest i-path from s to any point p. In more detail, S_i decomposes the plane into cells σ of two types: (1) cones with an apex at a vertex v of T_i, whose bounding rays are those of $r_i^b(v)$; and (2) unbounded three-sided regions associated with an edge e of T_i. Cells of type (2) can be further classified as being reflection cells or pass-through cells, depending on whether the rays reflect off e or enter the interior of P_i at e. For cells of type (1), v is the source of the cell; for cells of type (2), e is the source. The union of all pass-through cells defines the pass-through region. Refer to Figure 1 for an example showing S_1 and S_2.

Using $\{S_1, \ldots, S_i\}$ we can readily compute a shortest i-path to any query point $q \in \mathbb{R}^2$ as follows: We locate q in S_i. If the source of the cell σ containing q is a vertex, v, of T_i, then the last segment of $\pi_i(q)$ is $\overline{vq}$, and we recursively compute the shortest $(i-1)$-path to v (locating v in S_{i-1},etc). If the source of σ is an edge e of T_i, then there are two subcases: (a) cell σ is a pass-through cell, in which case $\pi_i(q) = \pi_{i-1}(q)$ and we recursively compute the shortest $(i-1)$-path to q; or (b) cell σ is a reflection cell, in which case we let q' be the reflection of q in the line through e, and recursively compute the shortest $(i-1)$-path to q'; replacing the portion of this path from e to q' by the segment from e to q yields the shortest i-path to q.

Lemma 3. Given $S_1, \ldots, S_i$, the path $\pi_i(q)$ can be determined in time $O(k \log(n/k))$ for any $q \in \mathbb{R}^2$.

Proof. The complexity of each S_i is clearly $O(|P_i|)$ (for $i = 1, \ldots, k$), and can be stored in a point location data stru-ture supporting $O(\log |P_i|)$-time queries for q in S_i. The time to find $\pi_i(q)$ is then $O(\sum_1^k log(|P_i|))$, which attains its maximum when each $|P_i| = n/k$.

We construct each of the subdivisions $S_1, S_2, \ldots, S_k$ iteratively, using the data structures $\{S_1, \ldots, S_{i-1}\}$ in the construction of S_i. We store the subdivisions in a point location data structure.

The algorithm is very simple: For each vertex v of P_i, we compute $\pi_{i-1}(v)$. If this path arrives at v from the inside of P_i then v is not a vertex of T_i. Otherwise it is, and the last segment of $\pi_{i-1}(v)$ determines the rays $r_i^b(v)$ and $r_i^s(v)$ that define the subdivision S_i. Thus, combining with Lemma 3 we have

Theorem 1. For the unconstrained TPP for disjoint convex polygons with input size n, a data structure of size $O(n)$ can be built in time $O(kn\log(n/k))$ that enables shortest i-path queries to any query point q to be answered in time $O(i\log(n/k))$.

4 THE GENERAL TPP ALGORITHM

We turn our attention now to the general touring polygons problem with intersecting polygons and fence constraints. As stated in the introduction, we assume that fence F_i contains P_i and P_{i+1}, and we require that the subpath from p_i to p_{i+1} must lie in F_i. We solve a more general case than that of convex P_i's. Define the facade of P_{i+1} in F_i to be $\partial P_{i+1} \cap cl(F_i - P_{i+1})$. We assume that the facade of P_{i+1} in F_i is a (single) convex chain; i.e., adding the segment to close the chain yields a convex polygon (possibly degenerating to a line segment). The facade represents the points on the boundary of P_{i+1} from which rays in F_i can bounce.

4.1 Combinatorial Facts

In the presence of fences, a locally shortest path may bend at a fence vertex, with the usual condition for shortest paths inside a polygon — namely that the angle interior to the polygon be reflex. Intersecting polygons also complicate things: a path is locally shortest at an intersection point p of ∂P_{i-1} and ∂P_i if it is shorter than paths that visit P_{i-1} and then P_i close to p.

We again use last step shortest path maps. As before, we let T_i be the first contact set of P_i, i.e., the locus of points where a locally shortest fenced $(i-1)$-path first enters P_i after visiting $P_1, \ldots, P_{i-1}$, and then travelling though F_{i-1} to P_i. Because polygons may intersect, first contact may occur when a locally shortest fenced $(i-1)$-path ends at a point that is already inside P_i; thus T_i need not be part of P_i's boundary. For example, if s is contained in P_1, then T_1 is just the point s. In general, T_i will consist of the portions of T_{i-1} that lie inside P_i together with parts of the boundary of P_i. As before, we define R_i to be the set of rays leaving points of T_i to begin locally shortest fenced i-paths.

In order to prove that a locally shortest path is globally shortest, we need to prove some results on T_i and R_i. We begin by describing T_i in terms of T_{i-1} and the fence F_{i-1}. T_i can be described as the set of points $p \in P_i$ s.t. there is a shortest path inside F_{i-1} starting along a ray of R_{i-1} from some point of T_{i-1}, and intersecting P_i for the first time at p. Let A_{i-1} be the last rays of such paths. We include in A_{i-1} the rays of R_{i-1} leaving points $p \in T_{i-1} \cap P_i$. Sources for the rays of A_{i-1} are vertices of T_{i-1}, points on edges of T_{i-1}, and vertices of F_{i-1}. In the absence of fences, A_{i-1} was a subset of R_{i-1}, but now F_{i-1} intervenes.

We will assume by induction that T_{i-1} is a tree and R_{i-1} is a starburst. We will prove that any two distinct rays of A_{i-1} do not intersect; that R_i is a starburst, that T_i is a tree.

We begin by generalizing the uniqueness of locally shortest $s-t$ paths in a polygon to the case where the source is a starburst.

Lemma 4. Suppose F is a simple polygon, and we have rays emanating in a starburst from a connected source T inside F. Then any point p interior to F is reached by a unique locally shortest path starting from a ray emanating from a point of T.

Proof. This is true if all points in F are reached directly by rays of the starburst, i.e., every point $p \in F$ is reached by a ray r of the starburst emanating from some $t \in T$, with the line segment $\overline{tp}$ contained in F. Otherwise there must be a ray r of the starburst that travels in F to a reflex vertex v of F and then cuts off a hidden pocket P in F. Locally shortest paths from the starburst to points of P go through v, and locally shortest paths inside P from v are unique (this is the single source case). We can apply induction on $F - P$ to conclude that locally shortest paths from the starburst are unique in $F - P$. Furthermore, since the extension of ray r is in both parts of this partition, and provides a locally shortest path, thus no locally shortest path from either part may cross it.

Lemma 5. If R_{i-1} is a starburst from a tree T_{i-1}, then no two distinct rays of A_{i-1} intersect.

Proof. This is more general than the lemma above since the intersection point of the rays may lie outside F_{i-1}. But the same proof idea applies. Consider a ray r of R_{i-1} from point $p \in T_{i-1}$ that travels in F_{i-1} to a reflex vertex v of F_{i-1} and then cuts off a hidden pocket P. If the portion of r up to v makes first contact with P_i, then no points inside the pocket P are first contact points, and we are done by inducting on the smaller fence $F_{i-1} - P$. (Note that P_i still has a convex facade in this smaller fence.) On the other hand, suppose the portion of r up to v does not make first contact with P_i. Consider the line segments l^+ and l^- where l^+ goes from v along r to the first contact with ∂F_{i-1}, and l^- goes from v backwards along r, past p, to the first

contact with ∂F_{i-1}. (In other words, l^+ forms the boundary of pocket P.) We claim that because P_i has a convex facade it cannot intersect both l^+ and l^-. Thus we can apply induction on the smaller fence formed by cutting off the portion of F_{i-1} beyond l^+ or l^-, whichever does not contain part of P_i.

Lemma 6. If R_{i-1} is a starburst from a tree T_{i-1}, then the rays of A_{i-1} form an interval—i.e., the points at infinity reached by rays of A_{i-1} form an interval.

Corollary 1. If R_{i-1} is a starburst from a tree T_{i-1} then T_i is connected.

We begin with a discussion of what it means for rays to "bounce" at a vertex of T_i.

Claim 1. Let v be a vertex of T_i, with edges e^1 and e^2 incident to v and consecutive in clockwise order around v. Let R^1 [R^2] be the set of rays leaving points on e^1 [e^2], and entering the region between e^1 and e^2 in clockwise order. Let r^1 and r^2 be the limiting rays of these sets as the source approaches v along the edge. Then the rays that "bounce" (i.e., form locally shortest paths) at v consist of all rays between r^1 and r^2.

Lemma 7. If R_{i-1} is a starburst from a tree T_{i-1}, then R_i is a starburst.

Proof. We already have that no two distinct rays of A_{i-1} intersect. All of these become rays of R_i, though their sources move from points on T_{i-1} and F_i to points on T_i. It remains to show that the various rays that are formed when rays of A_{i-1} "bounce" off points of ∂P_i remain non-intersecting, and that they fill in the "gaps" to reach the whole plane. We address the issue of intersections first, dealing with rays that bounce from interior points of edges of T_i, and then with rays that bounce from vertices of T_i.

Two rays that bounce off edges of P_i cannot intersect each other because P_i has a convex facade.

We next consider the case of one ray bouncing off an edge e of P_i intersecting one ray of A_{i-1}. Suppose that a ray r from a source point t_1 of T_{i-1} bounces at edge e of P_i to form ray b. Suppose, for ease of description, that b bounces to the left of r. Suppose by contradiction that b is intersected by a ray s coming from a point t_2 of T_{i-1} inside P_i. This intersection must occur after s exits P_i. Because s and r do not intersect, t_2 must lie on the left of r. Now t1 and t2 are connected by a path, say γ, in T_{i-1}. The path γ must cross into P_i. Let p be the first point of γ (traversed from t_1 to t_2) on ∂P_i. Because γ cannot cross r or s, point p must lie between the points where r and s cross ∂P_i . Let e' be the edge of T_{i-1} containing p. Consider the two rays u and v of R_{i-1} that emanate from point p in R_{i-1}. We will argue that one of these rays must intersect r or s, contradicting the fact that R_{i-1} was a starburst. We crucially use the fact that u and v are reflections of each other in the edge e'. Let r' and b' be rays emanating from p parallel to r and b respectively. The wedge in which u can live extends from r' counterclockwise to the part of e' inside P_i (which comes before e). The reflecting wedge in which v can live extends from e' to a ray that comes before b'. Any possible ray v in this wedge will intersect either the path γ or the ray s. This is a contradiction.

The only rays left to consider for possible intersections are those that "bounce" at vertices of T_i on ∂P_i. These cannot produce an intersection, since, by Claim 1, they simply fill in the gaps between the rays emanating from interior points of edges.

Finally, we must argue that the rays of R_i reach every point in the plane. This follows from Claim 1, together with the fact that T_i is connected (Corollary 1).

Corollary 2. If R_{i-1} is a starburst from a tree T_{i-1}, then T_i is a tree.

Proof. This follows from Corollary 1 plus the fact that the source of a starburst cannot contain a cycle (otherwise the ray of the starburst to a point inside the cycle would hit the cycle).

As an immediate consequence, we obtain

Lemma 8. For any $i \in \{0, \ldots, k\}$ and any $p \in F_i$ there is a unique locally shortest fenced i-path to p. Thus, local optimality is equivalent to global optimality.

4.2 The Algorithm

As before, we use a last step shortest path map. Actually, we use several such maps—with and without fences. We associate with the starburst R_i a subdivision of the plane, S_i^R , that groups together points reached by rays of R_i that leave from the same vertex of T_i, or leave from the same side of the same edge of T_i. A pass-through cell is one in which the rays leaving the source are inside P_i in a very small neighbourhood of the source. In particular, any cell of S_i^R whose source is a vertex or edge of some T_j, $j < i$, is a pass-through cell. For purposes of space efficiency, we group together all of the pass-through cells of S_i^R into one (possibly disconnected) "pass-through" region. Other cells of S_i^R have as their source either a vertex or an edge of T_i.

The structure S_i^R tells us how shortest paths leave P_i ignoring the fence F_i. To take the fence into consideration, we define a subdivision S_i^F of F_i that groups together points reached by shortest paths in F_i that start from rays of R_i, and whose last edges leave from the same vertex of F_i, or leave from the same vertex of T_i or leave from the same side of the same edge of T_i. Again, we group together all pass-through cells. (To add some intuition, for $P_0 = s$, S_0^R has a single region containing all rays leaving s, but S_0^F is a standard shortest path map inside the fence F_0.)

Finally, we define the subdivision of the plane, S_i^A, based on the rays A_i that arrive at P_{i+1}. Here we group together points of the plane reached by rays of A_i that emanate from the same vertex of T_i, or the same vertex of F_i, or from the same side of the same edge of T_i. Again, we group together all pass-through cells.

We describe below how to compute these structures, but first we show how to use them to answer queries.

4.2.1 Answering Shortest Path Queries

Given a point $q \in F_i$, we can find a shortest fenced i-path to q by calling Query(q, S_i^F). We answer such a query by locating q in S_i^F, and, based on the results, passing q, or an appropriate reflection of q, to appropriate level-$(i-1)$ data structures.

As we recurse, we will need to query S_{i-1}^A rather than S_{i-1}^F, since our query point will not be guaranteed to lie inside F_{i-1}, but will be guaranteed to be hit by a ray of A_{i-1}. We say that a path respects F_i if it lies in F_i except that the last edge may possibly exit F_i after entering P_{i+1}. Our algorithm to construct the shortest path maps will also need to query S_i^R. We thus describe a generic query in any of these structures.

For $X = F$ or A or R, Query(q, S_i^X) locates q in S_i^X, and then follows these cases:

- **Case 1**. q is in a cell whose root is a vertex $v \in F_i$ or a vertex $v \in T_i$. In this case, the last edge of the desired path is the line segment (v, q), and we can find the actual path by using stored information at v (which we will have queried previously).
- **Case 2**. q is in the "pass-through" region. In this case, a shortest fenced $(i-1)$-path to q automatically visits P_i, and the portion of the path from the entry point of P_i to q respects F_i. Thus, we desire a shortest fenced $(i-1)$-path to q. Note that q need not be inside F_{i-1}. However, it is reached by a ray of A_{i-1}, and so we obtain the correct answer by calling Query(q, S_{i-1}^A).
- **Case 3**. q is in a cell whose root is an edge e of T_i. Then the desired path to q bounces off edge e. In this case, let q'be the reflection of q with respect to the line through edge e. A shortest fenced $(i-1)$-path to q'automatically visits P_i, and the final portion of it respects F_i. Thus, we desire a shortest fenced $(i-1)$-path to q'. The last segment of such a path will cross edge e at a point p, say, and adding the line segment from p to q gives us the final path. Note that q'need not be inside F_{i-1}. However, it is reached by a ray of A_{i-1}, and so we obtain the correct answer by calling Query(q', S_{i-1}^A).

Provided we have the required planar subdivisions on hand, and their sizes are polynomial in n, the above algorithm proves:

Lemma 9. For any point q in F_i we can find the last ray of a shortest fenced i-path to q in time $O(i \log n)$. To compute the actual path we need to take into account the length of the path:

Lemma 10. For any point q in F_i we can find the shortest fenced i-path to q in time $O(i \log n + m) = O(i \log n + f)$, where m is the combinatorial length of the path, and $f = \sum_{j \le i} f_i$ is the total size of all the relevant fences.

4.2.2 Constructing the Last Step Shortest Path Maps

Assuming we have S_{i-1}^F and S_j^F, $S^A)j$ for $j < i$, we show how to construct S_{i-1}^A and T_i and S_i^R, and then, from those, how to construct S_i^F.

Constructing S_{i-1}^A. T_i, S_i^R. We compute all potential vertices of T_i that are not vertices of previous T_j's as follows: throw in all vertices of the facade of P_i plus all intersection points of the facade of P_i and the facade of P_j for $j < i$. For each such potential vertex v, we find the last edge of a locally shortest fenced (i- 1)-path to v by calling Query(v, S_{i-1}^F). On the basis of how the last edge of the path arrives at v, we determine whether v is indeed a vertex of T_i. If it is, the arriving ray becomes part of S_{i-1}^A, and we compute the rays $r_i^s(v)$ and $r_i^b(v)$ to become part of S_i^R. We preprocess S_{i-1}^A and S_i^R for point location queries.

Constructing S_i^F.

We preprocess F_i for ray shooting queries; this takes $O(|F_i|)$ time, and allows queries in $O(\log n)$ time [17]. We call Query(v, S_i^R), for each reflex vertex v of F_i, to find the last edge l of a shortest fenced i-path to v ignoring the fence F_i. Using a ray shooting query in F_i, we determine if l is contained in F_i. If not, then we ignore v. Otherwise we have found the shortest fenced i-path to v. Let r be the continuation of l beyond v until its first intersection point with ∂F_i. As in the proof of Lemma 4, we can identify the hidden pocket beyond r. We can compute a standard shortest path map for source point v inside this pocket in linear time [17]. Putting together this information for all reflex vertices v of F_i, and combining with the relevant portions of R_i, yields the subdivision S_i^F of F_i, which we then process for point location queries.

Running time for the algorithm. Let f_i be the size of F_i, and p_i be the size of the facade of P_i in F_i. Let $f = \sum_i f_i$, and $p = \sum_i p_i$. In our general case f and p are $O(n)$. Let t_i be the number of vertices of T_i that are not vertices of a previous T_j, and let $\sum_i t_i$.

Lemma 11. t is $O(pk)$.

Proof. Each vertex of T_i that is not a vertex of some previous T_j, $j < i$, is an intersection point of an edge of the convex facade of P_i with an edge of the convex facade of P_j for some $j < i$. Since the facades are

convex, the number of intersection points is at most P_i, $\sum_{i,j} O(p_j + p_i)$, which attains its maximum value when $p_i = \lceil p/k \rceil$ for each i. Hence, the total number of these vertices is $O(pk)$.

The complexity of S_i^R is $O(ti)$, the complexities of S_i^F and S_{i+1}^A are $O(t_i + t_{i+1} + f_i)$. In the general case these are both $O(nk)$ and, by Lemma 9 a query to find the last edge of a shortest fenced i-path ($i \leq k$) to a point q takes $O(k \log n)$ time.

We perform such a query for each vertex of each F_i, and each vertex of each T_i that is not already a vertex of a previous T_j. Thus we do $p + f + t$ queries, and the total time required is $O((p + f + t)k \log n)$. In our general case, this is $O(nk^2 \log n)$.

Theorem 2. The TPP for arbitrary convex polygons P_i and fences F_i can be solved in time $O(nk^2 \log n)$, using $O(nk)$ space.

The Watchman Route Problem. For the watchman route problem in polygon P, the polygons P_i are defined by the $k = O(n)$ essential cuts of P. Each P_i may have $O(n)$ vertices, but p_i, the size of the facade of P_i, is just 2. Thus p is $O(n)$. Each fence, F_i, is all of P, so f is $O(n^2)$. Applying Lemma 11 we have that t is $O(nk)$. From our analysis above, our algorithm takes time $O((p + f + t)k \log n)$. Plugging in, we get $O(n^3 \log n)$.

Corollary 3. The (fixed-source) watchman route in a simple polygon can be solved in time $O(n^3 \log n)$. The Safari Problem: For the safari problem, p is $O(n)$.

Because the P_i's are disjoint, t_i is just p_i, and t is $O(n)$ as well. Finally, fences can be defined such that f is $O(n)$ ([3]). Plugging into the running time of $O((p + f + t)k \log n)$ yields $O(nk \log n)$.

Corollary 4. The safari problem can be solved in time $O(nk \log n)$.