Discrete Comput Geom 22:377–402 (1999)

Discrete & Computational
Geometry

Finding the Shortest Watchman Route in a Simple Polygon

S. Carlsson,[1] H. Jonsson,[1] and B. J. Nilsson[2]

[1]Department of Computer Science, Luleå University of Technology,
971 87 Luleå, Sweden
{Svante.Carlsson,Hakan.Jonsson}@sm.luth.se

[2]Department of Computer Science, Lund University,
Box 118, 221 00 Lund, Sweden
Bengt.Nilsson.lth.se

Abstract. We present the first polynomial time algorithm that finds the shortest route in a simple polygon such that all points of the polygon are visible from the route. This route is called the shortest watchman route, and we do not assume any restrictions on the route or on the simple polygon. Our algorithm runs in worst case $O(n^6)$ time, but it is adaptive, making it run faster on polygons with a simple structure.

1. Introduction

It has been known for a long time [1], [11] that the so-called art gallery problem is NP-hard. This is the problem of finding the smallest set of guards within a simple polygon such that each point of the polygon is visible from at least one guard. At the same time there are many examples of optimization problems and in particular shortest route problems (for instance, the Traveling Salesperson Problem) that are NP-hard. The combined problem, to find the shortest closed curve (watchman route) inside a simple polygon such that each point of the polygon is visible to at least one point on the curve, seems to be at least as hard as the two above. Therefore, it was quite surprising when Chin and Ntafos claimed that it was possible to find the shortest watchman route that is forced to pass a given point on the boundary of the polygon in polynomial time [6]. Using variants of the original algorithm the running times were subsequently improved by Tan et al. [17], [18]. An error that in some special instances lead to exponential running times in all previously presented algorithms was discovered by Hammar and Nilsson [8] and a possible solution was suggested. However, their proposed solution only reduces the types of instances that have exponential behavior. Recently, Tan et al. [19] presented a correct algorithm based on the original techniques in conjunction with dynamic programming, thus removing the exponential behavior in all instances. This algorithm runs in worst case $O(n^4)$ time.

In some practical applications, for instance, if we would like to patrol a building with a robot that has to enter the building through a door, this restriction is of minor importance. In other cases, as, for instance, in illumination problems, the restriction of forcing the route through a specific point can be devastating since the route can be arbitrarily longer than the shortest watchman route without any restrictions. Despite the importance of the problem and a number of attempts to solve it the problem has stayed open until now.

In this paper we make some important observations to solve the general problem of finding the shortest watchman route in a simple polygon. We reduce the problem to a polynomial number of shortest watchman route problems with a fixed boundary point, and solve these using an existing algorithm. This, together with a sweep technique that we call "sliding," enables us to construct the shortest watchman route in worst case $O(n^6)$ time. In many polygons though, the algorithm will run faster. The presented algorithm is a modified and corrected version of a result presented at ISAAC '93 [2].

2. Definitions and Preliminary Results

Let **P** be a simple polygon having n edges. We assume a representation of **P** as a list of the coordinates of the vertices as they are encountered during a counterclockwise scan of the boundary of **P**. This representation implies an orientation on the edges of **P** and, hence, we can say that the interior of the polygon is (locally) to the left of an edge.

A point p in **P** is said to *see* a point q in **P** if the line segment between the two points is contained in **P**. We also say that the two points are *visible* to/from each other. A *guard set* for **P** is a set of points in **P** such that for each point p in **P** there is a point q in the guard set that sees p.

A *watchman route* is a closed curve W in **P** such that W is a guard set for **P**. If we specify a point d on the boundary of **P** and force the watchman route to pass through this point, we talk about a *fixed watchman route* with the point d being the *door* of the route. If no such point is specified, the route is called a *floating watchman route*. In the following, when we talk about a watchman route we mean a floating watchman route unless otherwise specified.

Since our aim is to compute the shortest watchman route, we need to be able to measure length. Our measure of distance is the standard Euclidean distance function and the distance between two points p and q is denoted $\|p, q\|$. The length of a segment is the distance between the two end points of the segment. A chain is a curve consisting of consecutive segments that are not collinear. The length of a chain C, denoted $length(C)$, is the sum of the lengths of the segments of C.

The shortest watchman route, whether floating or fixed, consists of line segments such that no two consecutive segments are collinear, i.e., it is a closed chain. Similarly as for polygons, we represent a watchman route by a list of the vertices as they are encountered during a counterclockwise scan of the route.

Consider the example polygon in Fig. 1. The shortest watchman route is a tour with (some) turning points on (some of) the extensions of the polygon edges that are adjacent to reflex vertices. The reason for this is that the tour needs to see everything behind each polygon edge. Hence, the extensions of polygon edges are important to know and this leads us to the following definitions.

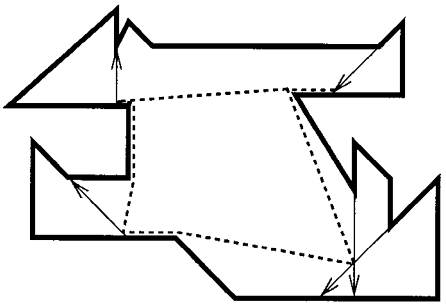
Fig. 1. An example polygon with its shortest watchman route.

We define a *cut* to be a directed line segment in **P** with the following properties. The end points of a cut must coincide with the boundary of **P** and part of the cut's interior must lie in the interior of the polygon. Hence a polygon edge is not a cut. A cut separates **P** into two subpolygons. If a cut is represented by the segment $[p, q]$ we say that the cut is directed from p to q and we call p the *start point* of the cut. We say that a point lies to the right/left of a cut if the point lies locally to the right/left in the subpolygon separated by the cut.

Consider a reflex vertex of a polygon. The two edges connecting at the vertex can each be extended inside **P** until the extensions reach a boundary point. These extended segments are given the same direction as the edge they are collinear to. We call the cuts thus constructed *extension cuts*. Now, it is easy to see that all guard sets must have a point to the left of (or on) each extension cut, since otherwise the edge collinear to the cut will not be seen by the guard set; see Fig. 1.

To illustrate the next concept, we assume that one point of a shortest watchman route is known to us. Let this point be denoted p. It turns out that not all extension cuts are interesting to maintain, but only the ones that have the point p, and thus, the main part of the watchman route, to the right, since those are the ones where visibility is blocked by the associated polygon edges; see Fig. 1. We therefore make the following further separation between types of extension cuts.

Given a point p of a polygon, we say that an extension cut c is *forward with respect to* p if p lies to the left of the cut c. Otherwise c is *backward with respect to* p; see Fig. 2(a).

An extension cut c *dominates* another extension cut c' if all points in **P** to the left of c are also to the left of c'; see Fig. 2(b).

We say that an extension cut is an *essential cut* if it is not dominated by any other extension cut; see Fig. 2(b). We state the following lemma without proof.

Lemma 2.1. *A closed curve is a watchman route if and only if the curve has at least one point to the left of* (*or on*) *each essential cut.*

We can view the essential cuts as having a cyclic ordering specified by the start points of the cuts as they are encountered during a counterclockwise scan of the polygon boundary. In this way each cut has a predecessor and a successor. The set of essential cuts of the polygon **P** will henceforth be denoted $\mathcal{C}$.

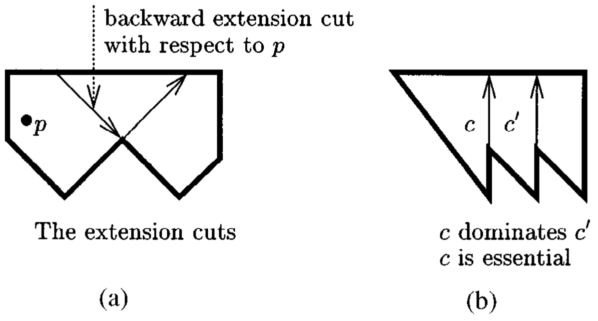

Fig. 2. Illustrating the definitions of extension cut, domination, and essential cut.

Consider an essential cut. The cut is intersected by at most $k-1$ other essential cuts, k being the total number of essential cuts in $\mathcal{C}$, and hence, each essential cut is subdivided into at most k segments spanning between the cut intersection points. We call these segments the *fragments* of a cut. As before we can define the dominance relation between a fragment f and a cut c. We say that f dominates c if f lies to the left of (or on) c. Hence, a fragment of a cut dominates its cut.

We can now formulate the shortest watchman route problem as: "Compute the shortest closed curve that intersects all essential cuts." The rest of this presentation is devoted to showing how to obtain such a curve.

3. Overview of the Fixed Case

All suggested algorithms for the fixed shortest watchman route problem start by constructing an initial watchman route through the door d, i.e., a closed curve that intersects all backward essential cuts with respect to d. This follows from Lemma 2.1 since d is, by definition, to the left of the forward essential cuts with respect to d, and, hence, these cuts do not have to be considered.

The algorithms then progress by applying a sequence of *adjustments* to the initial route. In order to explain these adjustments, it is important to know what kind of intersections a watchman route can make with the backward essential cuts with respect to d. A watchman route makes a *reflection contact* with a cut c if the intersection of the route and c is one point and all other points of the route lie to the right of c; see Fig. 3(a). A reflection contact is *perfect* if the incoming angle equals the outgoing angle of the reflection. A watchman route makes a *crossing contact* with c, if each intersection is one point and the contact is not a reflection contact; see Fig. 3(b). Finally, the route makes a *tangential contact* with c if the intersection is a line segment and all other points of the route lie to the right of c; see Fig. 3(c).

Consider the essential cuts where a watchman route makes reflection contact. We call these cuts the *active cuts* and the fragments that contain the intersection points are the *active fragments*. Two conditions are imposed on the set of active fragments in order to ensure the correctness of the algorithms and for ease of computation.

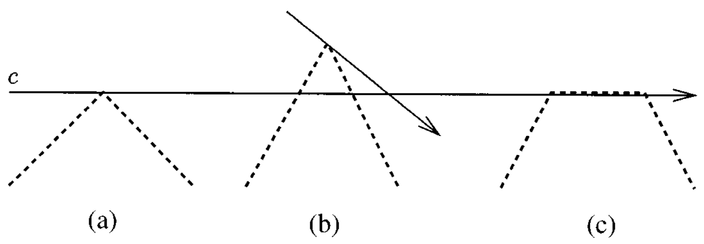

Fig. 3. The three different types of possible contacts made by a shortest watchman route.

Completeness. The set of active fragments dominate all essential cuts of **P**.

Independence. An essential cut is dominated by exactly one active fragment.

Now the following lemma can be shown.

Lemma 3.1. *The completeness condition must hold for the active fragments of a shortest watchman route.*

Proof. If there is some essential cut not dominated by an active fragment, the edge of **P** corresponding to the essential cut is not seen by the watchman route. □

Furthermore, the following two lemmas provide a way to construct the shortest fixed watchman route given an initial watchman route.

Lemma 3.2 [6]. *There is a shortest watchman route that visits the active cuts in the order that they appear as the boundary of the polygon is traversed.*

Lemma 3.3 [6]. *A shortest watchman route either makes perfect reflections on the active cuts or it reflects at fragment end points of the active cuts.*

Both lemmas also hold for the shortest floating watchman route. Thus, a shortest route that visits all the essential cuts and obeys the properties of the two lemmas will be a shortest route overall.

The problem thus becomes that of adjusting the initial route so that all the reflection contacts are perfect, or no more reflection contacts can be made perfect.

Given a set of active cuts, how is the shortest fixed watchman route with reflection contacts at these cuts computed? The approach taken is by *unfolding* the polygon **P**, which is a process that produces a polygonal shape that we call an *hourglass*, such that the shortest path from d to its image in the hourglass corresponds to the shortest fixed watchman route through d that reflects on the active cuts. The process is carefully explained by Chin and Ntafos [6]. The hourglass is constructed from **P** by cutting off the parts of **P** that lie to the left of the active cuts. To do this, we assume that the active cuts are given in the order as their start points are traversed in counterclockwise order

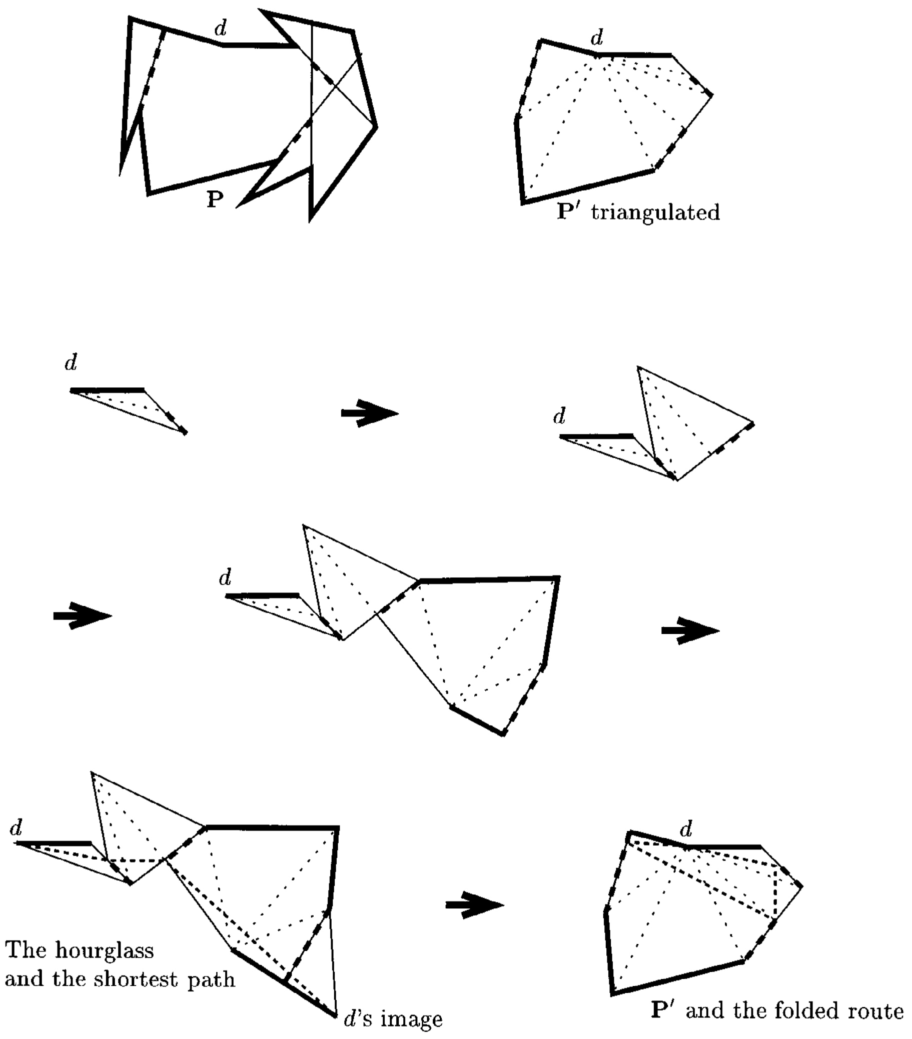

Fig. 4. Illustrating the unfolding procedure.

along the boundary of **P**. Now we take each active cut in the ordering and remove the part of **P** to the left of the cut. This involves computing the intersection point with the previous active cut in the ordering and, if it exists, introduce a new vertex at this intersection point. The process takes constant time for each active cut and, hence, linear time in total. In this way we get a new polygon **P**′ with the active cuts on the boundary. The polygon **P**′ is triangulated using Chazelle's algorithm [4] and unfolded using the active cuts as mirrors (see Fig. 4) in the following way: from the point *d* follow the boundary in clockwise fashion until the first active cut is reached. Construct a polygon from the triangles of the triangulation that are adjacent to the traversed part of the boundary. Now follow the polygon boundary from the first active cut to the second active cut and construct a polygon consisting of the triangles adjacent to this section of the boundary. Attach this polygon to the previously constructed polygon using the active cut as a mirror. Continue the process as in Fig. 4 until the starting point d is reached. The constructed polygon is the hourglass of $\mathbf{P}$. The shortest path in the hourglass from d to its image point is computed [7], [13]. Finally, the route is folded back to give the shortest fixed watchman route in $\mathbf{P}$. The time complexity is linear, since $\mathbf{P}'$ can be triangulated in linear time and the shortest path can be computed in linear time in a triangulated polygon [7], [13].

Recently, it has been shown how to compute the shortest fixed watchman route in $O(n|\mathcal{C}|F)$ time and $O(n|\mathcal{C}|)$ storage, where $|\mathcal{C}|$ is the number of essential cuts and F is the number of fragments, using an incremental algorithm in conjunction with dynamic programming.

Since we will use this result, we claim it as a theorem.

Theorem 1 [19]. *There is an algorithm that, given a boundary point d in a simple polygon of n edges, the backward essential cuts with respect to d, and their subdivision into fragments, computes the shortest fixed watchman route through d in $O(n|\mathcal{C}|F)$ time and $O(n|\mathcal{C}|)$ storage, where $|\mathcal{C}|$ is the number of essential cuts and F is the number of fragments.*

In Section 4.2 we show that the time to compute the set of essential cuts is $O(n \log n)$, and that the fragments can be computed in time $O(n \log n + F)$. This, together with Theorem 1, implies that the shortest fixed watchman route through a given boundary point can be computed in $O(n^4)$ time, since $F = O(n^2)$.

We denote the time and storage complexities to compute a shortest fixed watchman route by $T(n, |\mathcal{C}|, F)$ and $S(n, |\mathcal{C}|, F)$ repectively. Hence, by Theorem 1, $T(n, |\mathcal{C}|, F) = O(n|\mathcal{C}|F)$ and $S(n, |\mathcal{C}|, F) = O(n|\mathcal{C}|)$.

The adjusting technique that we have described in this section is used extensively in the following, where we show how to eliminate the door restriction.

4. The Algorithm

4.1. *Presentation*

We present a polynomial time algorithm to compute a shortest floating watchman route in a simple polygon. The idea of the algorithm is to precompute the shortest fixed watchman routes making reflections at the fragment end points. Thus, we are left with only a restricted case to handle, the case when the shortest watchman routes makes only perfect reflections in the interior of fragments. To solve the problem in this case, we apply a process we call *sliding* that makes a discrete simulation of the continuous motion performed by a reflection point of a watchman route as the reflection point moves between the two end points of an active fragment.

To simplify our presentation, we assume that the input polygon is not star-shaped. In a star-shaped polygon, the problem of computing the shortest watchman route has a linear time solution—compute the kernel of the polygon [12], and select any point of the kernel as the resulting route.

The pseudocode of the algorithm that computes a shortest floating watchman route is presented above. The rest of this presentation is devoted to proving the correctness of the algorithm, and analyzing its complexity. In Step 1 of the pseudocode, we compute the set $\mathcal{C}$ of essential cuts. This part of the algorithm is described in Section 4.2. The description of how to perform Step 2.1 is presented in Section 4.3, and we show how to do the sliding process of Step 3.1 in Section 4.4. In Section 4.5 we prove the correctness and analyze the running time of the algorithm.

```
Algorithm   Shortest-Floating-Watchman-Route

Input:      A simple polygon P of $n$ edges
Output:     A shortest floating watchman route $W$

1           Compute the set $\mathcal{C}$ of essential cuts and the subdivision into
            fragments
2           for each fragment end point $d$ do
2.1            Compute the shortest watchman route $W_d$ forced to
               reflect on $d$
            endfor
3           for each fragment $f$ and each end point $d$ of $f$ do
3.1            Apply a sliding process on $f$ from $d$ to the other end point,
               and compute the shortest watchman routes forced to reflect
               on an interior point of $f$
            endfor
4           Return the shortest of the computed watchman routes
End         Shortest-Floating-Watchman-Route
```

4.2. *Computing the Essential Cuts*

To compute the set $\mathcal{C}$ of essential cuts, we begin by computing all the extension cuts of the polygon. To do this, we use a ray shooting data structure as presented by Guibas et al. [7] or Hershberger and Suri [10]. The ray shooting operations can be performed in $O(\log n)$ time each, with the initial preprocessing step taking linear time. At every reflex vertex of the polygon, we perform two ray shooting operations, one in the direction of each of the two adjacent edges toward the interior of the polygon. In this way we specify the two extension cuts associated to every reflex vertex. The total time used is $O(n \log n)$.

Next we determine one essential cut. Let $\mathcal{E}$ denote the set of extension cuts. Between two cuts it is easy to check in constant time whether one cut dominates another, if we maintain information on where the cut end points lie on the boundary of **P**. Since the dominance property is transitive, we can, in linear time, find one essential cut by performing pairwise comparisons, always keeping the cut that is not dominated. Let c_1 be the essential cut we get through this process.

Now we sort the set $\mathcal{E}$ so that the extension cuts appear in the same order as their start points occur in a counterclockwise traversal of the boundary, beginning at the start point of c_1.

To compute the essential cuts, we traverse the ordered set $\mathcal{E}$ and perform the following steps:

```
Let $current := c_1$ and set $\mathcal{C} := \{c_1\}$
**for** $i := 2$ **to** $|\mathcal{E}|$ **do**
  **if** $current$ does not dominate $c_i$ **then**
    $\mathcal{C} := \mathcal{C} \cup \{c_i\}$
    $current := c_i$
  **endif**
**endfor**
```

Lemma 4.1. *The set $\mathcal{C}$ contains the essential cuts once the loop has terminated.*

Proof. To see that all the essential cuts are in $\mathcal{C}$, when the loop terminates, note that an extension cut c is inserted in $\mathcal{C}$ unless we can determine some essential cut that dominates c. Hence, it only remains to prove that the set $\mathcal{C}$ contains only the essential cuts.

First, note that if a cut $c_j \in \mathcal{E}$ dominates a cut $c_i \in \mathcal{E}$, according to the index ordering of $\mathcal{E}$ determined previously, then $j < i$. If this is not the case, there are points to the left of c_j that are not to the left of c_i, e.g., the boundary points between the start points of c_i and c_j.

Now, assume that there is a nonessential cut c_i in $\mathcal{C}$, where i is the index of the sorted order in $\mathcal{E}$. The cut c_i is dominated by some essential cut $c_j \in \mathcal{C} \subseteq \mathcal{E}$, with $j < i$. Consider the subsequence $c_j, \ldots, c_i$ of cuts in $\mathcal{E}$. When the dominance test is applied to the cut c_i, the variable $current = c_k$, with $j \le k \le i - 1$. We claim that, in this case, the cut c_i is also dominated by c_k. To see this, observe that, since c_i is dominated by c_j, the cut c_k must intersect c_j, otherwise c_k is dominated by c_j. However, this means that c_k dominates c_i, and, in turn, it means that when the loop considers c_i, the cut c_i will not be included in $\mathcal{C}$; see Fig. 5. □

We conclude that the total time consumption for the computation of the essential cuts is $O(n \log n)$. In addition, our shortest watchman route algorithm also requires the subdivision of the essential cuts into fragments, i.e., the line segments between consecutive intersection points of pairs of essential cuts. These can be computed, and ordered along each essential cut, in time $O(n \log n + F)$, where F denotes the number of fragments, using an intricate plane sweep algorithm developed by Chazelle and Edelsbrunner [5].

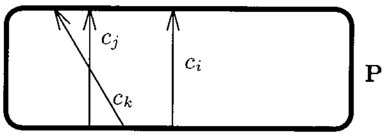

Fig. 5. Dominated cuts are removed from the set $\mathcal{C}$.

4.3. *Shortest Watchman Routes for the Fragment End Points*

Consider a shortest watchman route in the polygon $\mathbf{P}$. The route will make at least one reflection contact with some essential cut c, since we assume that $\mathbf{P}$ is not starshaped. Furthermore, the reflection contact can be one of two types. The first case is that the route reflects at a fragment end point of c, i.e., the intersection point of c with some other essential cut, or an end point of the cut c. The second case arises when the route reflects in the interior of a fragment, in which case the reflection will be perfect by Lemma 3.3.

In this section we determine how to compute the shortest watchman routes forced to have reflection contacts at the fragment end points. The case when the shortest watchman route only has perfect reflections in the interior of active fragments will be taken care of in the next section.

Let $\mathcal{C} = \{c_1, \ldots, c_k\}$ be the essential cuts ordered on their start point cyclically around the boundary of $\mathbf{P}$. We assume that d, the fragment end point through which we are computing the shortest watchman route, is the intersection between c_i and c_j, with $1 \le i < j \le k$.

Let $\mathcal{C}_d$ be the backward essential cuts with respect to d, i.e., the cuts of $\mathcal{C}$ having d to the right according to their associated direction. The set $\mathcal{C}_d$ can be established in $O(|\mathcal{C}|)$ time by testing each cut in $\mathcal{C}$ against the point d. Let $\mathbf{lP}_c$ denote the part of $\mathbf{P}$ lying to the left, or on, the cut c.

Since we assume that a reflection is made at d, at least one of c_i and c_j must be active. Consider the case when c_i is active. In this case the shortest watchman route through d cannot make reflection contact with the parts of other cuts in $\mathcal{C}_d$ that lie to the left of c_i, since this would imply that the shortest watchman route crosses c_i. Hence, we only need to concern ourselves with the part of $\mathbf{P}$ lying to the right of c_i. Similar arguments can be applied if c_j is active or both c_i and c_j are active.

We construct three new polygons $\mathbf{P}_d^i = \mathbf{P} \backslash \mathbf{lP}_{c_i}$, $\mathbf{P}_d^j = \mathbf{P} \backslash \mathbf{lP}_{c_j}$, and $\mathbf{P}_d^{i,j} = \mathbf{P} \backslash (\mathbf{lP}_{c_i} \cup \mathbf{lP}_{c_j})$, and three sets of backward essential cuts in $\mathbf{P}_d^i$, $\mathbf{P}_d^j$, and $\mathbf{P}_d^{i,j}$, denoted $\mathcal{C}_d^i$, $\mathcal{C}_d^j$, and $\mathcal{C}_d^{i,j}$, respectively. We have

$$\begin{aligned}
\mathcal{C}_d^i &= \{c \cap \mathbf{P}_d^i \mid c \in \mathcal{C}_d \backslash \{c_i\}\}, \\
\mathcal{C}_d^j &= \{c \cap \mathbf{P}_d^j \mid c \in \mathcal{C}_d \backslash \{c_i\}\}, \qquad \text{and} \\
\mathcal{C}_d^{i,j} &= \{c \cap \mathbf{P}_d^{i,j} \mid c \in \mathcal{C}_d \backslash \{c_i, c_j\}\};
\end{aligned}$$

see Fig. 6.

Constructing the polygons $\mathbf{P}_d^i$, $\mathbf{P}_d^j$, and $\mathbf{P}_d^{i,j}$ takes $O(n)$ time and constructing the sets $\mathcal{C}_d^i$, $\mathcal{C}_d^j$, and $\mathcal{C}_d^{i,j}$ takes $O(|\mathcal{C}|)$ time using straightforward techniques. The point d lies on the boundary of each of the three polygons, and the sets $\mathcal{C}_d^i$, $\mathcal{C}_d^j$, and $\mathcal{C}_d^{i,j}$ can be viewed as the sets of backward essential cuts with respect to d in the polygons. Thus, we can use Theorem 1 of Section 3 and compute the shortest fixed watchman route through d, inside each of $\mathbf{P}_d^i$, $\mathbf{P}_d^j$, and $\mathbf{P}_d^{i,j}$. The only problem here is that some c in $\mathcal{C}_d$ may be such that $c \cap \mathbf{P}_d^{i,j} = \emptyset$, in which case the shortest fixed watchman route in $\mathbf{P}_d^{i,j}$ does not correspond to a watchman route in $\mathbf{P}$, since c is not visited. However, this can be detected when the set $\mathcal{C}_d^{i,j}$ is constructed and, hence, we do not have to compute the route in $\mathbf{P}_d^{i,j}$.

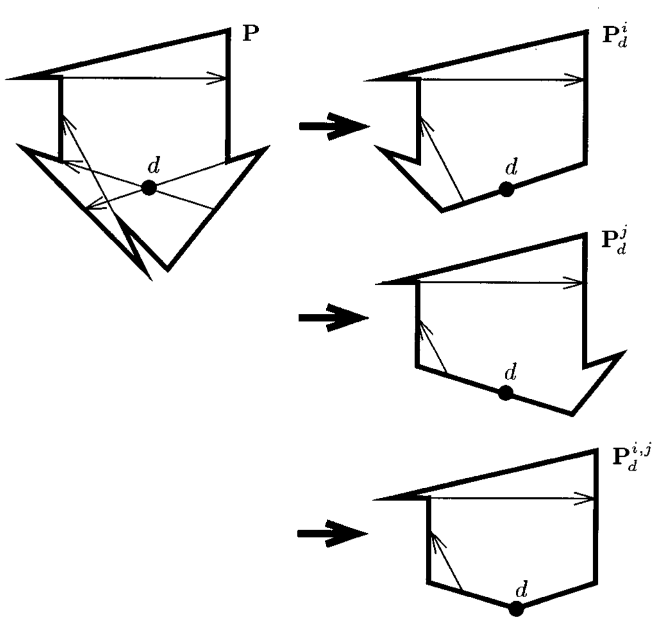

Fig. 6. Illustrating the polygons $\mathbf{P}_d^i$, $\mathbf{P}_d^j$, and $\mathbf{P}_d^{i,j}$, and the sets $\mathcal{C}_d^i$, $\mathcal{C}_d^j$, and $\mathcal{C}_d^{i,j}$.

Selecting the shortest of the valid routes gives us the shortest watchman route reflecting at d in $O(T(n, |\mathcal{C}|, F))$ time. Repeating for each fragment end point, we have the following lemma.

Lemma 4.2. *The set of shortest fixed watchman routes, each forced to reflect at an end point of a fragment, can be computed in $O(F \cdot T(n, |\mathcal{C}|, F))$ time, where F is the number of fragments of the essential cuts.*

4.4. *The Sliding Process*

The sliding process we apply in Step 3.1 of the pseudocode is the main step of our algorithm. The technique we use is similar to the one used by Melissaratos and Souvaine [14]. However, since our problem is more complicated we present the technique in detail. This section is divided into three parts. In the first part we introduce some notation and show some initial results on the key points at which structural changes occur in the sliding process. In the second part we discuss the necessary adjustments that have to be made as the sliding reaches a key point and we describe how to compute the key points in the third part.

The Event Points

We begin by reviewing some of the notation introduced in Section 3. Let f be the fragment of some essential cut on which we perform the sliding process, i.e., a fragment adjacent to d. We denote by W_p, the shortest watchman route passing through the point p of f. The route W_p corresponds to a shortest path S_p in some hourglass that we denote by $\mathbf{H}_d$. An hourglass is a two-manifold with the same properties as a triangulated polygon, and is obtained by the unfolding procedure described in Section 3. Denote the unfolded image of f in $\mathbf{H}_d$ by $\bar{f}$ and correspondingly the image of a point p on f by $\bar{p}$ on $\bar{f}$. Hence, S_p is the shortest path from p to $\bar{p}$ in $\mathbf{H}_d$. An hourglass is completely specified by the set of active cuts, and, hence, there is a direct correspondence between $\mathbf{H}_d$ and the current set of active cuts. Note also that there is a direct correspondence between W_p in $\mathbf{P}$ and S_p in $\mathbf{H}_d$, and, therefore, we refer to W_p and S_p interchangeably. Initially, we have the route W_d passing through an end point d of f and it corresponds to the shortest path S_d in $\mathbf{H}_d$.

Let $p = d$, and suppose we move the point p slightly toward the other end point d' of f. We are interested in the structural changes that occur to W_p as the sliding proceeds in $\mathbf{H}_d$. At certain key points, called the *event points*, we have to update the route because its structure changes. Since the sliding point p is not fixed, we have to be able to look ahead along f to compute the next event point. Therefore, we have to ensure that the fragment f is part of the boundary of $\mathbf{H}_d$, but this follows, since we assume that the route W_p reflects on f, and, hence, that f is active and part of the boundary of $\mathbf{H}_d$.

The next lemma describes the structure of the watchman routes that we compute.

Lemma 4.3. *The shortest path S_p in $\mathbf{H}_d$ makes turns at vertices that correspond to active fragment end points in* $\mathbf{P}$ *or to vertices of* $\mathbf{P}$.

Proof. Follows directly from Lemma 3.3, since perfect reflections of W_p in $\mathbf{P}$ translate to straight line segments of S_p in $\mathbf{H}_d$. □

We define the event points formally.

Definition 4.1. An *event point* is a point p on f such that one of the following properties holds:

1. The interior of either the first or last segment of S_p intersects a vertex of $\mathbf{H}_d$.
2. The first and the last segments of S_p have the same angle to f. This corresponds to perfect reflection in the interior of f.

Refer to Fig. 7 for an illustration of the different types of event points. We refer to the event point types by their corresponding number as above.

The shortest watchman route either reflects on a fragment end point or at an event point in the interior of an active fragment. This is shown in the next lemma.

Lemma 4.4. *Between two consecutive event points on a fragment f, the path S_p makes turns at the same vertices of* $\mathbf{H}_d$, *and the length of S_p either increases or decreases monotonically.*

Proof. Let q and r be two consecutive event points on f. We first prove that, for every point p between q and r, the shortest path S_p makes turns at the same points. Assume

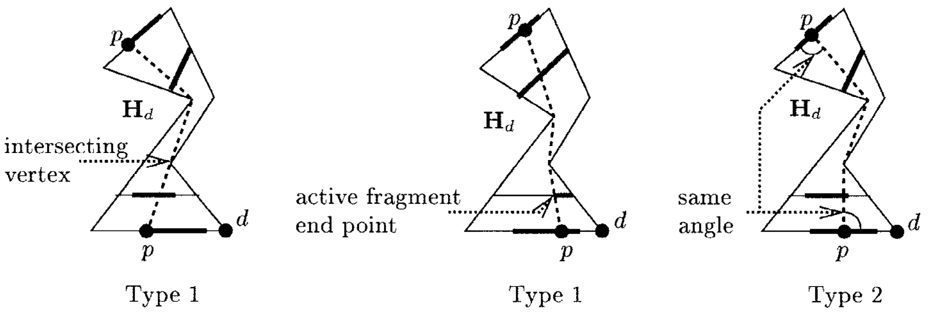

Fig. 7. The event points of the sliding process.

the contrary, that the turning points are not the same. Let p and p' be two points lying between q and r such that p is reached before p' as the sliding proceeds from q to r. Since subpaths of shortest paths are also shortest paths, the two paths S_p and $S_{p'}$ either do not intersect or they have one common subpath, the common subpath possibly degenerating to a single point of intersection. Now, if the two paths have a different turning point, then, evidently, this point cannot lie on the common subpath. Hence, it lies either before or after the common subpath. These cases are symmetric, so we assume that a different turning point lies before the common subpath.

To simplify the argument, we assume that p and p' have been selected close enough so that S_p and $S_{p'}$ only have one different turning point.

If S_p makes a turn at some point v, but $S_{p'}$ does not, then extend the second link of S_p until it hits f at the point p''; see Fig. 8(a). The path $S_{p'}$ cannot intersect the segment $[p'', v]$, because that would imply that v lies after the common subpath. This in turn means that p'' lies between p and p', but this is a contradiction, since, by Definition 4.1, p'' is a Type 1 event point.

If $S_{p'}$ makes a turn at some point v, but S_p does not, then extend the second link of $S_{p'}$ until it hits f at the point p''; see Fig. 8(b). The path S_p cannot intersect the segment $[p'', v]$, by the argument stated above. Again, p'' must lie between p and p', and this leads to a contradiction, since p'' is a Type 1 event point.

Thus, we have proved that S_p and $S_{p'}$ differ only by their first and last segments, and since p and p' were chosen arbitrarily between q and r, this also holds for S_q and S_r.

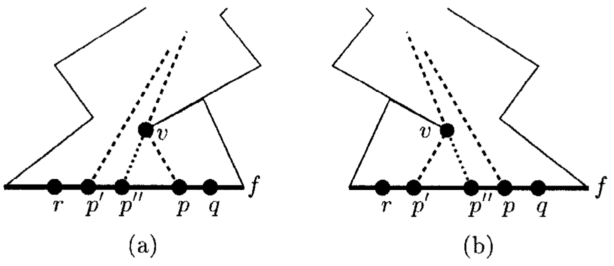

Fig. 8. Illustrating Lemma 4.4.

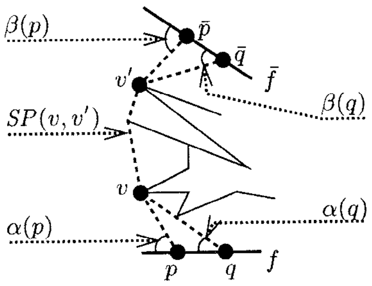

Fig. 9. Illustrating Lemma 4.4.

To prove that the length of S_p changes monotonically as p slides from q to r, we define the function

$$\mathcal{L}_q(p) = length(S_p) - length(S_q),$$

where we view $\mathcal{L}_q(p)$ as a one parameter function, with p between q and r on f. If we denote the shortest path between any two points u and u' in $\mathbf{H}_d$ by $SP(u, u')$, we have that $length(S_q) = \|q, v\| + length(SP(v, v')) + \|v', \bar{q}\|$ and $length(S_p) = \|p, v\| + length(SP(v, v')) + \|v', \bar{p}\|$, where $SP(v, v')$ is the common subpath of S_q and S_p, and the two points $\bar{q}$ and $\bar{p}$ are the images of q and p on the image $\bar{f}$ of f in $\mathbf{H}_d$; see Fig. 9. Hence,

$$\mathcal{L}_q(p) = \|p, v\| + \|v', \bar{p}\| - \|q, v\| - \|v', \bar{q}\|.$$

If we define the two angle functions $\alpha(p)$ and $\beta(p)$ as the angle between $[p, v]$ and f, and the angle between $[\bar{p}, v']$ and f', it is easy to show that

$$\mathcal{L}_q(p) = \frac{\|q, v\| \sin\alpha(q)}{\sin\alpha(p)} + \frac{\|v', \bar{q}\| \sin\beta(q)}{\sin\beta(p)} - \|q, v\| - \|v', \bar{q}\|.$$

We differentiate with respect to $\alpha(p)$ and set to zero to obtain the local optimum, yielding

$$\alpha(p) = \pi - \beta(p).$$

Since the sum of $\alpha(p)$ and $\beta(p)$ is π, the local optimum occurs when the segments $[p, v]$ and $[v', \bar{p}]$ in $\mathbf{H}_d$ correspond to segments in $\mathbf{P}$ making a perfect reflection on f. However, these points are defined as the Type 2 event points, and, hence, the function $\mathcal{L}_q(p)$ is monotone between two consecutive event points. □

We can actually make the following stronger statement

Corollary 4.5. *Between any pair of consecutive Type* 1 *event points there is at most one event point of Type* 2.

Proof. A careful examination of the function $\mathcal{L}_q(p)$ defined for p in the interval between two consecutive Type 1 event points shows that it can have at most one local optimum, thus proving the result. □

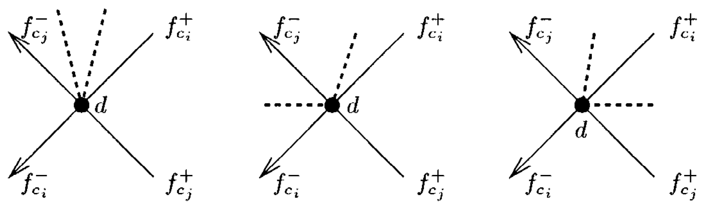

Fig. 10. The fragments adjacent to d are $f_{c_i}^+$, $f_{c_j}^+$, $f_{c_i}^-$, and $f_{c_j}^-$.

Adjusting the Route

Now, the question is what type of changes are to be made when p reaches an event point on f.

Consider the intersection point d between two essential cuts c_i and c_j. The point d is either the end point of four fragments of c_i and c_j or the end point of one fragment c_i lying on the boundary of **P**. In the previous section we showed how to compute the shortest watchman route reflecting at each fragment end point, so we assume that this route and the corresponding hourglass $\mathbf{H}_d$ together with the current set of active fragments are given.

We perform the sliding process at most four times starting at d, once for each active fragment adjacent to d, and in the direction of the opposite fragment end point. We denote the four fragments of c_i and c_j by $f_{c_i}^+$, $f_{c_j}^+$, $f_{c_i}^-$, and $f_{c_j}^-$; see Fig. 10. From the previous section we know that there are three different cases to handle: if both c_i and c_j are active, then we slide once along each fragment $f_{c_i}^+$, $f_{c_j}^+$, $f_{c_i}^-$, and $f_{c_j}^-$; see Fig. 11(a). If only c_i is active, then we slide along $f_{c_i}^+$ and $f_{c_i}^-$; see Fig. 11(b). The third case occurs when only c_j is active, and we slide along $f_{c_j}^+$ and $f_{c_j}^-$.

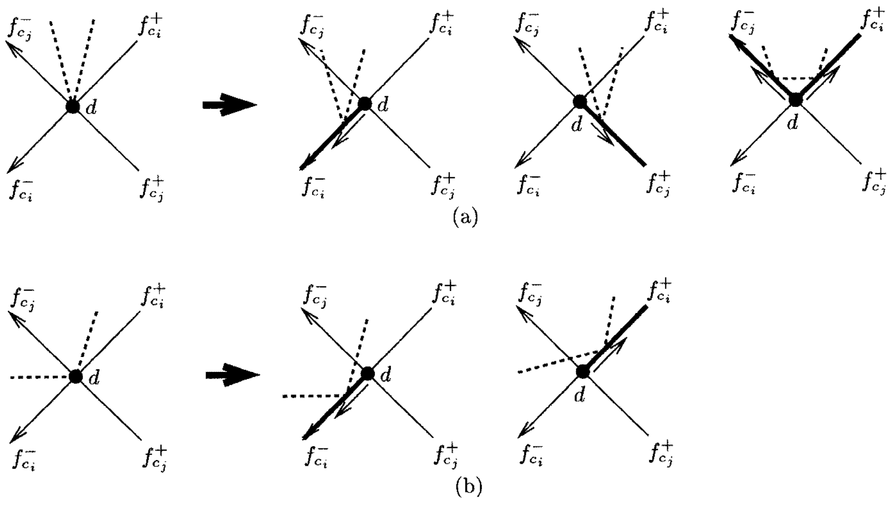

Fig. 11. The different cases of sliding depending on the initial route W_d.

As the sliding of a point p proceeds along one of the fragments f we encounter event points in sequence. Each event point requires some update of the path S_p. For the Type 1 event points, there are two possible updates. Either the first or the last segment of S_p is leaving a vertex or fragment end point v at the event point, in which case the two edges of S_p adjacent to v are collinear and can be merged into one first or last segment of S_p. The other case occurs when the path reaches a vertex or fragment end point v at the event point. In this case the edge $[p, u]$ of S_p intersecting v is split into one first or last edge $[p, v]$ and one second or penultimate edge $[v, u]$ of S_p.

The Type 2 event points do not induce any change in the path, and, hence, no updates are necessary. These points give local optima of the route length and are therefore interesting to maintain.

Computing the Event Points

To be able to perform the adjustments correctly, we need to compute the set of event points on a fragment f efficiently. This is done by first computing an ordered list of the Type 1 event points and then as the sliding process moves through the points in the list compute a potential Type 2 event point. The actual next event point is always the one closest to the sliding point p. As soon as one of the event point is reached by p, the proper changes and updates are performed and we compute a new potential next event point of Type 2.

Type 1 *Event Points*. Before the sliding on f starts, we do some preprocessing to obtain the list of Type 1 event points. Let d be the end point of f where the sliding process starts. Let $\bar{d}$ be the image of d on $\bar{f}$ in $\mathbf{H}_d$. Furthermore, let d' be the other end point of f and let $\bar{d}'$ be the image of d' on $\bar{f}$. We compute the shortest path trees rooted at d, d', $\bar{d}$, and $\bar{d}'$ in $\mathbf{H}_d$ [7]. Denote these by $\mathcal{SPT}_d$, $\mathcal{SPT}_{d'}$, $\mathcal{SPT}_{\bar{d}}$, and $\mathcal{SPT}_{\bar{d}'}$, respectively.

In each of the trees we compute the nearest common ancestor [9] with respect to the *opposite* fragment end points in $\mathbf{H}_d$. For $\mathcal{SPT}_d$ and $\mathcal{SPT}_{d'}$ these are the image points $\bar{d}$ and $\bar{d}'$, whereas for $\mathcal{SPT}_{\bar{d}}$ and $\mathcal{SPT}_{\bar{d}'}$ they are the fragment end points d and d'. Consider for example the shortest path tree $\mathcal{SPT}_{\bar{d}}$. Here we compute the nearest common ancestor v to d and d'; see Fig. 12(a). Thus, we have a path in the tree from d to v and a path from d' to v. Extend each link of these two paths and compute the intersection points with f; see Fig. 12(b). These intersection points are given in order along f and are inserted in a list. We perform the same construction with respect to the other shortest path trees, except that for $\mathcal{SPT}_d$ and $\mathcal{SPT}_{d'}$, we get intersection points on $\bar{f}$. These have image points on f that can be easily computed and inserted in corresponding lists. Now we have four ordered lists of points on f that can be merged together into one ordered list $\mathcal{L}_1$ of points. Note that the points d and d' belong to $\mathcal{L}_1$.

We assume from now on that for each point p in $\mathcal{L}_1$ we have the following information stored in association to p, *image*(p) is the image point $\bar{p}$ on $\bar{f}$, a flag *flag*(p) that tells which shortest path tree generated the point p, the two points $u(p)$ and $u'(p)$ corresponding to the end points of the shortest path tree edge that generated p if *flag*(p) is $\mathcal{SPT}_{\bar{d}}$ or $\mathcal{SPT}_{\bar{d}'}$, or $\bar{p}$ if *flag*(p) is $\mathcal{SPT}_d$ or $\mathcal{SPT}_{d'}$, the ordering is such that $u(p)$ is the middle point of the three; see Fig. 12(b). Furthermore, we assume that *next*(p) and *prev*(p) give the next and previous point in $\mathcal{L}_1$ according to the order from d to d'. In addition, we have

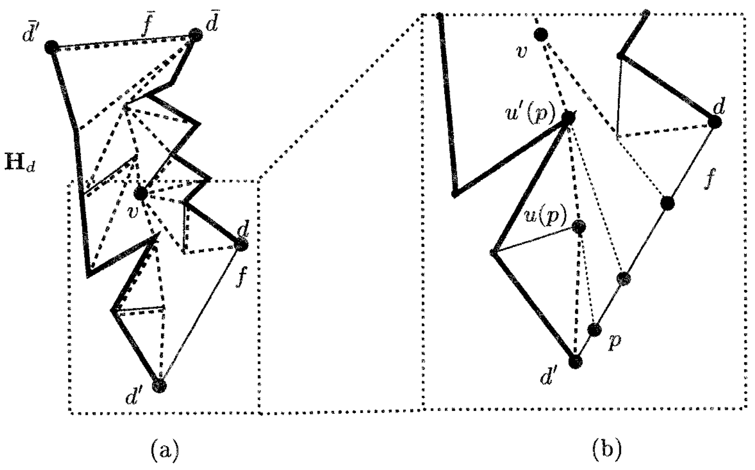

Fig. 12. Illustrating the computation of Type 1 event points from $\mathcal{SPT}_{\bar{d}}$.

a status field associated to every vertex of $\mathbf{H}_d$. For a vertex v in $\mathbf{H}_d$, the field *status*(v) can either have the value T or U depending on whether v is touched or untouched by the current shortest path S_p.

In most cases the list $\mathcal{L}_1$ will contain the complete set of Type 1 event points, however, in a few cases we might have missed some of them. Our next objective is therefore twofold: find a compact representation of the route S_p, for each point p in $\mathcal{L}_1$, and identify when we have missed Type 1 event points and compute these.

The compact representation of the routes consists of the complete shortest path S_d and $S_{d'}$ and, for every other point p in $\mathcal{L}_1$, the second and penultimate points of S_p, denoted *first*(p) and *last*(p), so that $[p, \mathit{first}(p)]$ and $[\mathit{last}(p), \bar{p}]$ are the first and last links of S_p. We also maintain the length of S_p, $\mathit{length}(p) = \mathit{length}(S_p)$.

Consider the case when S_d and $S_{d'}$ in $\mathbf{H}_d$ have a nondegenerate subpath in common, by which we mean a common subpath that starts and ends at vertices of $\mathbf{H}_d$. In this case this subpath is going to be a part of each path S_p, for every point p on f. Let v and v' denote the end points of this subpath, with v lying before v' as S_p is traversed from p on f to the image point $\bar{p}$ on $\bar{f}$. This means that an event point is an intersection point between an extension of an edge of the shortest path from v to d or from v to d' with the segment f, or the image on f of an intersection point between an extension of an edge of the shortest path from v' to $\bar{d}$ or from v' to $\bar{d}'$ with the segment $\bar{f}$. However, these points are all included in the list $\mathcal{L}_1$ as can be seen from the construction of $\mathcal{L}_1$, since v and v' will be the nearest common ancestors of d and d' in $\mathcal{SPT}_{\bar{d}}$ and $\mathcal{SPT}_{\bar{d}'}$ and $\bar{d}$ and $\bar{d}'$ in $\mathcal{SPT}_d$ and $\mathcal{SPT}_{d'}$ respectively.

The compact representation of S_p can now be easily computed for the points in $\mathcal{L}_1$ in sequence from d to d', if we assume that all the vertices of $\mathbf{H}_d$ have their status fields set to U except for the vertices along S_d which have their status fields set to T: if *flag*(p) equals $\mathcal{SPT}_{\bar{d}}$ or $\mathcal{SPT}_{\bar{d}'}$, then if $\mathit{status}(u(p)) = T$, we have that S_p is leaving the vertex

$u(p)$ and set $\mathit{first}(p) := u'(p)$, $\mathit{last}(p) := \mathit{last}(\mathit{prev}(p))$, $\mathit{status}(u(p)) := U$, and

$$\begin{aligned}\mathit{length}(p) \;:=\; & \mathit{length}(\mathit{prev}(p)) - \|u(p), u'(p)\| - \|\mathit{prev}(p), \mathit{first}(\mathit{prev}(p))\| \\ & - \|\mathit{image}(\mathit{prev}(p)), \mathit{last}(\mathit{prev}(p))\| + \|p, \mathit{first}(p)\| + \|\mathit{image}(p), \mathit{last}(p)\|.\end{aligned}$$

On the other hand, if $\mathit{status}(u(p)) = U$, we have that S_p is hitting the vertex $u(p)$ and therefore we set $\mathit{first}(p) := u(p)$, $\mathit{last}(p) := \mathit{last}(\mathit{prev}(p))$, $\mathit{status}(u(p)) := T$, and

$$\begin{aligned}\mathit{length}(p) \;:=\; & \mathit{length}(\mathit{prev}(p)) - \|\mathit{prev}(p), \mathit{first}(\mathit{prev}(p))\| \\ & - \|\mathit{image}(\mathit{prev}(p)), \mathit{last}(\mathit{prev}(p))\| + \|p, \mathit{first}(p)\| + \|u(p), u'(p)\| \\ & + \|\mathit{image}(p), \mathit{last}(p)\|.\end{aligned}$$

The case when $\mathit{flag}(p)$ equals $\mathcal{SPT}_d$ or $\mathcal{SPT}_{d'}$ is handled in the same way with respect to the image of p. The following lemma shows that we have considered all cases.

Lemma 4.6. *If S_d and $S_{d'}$ have a nondegenerate subpath in common, then the list $\mathcal{L}_1$ contains all the Type 1 event points.*

Proof. Let r be a Type 1 event point. By definition, the path S_r has either a first or last link that intersects a vertex of $\mathbf{H}_d$. Assume first that it holds for the first link and let the vertex be u. The common subpath of S_d and $S_{d'}$ must also be a subpath of S_r, and, hence, the end vertex v closest to r of this subpath also belongs to S_r. Since v is the nearest common ancestor of d and d' in both $\mathcal{SPT}_{\bar{d}}$ and $\mathcal{SPT}_{\bar{d}'}$, the two trees also contain the path from v to u. Hence, r is the intersection between f and the extension of the last shortest path tree edge on the path from v to u. Since $\mathcal{L}_1$ contains all these intersection points, the point r belongs to $\mathcal{L}_1$.

If it is the last link of S_r that intersects a vertex of $\mathbf{H}_d$, then we argue in a similar manner to show that the image point $\bar{r}$ of r on $\bar{f}$ is the intersection point of $\bar{f}$ and an extension of a shortest path edge in $\mathcal{SPT}_d$ and $\mathcal{SPT}_{d'}$, thus r belongs to $\mathcal{L}_1$. □

When S_d and $S_{d'}$ do not have a nondegenerate subpath in common, there may exist points p on f such that S_p are straight line segments in $\mathbf{H}_d$, and, hence, the paths S_p do not touch any of the vertices of $\mathbf{H}_d$.

We can check for this possibility by comparing the nearest common ancestor of d and d' in $\mathcal{SPT}_{\bar{d}}$ and the nearest common ancestor of d and d' in $\mathcal{SPT}_{\bar{d}'}$. If they are equal, then S_d and $S_{d'}$ have a common nondegenerate subpath and, by Lemma 4.6, the list $\mathcal{L}_1$ contains all the Type 1 event points.

If S_d and $S_{d'}$ do not have a common subpath, then we have to differentiate between two subcases.

(i) *Even number of active fragments*. If W_d in $\mathbf{P}$ corresponding to S_d in $\mathbf{H}_d$ has an even number of reflection points on active fragments, i.e., there is an even number of active fragments, the two points p and $\bar{p}$ will slide in the same general direction; see Fig. 13. In this case we traverse the current list $\mathcal{L}_1$ while maintaining the compact representation of the shortest path S_p for each event point p in $\mathcal{L}_1$ in the same way as was shown previously.

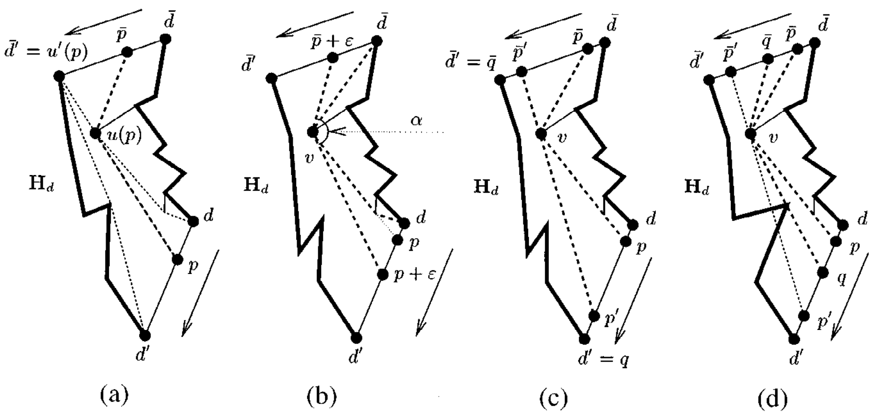

Fig. 13. Illustrating the computation of the compact representation of the shortest paths with an even number of active fragments.

Furthermore, we maintain a counter *turns*(p) on the number of turns that S_p makes. The counter *turns*(p) is either increased by one or decreased by one depending on whether S_p hits a vertex or leaves a vertex at p.

In this case we may run into the following problems: consider a point p in $\mathcal{L}_1$ with *flag*(p) equaling $\mathcal{SPT}_{\bar{d}}$ or $\mathcal{SPT}_{\bar{d}'}$. It may be that neither $u(p)$ nor $u'(p)$ is equal to *first*(*prev*(p)) or that $u(p)$ does equal *first*(*prev*(p)) but the segment $[p, u'(p)]$ is not part of S_p; see Fig. 13(a) for an example. The problem here is that S_p consists of links generated by say $\mathcal{SPT}_{\bar{d}}$ whereas the point p is an intersection point generated by the extension of a link from $\mathcal{SPT}_{\bar{d}'}$, that is the other shortest path tree. This has to be recognized and handled by our algorithm, and it is solved in the following way.

Assume that *flag*(p) equals $\mathcal{SPT}_{\bar{d}}$ or $\mathcal{SPT}_{\bar{d}'}$, then if *status*$(u'(p)) = T$, we can establish the compact representation of S_p as previously shown. On the other hand, if *status*$(u'(p)) = U$, then S_p does not pass through the vertex $u'(p)$, and, hence, the point p cannot be a proper event point. In this case we compute the compact representation for S_p by setting *first*(p) := *first*(*prev*(p)), *last*(p) := *last*(*prev*(p)), and compute the length of S_p accordingly. If *flag*(p) equals $\mathcal{SPT}_d$ or $\mathcal{SPT}_{d'}$, we have the same situation with respect to the image $\bar{p}$ and we can handle this case in a similar way.

When S_p makes only one turn, i.e., *turns*$(p) = 1$, for some p in $\mathcal{L}_1$, then the two links of the path may merge into one and our aim is to find the point on f where this happens. It is a point $p + \varepsilon$, for some value of ε, where the path is a straight line segment. Let $v = u(p)$ denote the turning point in the path S_p and let α be the angle $pv\bar{p}$. Similarly we denote the image of $p+\varepsilon$ on $\bar{f}$ by $\bar{p}+\varepsilon$. Thus, we can express the angle $(p+\varepsilon)v(\bar{p}+\varepsilon)$ as a one parameter function $\alpha(\varepsilon)$ of ε; see Fig. 13(b). Furthermore, we have that the scalar product of the two vectors $\overline{v(p+\varepsilon)}$ and $\overline{v(\bar{p}+\varepsilon)}$ equals

$$\overline{v(p+\varepsilon)} \times \overline{v(\bar{p}+\varepsilon)} = \cos\alpha(\varepsilon)\|v, p+\varepsilon\| \cdot \|v, \bar{p}+\varepsilon\|$$

and we are interested in the value of ε when $\alpha(\varepsilon) = \pi$. To get this value we simply solve the equation

$$\overline{v(p+\varepsilon)} \times \overline{v(\bar{p}+\varepsilon)} + \|v, p+\varepsilon\| \cdot \|v, \bar{p}+\varepsilon\| = 0.$$

Set $p' := p + \varepsilon$ and check if p' lies between p and q on f, where q is the point in $\mathcal{L}_1$ after p as $\mathcal{L}_1$ is traversed from d to d' such that $\mathit{flag}(q) = \mathcal{SPT}_{\bar{d}}$; see Fig. 13(c). If p' lies between p and q on f, then p' is inserted as a Type 1 event point in its proper place between p and q in $\mathcal{L}_1$, the possible points in $\mathcal{L}_1$ between p and p' are updated with the compact representation of their corresponding paths (each of them pass through the single vertex v), $\mathit{first}(p') := \bar{p}'$ and $\mathit{length}(p') := \|p', \bar{p}'\|$. Otherwise p' is discarded since the line segment $[p', \bar{p}']$ crosses the exterior of $\mathbf{H}_d$; see Fig. 13(d).

If p' does not lie between p and q, then there is some point p'' between p' and q on f where $S_{p''}$ hits some vertex and goes from being a straight line segment to having an increasing number of turns. To get p'' we simply start with the path $S_p = S_{d'}$, traverse $\mathcal{L}_1$ backward from d' to d, maintain a counter on the number of turns that S_p makes, and perform similar operations as previously shown.

(ii) *Odd number of active fragments*. If there is an odd number of active fragments that W_d corresponding to S_d in $\mathbf{H}_d$ reflect on, then the points p and $\bar{p}$ will slide in opposite directions. This means that S_d and $S_{d'}$ will always intersect in $\mathbf{H}_d$, i.e., they have a common subpath. However, the subpath may be degenerate in the sense that it is an intersection point in the interior of $\mathbf{H}_d$ that does not correspond to a vertex of $\mathbf{H}_d$; see Fig. 14(a). In this case we scan for missing Type 1 event points and construct the compact representations of the shortest paths in essentially the same way as when we have an even number of active fragments.

We traverse the list $\mathcal{L}_1$ from d to d' maintaining a compact representation of the shortest path S_p, for each event point p in $\mathcal{L}_1$, and a counter on the number of turns that S_p makes as $\mathcal{L}_1$ is traversed. The compact representation of the paths is computed as before.

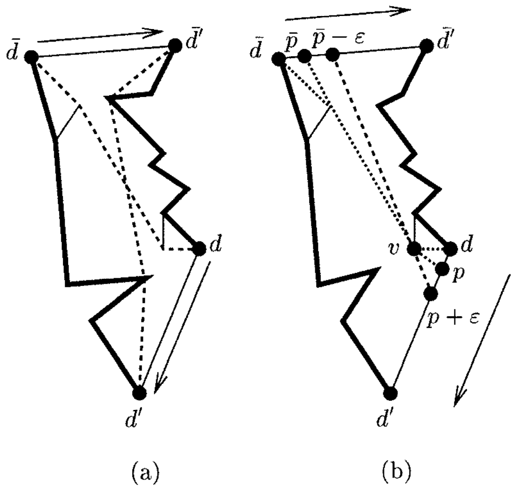

Fig. 14. Illustrating the computation of the compact representation of the shortest paths with an odd number of active fragments.

When S_p makes exactly one turn, for some p in $\mathcal{L}_1$, then we might, once again, have the case that for one point $p+\varepsilon$ the path $S_{p+\varepsilon}$ becomes a straight line segment from $p+\varepsilon$ to its image point on $\bar{f}$. Such a point can be computed by solving the equation

$$\overline{v(p+\varepsilon)} \times \overline{v(\bar{p}-\varepsilon)} + \|v, p+\varepsilon\| \cdot \|v, \bar{p}-\varepsilon\| = 0,$$

for ε, where v is the single turning point of S_p and $\bar{p}-\varepsilon$ should be considered as the image point of $p+\varepsilon$ since sliding on the two fragments f and $\bar{f}$ is done in opposite directions.

Set $p' := p+\varepsilon$ and check if p' lies between p and q on f, where q is the point in $\mathcal{L}_1$ after p as $\mathcal{L}_1$ is traversed from d to d'; see Fig. 14(b). If p' lies between p and q on f, then p' is inserted as a Type 1 event point between p and q in $\mathcal{L}_1$, otherwise p' is discarded and the sliding process is continued.

If p' does not lie between p and q, then there is some point p'' between p' and q on f where $S_{p''}$ hits some vertex and goes from being a straight line segment to having an increasing number of turns. To get p'' we simply start with the path $S_p = S_{d'}$, traverse $\mathcal{L}_1$ backward from d' to d, maintain a counter on the number of turns that S_p makes, and perform the same operations as previously shown. We have the following lemma.

Lemma 4.7. *If S_d and $S_{d'}$ do not have a nondegenerate subpath in common, then the list $\mathcal{L}_1$ including the end points of the interval on f where S_p is a straight line segment contains all the Type* 1 *event points.*

Proof. Let r be a Type 1 event point. If r is an intersection between f and the extension of some shortest path tree edge of $\mathcal{SPT}_{\bar{d}}$ or $\mathcal{SPT}_{\bar{d}'}$, or the image of an intersection between $\bar{f}$ and the extension of some shortest path tree edge of $\mathcal{SPT}_d$ or $\mathcal{SPT}_{d'}$, then r belongs to $\mathcal{L}_1$ by the proof of Lemma 4.6. Our objective is now to show that if this is not the case, then r is one of the end points of the interval on f where S_p is a straight line segment. By definition, the path S_r has either a first or last link that intersects a vertex of $\mathbf{H}_d$. Assume first that this holds for the first link and let the vertex be u. Since r is not an intersection point between f and the extension of some shortest path tree edge of $\mathcal{SPT}_{\bar{d}}$ or $\mathcal{SPT}_{\bar{d}'}$, this means that S_r cannot pass through any other vertex of $\mathbf{H}_d$ except u and therefore S_r must be a straight line segment, proving our claim.

If it is the last link of S_r that intersects a vertex of $\mathbf{H}_d$, then the situation is exactly the same as before, whereby the result follows. □

Type 2 *Event Points.* The list $\mathcal{L}_2$ of Type 2 event points is computed in the following way. The hourglass $\mathbf{H}_d$ is a triangulated two-manifold embedded in the plane. Hence, $\mathbf{H}_d$ can be copied with f as the image by taking each triangle in $\mathbf{H}_d$ on the way from $\bar{f}$ to f and placing it before f; see Fig. 15. This process gives us a new two-manifold $\mathbf{H}_d^+$ consisting of two versions of $\mathbf{H}_d$, the original one and a new one denoted $\mathbf{H}_d'$. Each vertex v of $\mathbf{H}_d$ corresponds to an image vertex in $\mathbf{H}_d'$ denoted $below(v)$.

Now we run through the list $\mathcal{L}_1$ from d to d' and for each point p in $\mathcal{L}_1$ we compute the intersection between the segment $[first(p), below(last(p))]$ and f to get a point

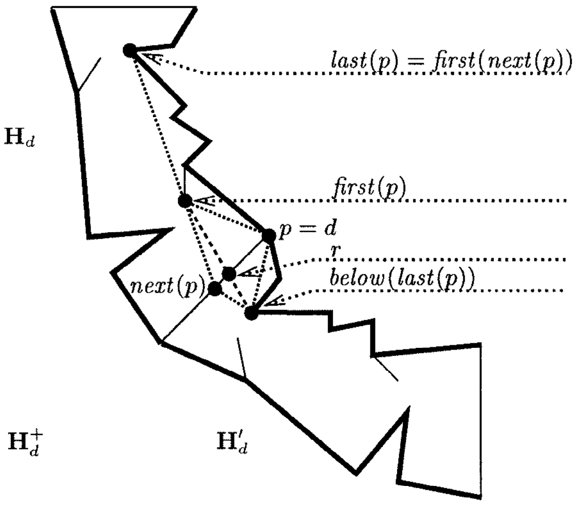

Fig. 15. Illustrating the computation of the Type 2 event points.

r. If r lies between p and $next(p)$, then r is inserted in $\mathcal{L}_2$ and the length of S_r is computed by

$$length(r) := length(p) - \|p, first(p)\| - \|image(p), last(p)\| + \|first(p), below(last(p))\|,$$

otherwise, the point r is discarded.

Lemma 4.8. *The list $\mathcal{L}_2$ contains all the Type 2 event points.*

Proof. By definition, the Type 2 event points are the points r on f where the first and last links have the same angle to f and $\bar{f}$. By Corollary 4.5 there can be at most one event point of Type 2 between any pair of consecutive Type 1 event points. Since the described process checks for intersection points on f having the same angles for all pairs of consecutive Type 1 event points, the claim follows. □

Finally, we run through the two lists $\mathcal{L}_1$ and $\mathcal{L}_2$ and establish the point p for which $length(p)$ is smallest, compute the path S_p, and return it as the result of the sliding process.

Complexity of the Sliding Process. Next, we prove that the sliding process on a fragment f can be performed in linear time.

Lemma 4.9. *The number of event points on a fragment f is $O(n)$ and they can be computed in linear time.*

Proof. We count the number of event points for each type separately.

1. The number of Type 1 event points is $O(n)$, since the number of points where S_p intersects vertices of $\mathbf{H}_d$ is at most linear. That they can be computed in linear time follows from the discussion above, since the set $\mathcal{L}_1$ can be computed from the shortest path trees in linear time and each of the four shortest path trees can be constructed in linear time. Updating the *first*(p) and *last*(p) fields takes constant time per event point.
2. The number of Type 2 event points is at most as large as the number of Type 1 event points. This is because between two consecutive event points of Type 1, there can be at most one Type 2 event point by Corollary 4.5, i.e., there is at most one local optimum in that interval. Thus, the Type 2 event points contribute at most a factor 2 to the total number of event points. Computing the set $\mathcal{L}_2$ of Type 2 event points is done by first computing the two-manifold $\mathbf{H}_d^+$, which can be done in linear time, and then testing for Type 2 event points in each interval between consecutive Type 1 event points, which takes constant time per interval.

Thus, the sum over all the event point types and the computation time is as stated, which concludes the proof. □

4.5. *Correctness and Analysis*

During the sliding process on a fragment f, we maintain the shortest path S_p in $\mathbf{H}_d$, and S_p corresponds to the shortest watchman route W_p in $\mathbf{P}$ reflecting on the same active cuts as W_d.

To prove the correctness of our algorithm we use the following lemma.

Lemma 4.10. *There is a fragment end point d such that the shortest watchman route W reflects on the same active cuts as W_d.*

Proof. The proof divides into two cases. The first case arises when W intersects a fragment end point, in which case the lemma obviously holds.

The second case occurs when W does not intersect any fragment end point. This means, by Lemma 3.3, that W makes perfect reflections at all its active cuts. Let p be one of the reflection points of W and let f be the active fragment containing p. Let $\mathbf{H}_p$ be the hourglass obtained by unfolding the polygon with respect to the active cuts of W and let S_p be the shortest path in $\mathbf{H}_p$ corresponding to W. Now, imagine that the point p moves in one direction along f and that the shortest path changes accordingly until the path touches a fragment end point. Let p' be the point on f where this occurs, i.e., $S_{p'}$ intersects some fragment end point but for any point q in the interval between p and p' the path S_q does not intersect any fragment end point. Let d be the fragment end point intersected by $S_{p'}$. Our objective is to prove that W and W_d, the shortest watchman route through d, have the same set of active cuts.

Let $\mathbf{H}_p^+$ be the two-manifold obtained by placing a second copy of $\mathbf{H}_p$, denoted $\mathbf{H}'_p$, below $\mathbf{H}_p$ and, for each boundary point q in $\mathbf{H}_p$, let *below*(q) denote the image of q in $\mathbf{H}'_p$; see Fig. 16. Let $SP(d, below(d))$ be the shortest path from d to *below*(d)

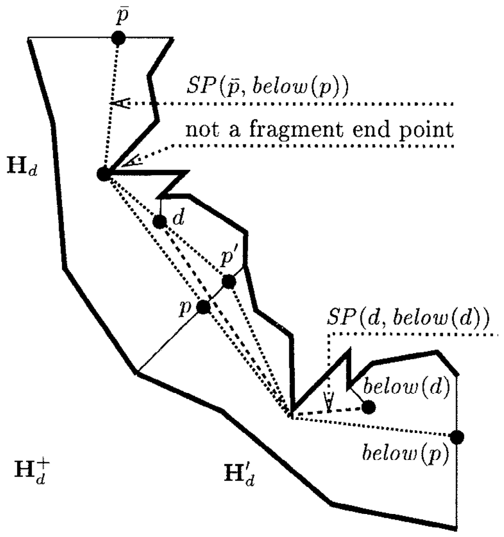

Fig. 16. Illustrating the proof of Lemma 4.10.

in $\mathbf{H}_p^+$. Let $SP(\bar{p}, below(p))$ be the shortest path from $\bar{p}$ to $below(p)$, i.e., S_p in $\mathbf{H}_p$ together with its copy in $\mathbf{H}'_p$. Since $SP(\bar{p}, below(p))$ does not have a turning point at p on f, the path $SP(d, below(d))$ must cross f at a point between p and p' on f, and, hence, $SP(d, below(d))$ does not intersect any fragment end point in $\mathbf{H}_p^+$, i.e., the path $SP(d, below(d))$ corresponds to some watchman route that passes through d and makes perfect reflections on all its active cuts; see Fig. 16 for an illustration. Denote this route by W'_d.

Now, Chin and Ntafos [6] prove that if a fixed watchman route is nonadjustable, i.e., the route cannot be locally shortened, then the route is the shortest fixed watchman route. Since W'_d is nonadjustable it therefore follows that $W'_d = W_d$, and, hence, W and W_d reflect on the same active cuts. $\square$

To get the shortest watchman route we perform sliding along all fragments allowing reflection contact, and maintain the shortest route obtained at the event points. It follows from Lemma 4.4 that the shortest route will pass through one of the event points since the length of a route is either monotonically increasing or decreasing as the sliding process proceeds between event points.

We prove the following theorem.

Theorem 2. *The Shortest-Floating-Watchman-Route algorithm computes a shortest floating watchman route in a simple polygon* $\mathbf{P}$ *of* n *edges in* $O(F(n + T(n, |\mathcal{C}|, F)) + n \log n)$ *time using* $O(S(n, |\mathcal{C}|, F) + Fn)$ *storage, where* $T(n, |\mathcal{C}|, F)$ *is the time and* $S(n, |\mathcal{C}|, F)$ *is the storage used to compute the shortest fixed watchman route in a polygon having at most* n *edges,* $|\mathcal{C}|$ *essential cuts, and* F *fragments of the essential cuts.*

Proof. The correctness of the algorithm follows from Lemmas 4.6–4.8, 4.10, and the discussion above. Hence, it remains to analyze the complexity of the algorithm.

Step 1 can be performed in $O(n \log n + F)$ time and $O(n + F)$ storage. Step 2.1 takes $O(T(n, |\mathcal{C}|, F))$ time and $O(S(n, |\mathcal{C}|, F))$ storage by definition. The loop at Step 2 is performed at most $3F$ times, and, hence, this step uses $O(F \cdot T(n, |\mathcal{C}|, F))$ time and $O(S(n, |\mathcal{C}|, F) + Fn)$ storage.

To show the complexity of Step 3.1, we have from Lemma 4.9 that the number of event points is $O(n)$ and they can be computed in linear time and storage. Thus, the time and storage complexities for Step 3 are both $O(Fn)$.

The total time for our algorithm becomes $O(F(n + T(n, |\mathcal{C}|, F)) + n \log n)$ and the storage becomes $O(S(n, |\mathcal{C}|, F) + Fn)$. $\square$

Since $T(n, |\mathcal{C}|, F) = O(n|\mathcal{C}|F)$ and $S(n, |\mathcal{C}|, F) = O(n|\mathcal{C}|)$ by Theorem 1, $|\mathcal{C}| = O(n)$, and $|\mathcal{C}| \leq F \leq |\mathcal{C}|^2$, the worst case running time and storage requirement of our algorithm is $O(n^6)$ and $O(n^3)$, respectively.

As a final remark, the bottleneck of our algorithm is the computation of the shortest watchman routes reflecting at every fragment end point. It may be possible to exploit the fact that between two fragment end points on an essential cut, the shortest route does not change much, and, therefore, all the routes could be computed faster. However, this approach remains to be investigated.

5. Conclusion

We have presented a polynomial time solution to the problem of computing the shortest floating watchman route in a simple polygon. The fact that there is a polynomial time solution for this problem settles an open question in computational geometry.

The algorithm is adaptive in the sense that the complexity depends not only of the size of the polygon but also of the number of essential cuts and the number of fragments. The number of fragments can be quadratic but often it will be much smaller, thus reducing the complexity of the algorithm.

Related problems are, for instance, those of computing several watchman routes in a polygon using different optimization criteria. Most versions of these problems turn out to be NP-hard but there exist polynomial time algorithms for some of these problems in certain restricted classes of polygons [3], [15], [16].

One important open question in this area is whether the problem of computing the two watchman routes in a polygon such that the sum of the lengths of the two routes is minimized has a polynomial time solution or not.

References

1. A. Aggarwal. The Art Gallery Theorem: Its Variations, Applications and Algorithmic Aspects. Ph.D. thesis, Johns Hopkins University, Baltimore, MD, 1984.
2. S. Carlsson, H. Jonsson, B. J. Nilsson. Finding the Shortest Watchman Route in a Simple Polygon. In *Proc.* 4*th International Symposium on Algorithms and Computation, ISAAC* '93, pages 58–67. Lecture Notes in Computer Science 762, Springer-Verlag, Berlin, 1993.

3. S. Carlsson, B. J. Nilsson, S. Ntafos. Optimum Guard Covers and m-Watchmen Routes for Restricted Polygons. *International Journal of Computational Geometry and Applications*, 3(1):85–105, 1993.
4. B. Chazelle. Triangulating a Simple Polygon in Linear Time. In *Proc*. 31*st Symposium on Foundations of Computer Science*, pages 220–230, 1990.
5. B. Chazelle, H. Edelsbrunner. An Optimal Algorithm for Intersecting Line Segments in the Plane. *Journal of the ACM*, 39(1):1–54, 1992.
6. W. Chin, S. Ntafos. Shortest Watchman Routes in Simple Polygons. *Discrete and Computational Geometry*, 6(1):9–31, 1991.
7. L. Guibas, J. Hershberger, D. Leven, M. Sharir, R. Tarjan. Linear Time Algorithms for Visibility and Shortest Path Problems Inside Triangulated Simple Polygons. *Algorithmica*, 2:209–233, 1987.
8. M. Hammar, B.J. Nilsson. Concerning the Time Bounds of Existing Shortest Watchman Route Algorithms. In *Proc*. 11*th International Symposium on Fundamentals in Computation Theory, FCT* '97, pages 210–221. Lecture Notes in Computer Science 1279, Springer-Verlag, Berlin, 1997.
9. D. Harel, R.E. Tarjan. Fast Algorithms for Finding Nearest Common Ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.
10. J. Hershberger, S. Suri. A Pedestrian Approach to Ray Shooting: Shoot a Ray, Take a Walk. In *Proc*. *SODA*, pages 54–63, 1993.
11. D. T. Lee, A. K. Lin. Computational Complexity of Art Gallery Problems. *IEEE Transactions on Information Theory*, IT-32:276–282, 1986.
12. D. T. Lee, F. P. Preparata. An Optimal Algorithm for Finding the Kernel of a Polygon. *Journal of the ACM*, 26:415–421, 1979.
13. D. T. Lee, F. P. Preparata. Euclidean Shortest Paths in the Presence of Rectilinear Barriers. *Networks*, 14:393–410, 1984.
14. E. A. Melissaratos, D. L. Souvaine. On Solving Geometric Optimization Problems Using Shortest Paths. In *Proc*. 6*th ACM Symposium on Computational Geometry*, pages 350–359, 1990.
15. J. S. B. Mitchell, E. L. Wynters. Watchman Routes for Multiple Guards. In *Proc*. 3*rd Canadian Conference on Computational Geometry*, pages 126–129, 1991.
16. B. J. Nilsson. Guarding Art Galleries—Methods for Mobile Guards. Ph.D. thesis, Lund University, 1995.
17. X.-H. Tan, T. Hirata. Constructing Shortest Watchman Routes by Divide and Conquer. In *Proc*. 4*th International Symposium on Algorithms and Computation*, pages 68–77. Lecture Notes in Computer Science 762, Springer-Verlag, Berlin, 1993.
18. X.-H. Tan, T. Hirata, Y. Inagaki. An Incremental Algorithm for Constructing Shortest Watchman Routes. *International Journal of Computational Geometry and Applications*, 3:351–365, 1993.
19. X.-H. Tan, T. Hirata, Y. Inagaki. Corrigendum to "An Incremental Algorithm for Constructing Shortest Watchman Routes". *International Journal of Computational Geometry and Applications*, to appear.

Received December 12, 1997, *and in revised form September* 30, 1998.