Routing in a Polygonal Terrain with the Shortest Beacon Watchtower

Bahram Kouhestani David Rappaport Kai Salomaa*

Abstract

In a paper by Biro et al. [2], a novel twist on guarding in art galleries, motivated by geographical greedy routing in sensor networks, is introduced. A beacon is a point that when activated induces a force of attraction, or repulsion that can move points within the environment. The effect of a beacon is similar to standard visibility with some additional properties. The effects of a beacon are asymmetric leading to separate algorithms to compute the "beacon kernel" and "inverse beacon kernel". In Biro [1] $O(n^2)$ algorithms are given to compute the beacon kernel and inverse beacon kernels in simple polygons. In this paper we revisit the problem of computing the shortest watchtower to guard a $2D$ terrain, using the properties of beacons, and we present an $O(n \log n)$ time algorithm that computes the shortest beacon watchtower. In doing this we introduce $O(n \log n)$ algorithms to compute the beacon kernel in a simple polygon and an inverse beacon kernel in a terrain polygon. Similar ideas are then used for an algorithm to compute the inverse beacon kernel in a monotone polygon in $O(n \log n)$ time.

1 Introduction

Motivated by routing in sensor networks, Biro et. al [2] introduced the notion of a *beacon*, as a new variation of visibility. A *beacon* is a point inside a polygon P that can induce a magnetic pull toward itself everywhere in P. When the beacon b is activated, points in P move greedily to minimize their Euclidean distance to b. A point p in the interior of P moves along the ray $\vec{pb}$ until it reaches b or hits the boundary of P. In the latter case it may still reduce its Euclidean distance to b by sliding on the boundary of P. A point in P is *attracted* by b if its Euclidean distance to b is eventually decreased to 0. The attraction region of a beacon b is the set of all points in P that b can attract. If a point $d \in P$ (which is not b) remains stationary in the presence of b, then d is called a dead point. The set of points ending up on d comprises the dead region of d with respect to b. In general, when a beacon is activated, the path of a point alternates between moving along the ray towards b and sliding along the boundary of P and eventually it either reaches b or gets stuck on a dead point. The boundary between two dead regions or a dead region and the attraction region of b is called a *split edge*.

Biro [1] showed that there are three different types of split edges, and all of them emanate from a reflex vertex. Note that a reflex vertex v will cause a split edge for a beacon b if both incident edges of v are located below the line perpendicular to $\overline{bv}$ at v (Fig 1). In our application, only the split edges between the boundary of the attraction region and a dead region are important. We call a split edge that separates the attraction region of the beacon from a dead region a *separation* edge (Fig 2). For more details on beacons see [1, 2, 3].

We say a point $x \in P$ is *routed* to a point $y \in P$ via b if b can attract x and y can attract a point located on b.

A $2D$ *terrain* T is a monotone polygonal chain with respect to the x-coordinate. A *watchtower* is a vertical line segment erected on T. The length of the watchtower is the length of the line segment. The notion of beacon visibility can be extended to a terrain environment and a *beacon watchtower* is a beacon located on the upper endpoint of a watchtower.

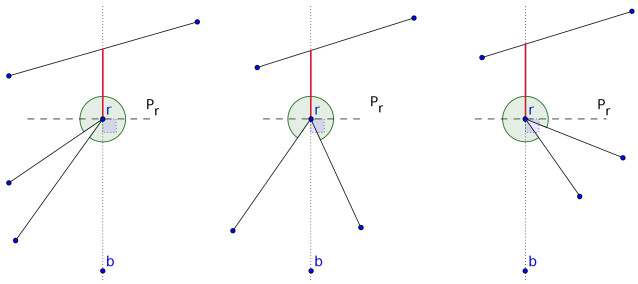

Figure 1: Three different types of split edges, from left to right 1) both edges incident to r are to the left of the line $\overline{rb}$ 2) one incident edge of r is to the left of $\overline{rb}$ and the other to the right 3) both are to the right of $\overline{rb}$ [1]. In all cases both edges are below the perpendicular line P_r. Split edges are shown in red.

*School of Computing, Queen's University, Kingston, Ontario, Canada. {kouhesta,daver,ksalomaa}@cs.queensu.ca

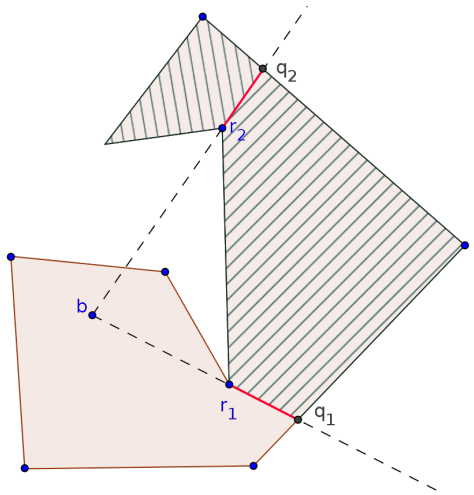

Figure 2: Attraction and dead regions (shaded regions) of a beacon b. Split edges are $\overline{r_1q_1}$ and $\overline{r_2q_2}$. $\overline{r_1q_1}$ is a separation edge while $\overline{r_2q_2}$ is not.

In this paper we address the following problem:
Shortest Beacon Watchtower: Given a $2D$ terrain T, the goal is to find the shortest beacon watchtower b, which guarantees that for any pair of points $x, y \in T$, x can be routed to y via b.

2 Computing the shortest beacon watchtower

Let $t_1, t_2, ..., t_n$ denote vertices of T from left to right. We construct a monotone polygon T' by adding two new vertices t_{n+1} and t_{n+2} to T with the same y-coordinate, where t_{n+1} has the same x-coordinate as t_n and t_{n+2} has the same x-coordinate as t_1 (See Fig 3). We call T' a terrain polygon. The y-coordinate value of these two points are chosen big enough so that for any points $p \in \overline{t_{n+1}t_{n+2}}$ and $q \in T$ the interior of the line segment $\overline{pq}$ does not intersect T. This can be done in linear time. Suppose l_i is the line going through the edge $\overline{t_it_{i+1}}$, and q_i and q_i' are the intersections of l_i with vertical lines going through t_1 and t_n, respectively. Let q be the point with the highest y-coordinate among all q_i and q_i' $(1 < i < n)$. We set the y-coordinates of t_{n+1} and t_{n+2} to be the y-coordinate of q. In this way, any pair of points on T can be routed via an arbitrary point on $\overline{t_{n+1}t_{n+2}}$.

The solution to the shortest beacon watchtower problem is a closest point to T which can attract all points on the terrain and in return is attracted by all points on the terrain. Therefore, by the definition of [1] the solution is located in the intersection of the beacon kernel and the inverse beacon kernel of T'.
The *beacon kernel* of a polygon P is the set of points in P that can attract all points in P. The *inverse beacon kernel* is the set of points that are attracted by all points in P. In general, due to the asymmetry of beacon attraction, these two sets are not equal. The beacon kernel and inverse beacon kernel of P can be computed in $O(n^2)$ time [1]. Both the beacon kernel and the inverse beacon kernel of P are convex with respect to P. A sub-polygon Q is convex with respect to P if the line segment connecting two arbitrary points of Q either completely lies in Q or intersects P. Note that Q may not be connected. It is easy to see that the intersection of two polygons which are convex with respect to P is also convex with respect to P and can be computed in time proportional to the complexity of the polygons. Therefore, the set of points that both attract all points of T and are attracted by all points of T can be computed in quadratic time. Using a sweep line algorithm, the solution to the shortest beacon watchtower problem can be computed by finding the shortest distance between T and this set of points in $O(n^2 \log n)$ time.

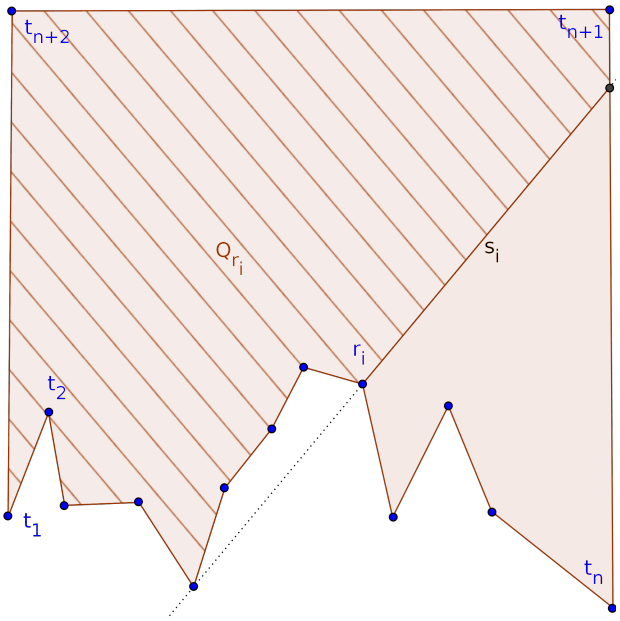

Figure 3: Constructing the terrain polygon T' from T. For space constraints, t_{n+1} and t_{n+2} are illustrated closer to T than they actually are. The shaded area is Q_{r_i} the portion of T' to the left of the steepest split edge of r_i as defined in lemma 1.

Here we show that the beacon kernel of a simple polygon and the inverse beacon kernel of a monotone polygon can be computed in $O(n \log n)$ time and present an $O(n \log n)$ time algorithm to compute the shortest beacon watchtower of T. We begin with some definitions. Let r be a reflex vertex incident to edges e_1 and e_2. Let H_1 and H_2 be half-planes perpendicular to e_1

and e_2 emanating from r which include the outside of P in a small neighbourhood of r. The *dead wedge* of r is defined as the intersection of H_1 and H_2 (Fig 4). Note that r introduces a split edge for points of P which reside inside its dead wedge.

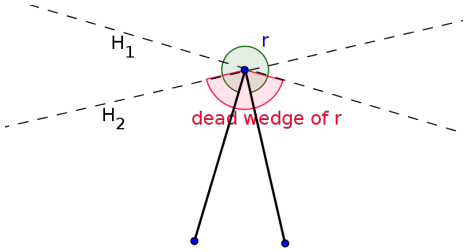

Figure 4: Dead wedge of a reflex vertex r. The lines H_1 and H_2 are perpendicular to e_1 and e_2, respectively.

According to [1] the beacon kernel of P is exactly the set of points in P which are outside of all dead wedges of reflex vertices of P.

Our solution to the shortest beacon watchtower problem consists of four steps; 1) computing the beacon kernel of T', 2) computing the inverse beacon kernel of T', 3) computing their intersection and 4) finding the shortest vertical distance between the resulting polygon and the terrain. Next we show how to perform each step in $O(n \log n)$ time.

2.1 Computing the beacon kernel of a simple polygon

Biro [1] presents an algorithm to compute the beacon kernel of a simple polygon which runs in $O(n^2)$ time. In this section we improve his result and present an algorithm to compute the beacon kernel of a simple polygon in $O(n \log n)$ time.

Let P be a simple polygon and let $r_1, r_2, ..., r_m$ be the set of reflex vertices of P. Let W_i denote the dead wedge of r_i and let C_i denote its complement. H_i^l and H_i^r are half-planes along edges of C_i excluding the exterior of P in a neighbourhood of r. Biro showed that the beacon kernel of P, $bker(P)$, is equal to $(\bigcap_{i=1}^{m} C_i) \cap P$ [1]. Therefore, we have

$$bker(P) = (\bigcap_{i=1}^{m} C_i) \cap P = (\bigcap_{i=1}^{m} (H_i^l \cup H_i^r)) \cap P =$$

$$((\bigcap_{i=1}^{m} H_i^l) \cap P) \cup ((\bigcap_{i=1}^{m} H_i^r) \cap P)$$

Let $H^l = (\bigcap_{i=1}^{m} H_i^l)$ and $H^r = (\bigcap_{i=1}^{m} H_i^r)$.

$$bker(P) = (H^l \cap P) \cup (H^r \cap P) = (H^l \cup H^r) \cap P$$

Note that H^r and H^l are convex sets and can be computed in $O(n \log n)$ time and their union, H, can be computed in $O(n)$ time. Using an algorithm of Chazelle and Edelsbrunner [4] the intersection of two simple polygons can be computed in $O(n \log n + k)$ time where k is the number of vertices of the resulting polygon. As an edge e of P can intersect at most 4 edges of a convex polygon (extreme case happens when e passes through two vertices of the convex polygon), e can intersect at most 8 edges of $H = H^l \cup H^r$. Therefore, the number of vertices of $P \cap H$ is $O(n)$ and $bker(P)$ can be computed in $O(n \log n)$ time using the algorithm in [4].

In our application, we compute the beacon kernel of a monotone polygon, where $bker(T') = (T' \cap H^l) \cup (T' \cap H^r)$. We claim that $bker(T')$ is also monotone with respect to the x-coordinate. According to [6] the intersection and union of two monotone polygons with respect to the same line is monotonic with respect to that line and can be computed in linear time. Therefore, $T' \cap H^l$ and $T' \cap H^r$ are both monotone polygons with respect to the x-coordinate and their intersection (the beacon kernel of T') is also monotone with respect to the x-coordinate.

2.2 Computing the inverse beacon kernel of a monotone polygon

According to [1] a point is in the inverse (beacon) kernel of a polygon P, if it belongs to the attraction region of each vertex of P. This immediately results in a quadratic time algorithm to compute the inverse kernel of a polygon [1]. Here, we present an algorithm that runs in $O(n \log n)$ time and computes the inverse kernel of a terrain polygon. Later we show how to extend this result and compute the inverse beacon kernel of an arbitrary monotone polygon in $O(n \log n)$ time.

We compute the inverse attraction kernel of T' in two runs. Let the right (left) attraction region of a vertex v be the set of points that are attracted by v and reside to the right (left) of v. Let the right (left) inverse kernel of T' be the set of points that are attracted by all vertices to their right (left). In each run we compute one of these kernels. By definition the intersection of the right inverse kernel and left inverse kernel is the inverse kernel of T'.

We use a new approach to compute the left inverse kernel which allows us to reduce the time complexity to $O(n \log n)$. Let r_i be a reflex vertex of T. Note

that r_i may introduce a split edge in the computation of right attraction regions of some vertices. A point to the right of or below any split edge of a right attraction region, cannot belong to the left inverse kernel. Among all split edges introduced by r_i, we are interested in the one which prevents the most points from residing in the left inverse kernel. Let Q_{r_i} be the portion of T' to the left of this split edge (Fig 3). We claim that the intersection of the polygons Q_{r_i} (for all reflex vertices, r_i) is the left inverse kernel of T'.

Lemma 1 *Let $r_1, r_2, ..., r_m$ be the set of reflex vertices of T' which introduce a split edge in the computation of the right attraction region of at least one vertex of T'. Let s_i be the steepest split edge among all the split edges introduced by r_i, i.e. the angle between s_i and an upward vertical ray emanating from r_i is the smallest among other split edges introduced by r_i. Let $Q = \bigcap_{i=1}^{m} Q_{r_i}$, where Q_{r_i} is the portion of T' to the left of s_i.*
Then Q is equal to the left inverse kernel of T'.

Proof. Let $q \notin Q$. There exists a reflex vertex r, where q is to the right of a split edge s introduced by r. Let s be the split edge of the vertex v. Clearly q is not in the right attraction region of v. Therefore, q is not in the left inverse kernel of T'.

Let $q \in Q$. For the sake of contradiction, let us assume there exists a vertex v, where q is not in the right attraction region of v. Therefore, q is to the right of the separation edge of v introduced by some reflex vertex r. By the construction of Q_r, q is not in Q_r which contradicts that $q \in Q$. □

Note that in Lemma 1 reflex vertices that do not introduce any split edges are ignored. Reflex vertex r does not introduce a split edge for any right attraction region, if no vertex to the left of r resides in the dead wedge of r.

Next we present an algorithm that computes the left inverse kernel of a terrain polygon. Later we show that the time complexity of the algorithm is $O(n \log n)$. The right inverse kernel is computed symmetrically. The algorithm starts by setting the *buddy* of some vertices of T. For each reflex vertex r, the first (rightmost) vertex v, which lies to the left of r and belongs to the dead wedge of r is found and buddy(v) is set to r. With this construction, v is the first vertex to the left of r for which r introduces a split edge. Note that later we may change the buddy of v to another reflex vertex located between v and r, this is done in order to cut through a separation edge rather than a split edge and discard more points that do not belong to the right attraction region of v.

Algorithm LeftInverseKernel
Input. Terrain T, terrain polygon T'.
Output. Left inverse kernel of T', i.e. the set of points $p \in T$ such that p is attracted by all vertices of T to its left.

1: Let $v_1, v_2, ..., v_n$ be the ordered set of vertices of T from right to left.
2: Let $r_1, r_2, ..., r_m$ be the ordered set of reflex vertices of T from right to left.
3: **for** $i = 1$ to m **do**
4: Let v_j be the left neighbour vertex of r_i in T.
5: **if** v_j is in the dead wedge of r_i **then**
6: Set buddy(v_j) to r_i.
7: **else**
8: Find l_i, the first intersection of the left edge of the dead wedge of r_i and T.
9: Let v_s be the first vertex of T located on or to the left of l_i.
10: **if** l_i exists **then**
11: Set buddy(v_s) to r_i.
12: **end if**{else r_i does not introduce a split edge.}
13: **end if**
14: **end for**
15: **for** $i = 2$ to n **do**
16: **if** buddy(v_i) is not set and buddy(v_{i-1}) is set **then**
17: buddy(v_i) = buddy(v_{i-1})
18: **else if** buddy(v_i) is located to the right of buddy(v_{i-1}) **then**
19: buddy(v_i) = buddy(v_{i-1}) {Details for the update of buddy appear in lemma 2.}
20: **end if**
21: **end for**
22: **for** $i = 1$ to m **do**
23: Find the vertex v such that buddy(v)=r_i and among all other vertices v_j such that buddy(v_j)=r_i, the line $\overline{vr_i}$ has the smallest angle with the vertical downward ray through r_i.
24: **if** such a vertex v exists **then**
25: Let ρ be the ray along the line $\overline{vr_i}$ emanating from r_i and let q be the first intersection of ρ with T'.
26: Add $\overline{r_i q}$ to the list of cut edges C.
27: **end if**{else ignore r_i}
28: **end for**
29: **return** Polygon R = CutOutBelow(C, T').

Note that by our construction, vertices with the same buddy are consecutive and vertices without a buddy are $\{v_1, v_2, ..., v_k\}$ $(1 \leq k \leq n)$, for the biggest k where these vertices attract all points of T' to their right. Therefore, we safely ignore these vertices in the construction of the left inverse kernel.

The procedure CutOutBelow(C, T'), returns a polygon by cutting T' through edges of C and discarding

the sub-polygon that is located below (or to the right of) a cut edge.

Lemma 2 *R is equal to $Q = \bigcap_{i=1}^{m} Q_{r_i}$ (of lemma 1).*

Proof. R is constructed by cutting through split edges of reflex vertices. Although these split edges may or may not be the steepest, we may assume that, $Q \subseteq R$.

We prove $R \subseteq Q$ by contradiction. Assume there exists a point $q \in R$, where $q \notin Q$. Therefore, there exists at least one reflex vertex r, where $q \notin Q_r$. Let r be the leftmost reflex vertex with this property. Let the steepest split edge of r (i.e. the cut edge of Q_r) s, be the split edge of v. If buddy$(v) = r$ then the algorithm chooses s as the cut edge of r and q cannot belong to R. So suppose buddy$(v) = r'$. By the construction of buddies, r' is a reflex vertex of T between v and r. Note that r' is above the line $\overline{vr}$, as otherwise (r, r') makes a steeper split edge than (r, v). Now consider $Q_{r'}$. It is constructed by cutting through the split edge of v or some steeper split edge (Fig 5). In both case q cannot belong to $Q_{r'}$, which contradicts the assumption that r is the leftmost reflex vertex with $q \notin Q_r$. □

The definition of Q implies that the left inverse kernel of T is monotone.

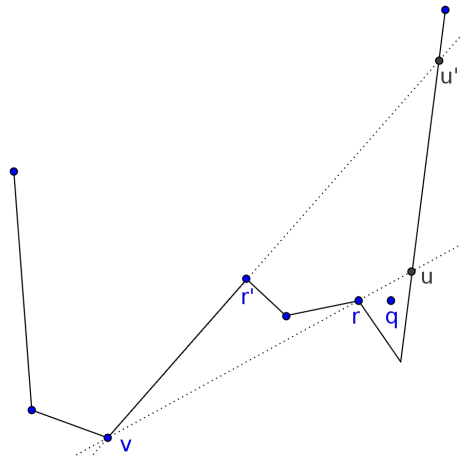

Figure 5: The split edge of v emanating from r, $\overline{ru}$, is completely to the right of the split edge of v emanating from r', $\overline{r'u'}$.

Lemma 3 *CutOutBelow(C, T') can be done in $O(n \log n)$ time.*

Proof. The input of CutOutBelow(C, T') is a set of edges and a monotone polygon. The output is the polygon resulting from cutting T' through these edges and discarding the sub-polygon below each edge. Let $c_1, c_2, ..., c_t$ be the set of edges in C. Let H_{c_i}, R_{c_i} and L_{c_i} be the half-plane above the line containing c_i, the half-plane to the right of the vertical line passing through the right endpoint of c_i and the half-plane to the left of the vertical line passing through the left endpoint of c_i, respectively (Fig 6). As T' is monotone, cutting though c_i and discarding the portion below c_i is equal to taking the intersection of T' and $(H_{c_i} \cup R_{c_i} \cup L_{c_i})$. Therefore, CutOutBelow$(C, T')$ results in Q, where

$$Q = (\bigcap_{i=1}^{t}(H_{c_i} \cup R_{c_i} \cup L_{c_i})) \cap T' =$$

$$((\bigcap_{i=1}^{t} H_{c_i}) \cap T') \cup ((\bigcap_{i=1}^{t} R_{c_i}) \cap T') \cup ((\bigcap_{i=1}^{t} L_{c_i}) \cap T'))$$

Let $H = (\bigcap_{i=1}^{t} H_{c_i})$, $R = (\bigcap_{i=1}^{t} R_{c_i})$ and $L = (\bigcap_{i=1}^{t} H_{c_i})$. H, R and L can be computed in $O(n \log n)$ time and are convex sets. The intersection of T' with any of these convex sets is a monotone polygon and can be computed in linear time. As the union of two monotone polygons with respect to the same line results in a monotone polygon with respect to to that line, Q can be computed in $O(n \log n)$ time. □

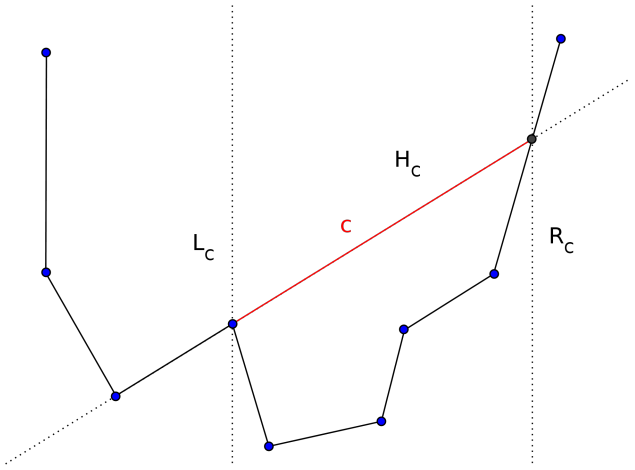

Figure 6: Three half-planes constructed from c in lemma 3.

Theorem 4 *The inverse kernel of the terrain polygon T' is a monotone polygon and can be computed in $O(n \log n)$ time.*

Proof. The inverse kernel of T' is the intersection of its right and left inverse kernels. As both of these polygons are monotone, their intersection is also monotone.

As T' is a monotone polygon a triangulation of T' can be computed in linear time and then T' can be preprocessed in linear time to answer ray shooting queries in $O(\log n)$ time [5]. We use ray shooting to find the intersection of the left dead wedge edge of each reflex vertex and the terrain in $O(\log n)$ time. After this, setting the buddies of vertices requires linear time. While processing a reflex vertex r_i and setting the buddies (steps 6 and 11 of the algorithm), we save a pointer to the first vertex v with buddy$(v) = r_i$. As vertices with the same buddy are consecutive, starting from v we can find the direction of the steepest split edge of r_i in time proportional to the number of vertices that have r_i as their buddy. Then the actual split edge is computed in $O(\log n)$ time using ray shooting. Each vertex has only one buddy and the overall time complexity of finding all the steepest split edges is $O(n \log n)$. By lemma 3 CutOutBelow(C, T') runs in $O(n \log n)$ time. Therefore, the total time complexity of the algorithm is $O(n \log n)$. Thus the left and right inverse kernels are computed in $O(n \log n)$ time. As both of them are monotone polygons, their intersection can be computed in linear time. □

The method presented for computing the inverse beacon kernel of a terrain polygon can be extended to compute the inverse beacon kernel of an arbitrary monotone polygon in $O(n \log n)$ time. Let M be a monotone polygon. Without loss of generality, assume that M is monotone with respect to the x-coordinate. Similar to lemma 1, we can define the inverse kernel of M. Let MU and ML be the upper and lower chains of M. From ML we construct a terrain polygon ML' similar to the construction of T' from T and compute the inverse beacon kernel of ML'. From MU we construct MU' which is also a terrain polygon and contains points below MU (the construction is similar to ML' but the resulting polygon is upside down compared to ML'). The previous method can be easily extended to compute the inverse beacon kernel of MU'.

There is only one modification in the computation of inverse kernels; when computing the cut edge of a reflex vertex r, the ray shooting is performed in M rather than ML' or MU'. This will discard points that are excluded from the inverse kernel due to r. We claim that the intersection of the resulting polygons is the inverse beacon kernel of M. We omit further details.

Theorem 5 *The inverse kernel of a monotone polygon can be computed in $O(n \log n)$ time.*

2.3 Computing the shortest beacon watchtower

To compute the shortest beacon watchtower we intersect the kernel and inverse kernel of the terrain polygon (in linear time). The result K is a monotone polygon. The shortest beacon watchtower is the minimum distance between K and T. So we are dealing with the problem of computing the minimum vertical distance between two monotone polygonal chains (T and the lower chain of K). This can be done with a simple vertical sweep line, where the vertices are the events. As both chains are piecewise linear, in each event we compute the vertical distance between the two chains and if necessary update the shortest distance. This process takes linear time.

3 Conclusion

We showed how to solve the shortest beacon watchtower problem in $O(n \log n)$ time and $O(n)$ space. In order to do this, we improved the time complexity of computing the beacon kernel of a simple polygon and the inverse kernel of a monotone polygon to $O(n \log n)$. Future work may consist of 1) computing the inverse kernel of a simple polygon in sub-quadratic time, 2) as some pair of points on the terrain can attract each other and do not need a watchtower, how can one compute the shortest watchtower in this case where both routing via the watchtower and direct routing is allowed, 3) extending the result to $3D$ by including techniques presented in [7].

References

[1] Michael Biro, Beacon-based Routing and Guarding. PhD Dissertation, Stony Brook University, 2013.

[2] Michael Biro and Jie Gao and Justin Iwerks and Irina Kostitsyna and Joseph S. B. Mitchell, Beacon-based routing and coverage. Proceedings of the 21st Fall Workshop on Computational Geometry, 2011.

[3] Michael Biro, Jie Gao, Justin Iwerks, Irina Kostitsyna, Joseph S. B. Mitchell. Combinatorics of beacon routing and coverage. Proceedings of the 25th Canadian Conference on Computational Geometry, 2013.

[4] Bernard Chazelle, and Herbert Edelsbrunner: An optimal algorithm for intersecting line segments in the plane. J. ACM 39(1):1-54 1992.

[5] Bernard Chazelle, Herbert Edelsbrunner, Michelangelo Grigni, Leonidas Guibas, John Hershberger, Micha Sharir, Jack Snoeyink. Ray shooting in polygons using geodesic triangulations. Algorithmica, 12(1):54-68, 1994.

[6] Darrell Shane, Monotone polygon intersection : geometry, computer applications, and computer graphics, Ft. Belvoir : Defense Technical Information Center, 1991.

[7] Binhai Zhu: Computing the shortest watchtower of a polyhedral terrain in O$(n \log n)$ time. Computational Geometry, 8(1):181-193, 1997.