Computing the Widest Empty Boomerang

Boaz Ben-Moshe* Binay Bhattacharya* Qiaosheng Shi*

Abstract

In this paper we consider the following obnoxious facility location problem: given a set S of n points in the plane, and two special points a and b, find the 1-corner polygonal chain (also known as boomerang) connecting a and b such that its minimum distance to S is maximized. In other words: find the widest empty polygonal chain of two edges having extremes anchored at a and b. We present a new $O(n \log n)$ algorithm which improves the previous $O(n^2)$ result [3].

1 Introduction

Facility location problems have been extensively studied by researchers from operations research and computational geometry. The classical p-center problem is to determine a set of facilities to service a set of clients, which minimizes the maximum distance between a client and the closest facility to which it is assigned.

In this paper we consider a variant of the facility location problem where the clients are represented by a set of points in the plane and we are required to locate one facility (an anchored 1-corner polygonal chain) so that the smallest distance from the points to the facility is maximized. Such problem is known as the obnoxious (undesirable) facility location problem.

Possible applications such as transporting hazardous material can be modeled as an obnoxious facility location problem. Other applications may involve 'robot-motion', for instance the problem of computing widest path which avoids collisions is an important tasks in robotics (e.g. [2]). In this sense the solution to the suggested problem gives a path with maximal clearance.

Several optimization problems which involve finding a 1-corner chain using a minimax criterion, have been posed by Glozman, Kedem and Shpitalnik [4]. Díaz-Báñez and Hurtado [3] suggested another version of 1-corner chain optimization problem, and gave an $O(n^2)$ algorithm for it. In this paper we consider the last problem (1-corner obnoxious chain optimization) and suggest an improved $O(n \log(n))$ algorithm for it.

The outline of the paper is as follows. In Section 2 we study the configuration cases that characterize the optimal solutions. A linear time algorithm is given to solve the decision version of the problem: Given r, does there exist any 1-corner chain from a to b so that the minimum distance of the points of S to the chain is at least r. We are assuming that the sorted lists of points of S around a and b are known. In Section 3 we present the main algorithm for computing the set (R) of all the *critical radii* which can be candidates for an optimal radius. It is shown that R is of linear size and can be computed in $O(n \log n)$ time. Once R is known, the optimal minmax solution can easily be found in $O(n \log n)$ time by applying the decision problem logarithmic number of times.

*School of Computing Science, Simon Fraser University, Burnaby BC, Canada V5A 1S6 {benmoshe,binay,qshi1}@cs.sfu.ca
*Research supported by NSERC & MITACS.

2 Preliminaries

In this paper we assume that a is to the left of b and $\overline{ab}$ is a horizontal. Moreover we will only consider the location of the boomerang above $\overline{ab}$ only (all other input-cases can be transformed to the above one using a linear rotation and flip operations). It is clear that the solution to the 1-corner polygonal chain problem might not be unique. Our objective in this paper is to find one optimal configuration. Observe that the following is an equivalent definition of the problem: given a set S of points, let S_r denote the set of (open) circles of radius r - centered at each point of S. Find the largest value r for which there exists a point t that are visible from both a and b, i.e. there exists a two-segments polygonal chain (from a to b) which does not penetrate any circle. More formally, we want to find the maximum value r^* for which $vis(a, S_{r^*}) \cap vis(b, S_{r^*})$ is not empty, where $vis(a, S_r)$ is the set of all points (in $\mathcal{R}^2$) which can be seen from a.
Given a 1-corner chain C, the locus of points at a distance r from C is called a boomerang centered at C and radius r. The boomerang is anchored if C is anchored. Our problem is to find an empty boomerang with the largest radius.

2.1 Computing the visibility picture from a

Given a radius r we would like to order around a all the circles in S_r which are at least partly visible from a. Let $V(a, S_r)$ be the set of circles that can be seen from a. Without any loss of generality, suppose a is the origin of the system. The *blocking angle* of a circle from a is defined to be the angle that the circle subtends at a. The blocking angle of a circle c_i of radius r with the center at p_i is represented by $[lt_i, rt_i]$ where lt_i and rt_i are the polar angles of the left and the right tangent points respectively of the circle c_i, Since all the circles have the same radii, only one single contiguous part (arc) of each circle (in $V(a, S_r)$) is visible from a. Therefore the circles in $V(a, S_r)$ can be ordered by their blocking angle as seen from a. Moreover, it is clear that if $r_1 < r_2$, then $V(a, S_{r_1}) \supset V(a, S_{r_2})$. Therefore, each ordered set $V(a, S_r)$, is an ordered subset of $V(a, S_0)$ which is simply the original points of S sorted around a.

Lemma 1 *Given a radius r and a set S of all points sorted around a, one can compute $V(a, S_r)$ in linear time.*

Proof. It is assumed that the points in S are sorted in increasing polar angle with the origin at a. Without any loss of generality we assume that $< p_1, p_2, \ldots, p_n >$ is the sequence of points in increasing polar angle. The algorithm can be described as follows:

```
1. $ST \leftarrow$ new stack, $max \leftarrow 0$.
2. Visit each point $p_i \in S$ in increasing polar angle.
(Let $c_i$ be the circle of radius $r$, centered at $p_i$)
3. Compute the blocking angle $[lt_i, rt_i]$ of $c_i$.
4. if($lt_i > max$) {Otherwise, $c_i$ is invisible from $a$. }
4.1 $max \leftarrow lt_i$.
4.2 Pop from $ST$ all ranges blocked by $[lt_i, rt_i]$.
4.3 Update the range of $c_j$ (top of $ST$) effected by $c_i$.
4.4 Update the range $[lt_i, rt_i]$ of $c_i$ effected by $c_j$.
4.5 Push the updated range onto $ST$.
```

The correctness of the algorithm follows easily from the following invariant that was maintained throughout.

Invariant 1 *(i) A circle which was removed from the stack is invisible (blocked by the circles in ST).*
(ii) The top circle in ST is visible (it is not totally blocked by the other circles in ST).
(iii) ST is an angle-ordered set, (by construction we only insert maximal circles to ST and the ranges are not overlapping – again by construction).

The running time of the algorithm is clearly linear once the points of S_i are already sorted around a. $\square$

Lemma 2 *Linear-time decision algorithm*
Given a problem instance (i.e. S, a, b and a radius r), one can check in $O(|S|)$ time whether there is an 1-corner polygonal chain (from a to b) such that its minimum distance all the points of S is at least r.

Proof. We assume that two radially sorted instances of S around a and b were pre-computed. We can then compute $V(a, S_r)$ and $V(b, S_r)$ (lemma 1) in linear time. The angular order of these arrays (each of linear size) allow us to sweep them (merge-like), one in counterclockwise order around a and the other in clockwise order around b. This way we can test in $O(n)$ time whether $vis(a, S_{r^*}) \cap vis(b, S_{r^*})$ is none-empty. Therefore, the total running time is clearly linear. $\square$

2.2 Critical points and critical radii

A boomerang of radius r is called *maximal* if its radius can not be increased. The points of S that determine a maximal empty boomerang (these points lie on the boundary of the boomerang) are called the *critical points*. The radius of the maximal empty boomerang is called the *critical radius* (see formal definition below).

Lemma 3 *[3]*
Let C^* be an optimal 1-corner polygonal chain and let r^* be the critical radius. The possible positions of the critical points for C^* falls in one of the three cases presented in figure 1. Case 1 has a similar configuration for the segment incident at b.

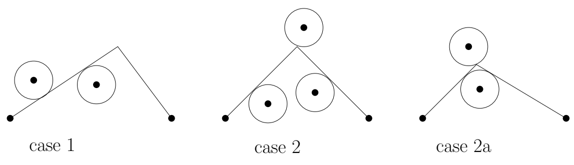

Figure 1: Cases of critical points

Denote by **critical radius** a positive value r' for which one of the following holds: (i) case 1: $V(a, S_{r'})$ or $V(b, S_{r'})$ are *topologically different*[1] from $V(a, S_{r'+\epsilon})$ or $V(b, S_{r'+\epsilon})$. (ii) case 2: $V(a, S_{r'}) \bigcap V(b, S_{r'})$ is *topologically different* from $V(a, S_{r'+\epsilon}) \bigcap V(b, S_{r'+\epsilon})$

Let C be an 1-corner chain $< a, t, b >$ for an arbitrary location t (above $\overline{ab}$). Consider a line l through $\overline{at}$. Let CH_l denote the convex hull of the points lying to the right of l. The necessary conditions for a point q to be a critical point of c are: (a) q must lie on CH_l, (b) q is visible from a, and (c) the orthogonal projection point of q on l lies above a. Let c_a and c'_a be the two points of CH_l such that any vertex in $[c_a, c'_a]$ is a potential critical point for a given l. $[c_a, c'_a]$ is the counterclockwise chain of CH_l from c_a to c'_a. Consider an edge e in $[c_a, c'_a]$. If the segment $\overline{at}$ is parallel to e, the critical radius must be the perpendicular distance between l and edge e. Let e and e' be two edges of $[c_a, c'_a]$ incident on a vertex p. If p is a critical vertex, the critical radius must lie between the critical radii of e and e'. The above discussions imply that: (i) Each vertex p of $[c_a, c'_a]$ determines a range of radii that contains the critical radius if p is a critical point of c. (ii) The range of radii determined by p is smaller than the range of radii determined by any vertex q if p is nearer to c_a than q. (iii) If p and q are adjacent, the intersection of the ranges of radii

[1]Looking at the relations between visible ranges within a set.

determined by them is the radius determined by the edge $\overline{pq}$.

Lemma 4 *All the critical radii ranges determined by the vertices of $[c_a, c'_a]$ monotonically increase as one traverses the chain from c_a to c'_a.*

3 Computing the critical-radii set

In this section an $O(n \log n)$ time algorithm is presented for computing the set R of all critical radii for the segment anchored at a. Only the case when the empty boomerang lies above $\overline{ab}$ is considered (the case when the boomerang is below $\overline{ab}$ is similar). We will separately compute the set of such radii for the cases mentioned in figure 1, namely; **case 1** and **case 2** (which also includes case 2a, a special sub-case of Case 2).
For both the cases a radial sweep ray (in counterclockwise order when centered at a and in clockwise order when centered at b) is used which stops at every point $p_i \in S$. The sweep starts with the ray pointing downwards. When we talk about the angle of a point p at a (b), we mean the counterclockwise (clockwise) angle p makes with the downward vertical ray at a (b). The convex-hull of all the points lying to the right (left) of the ray at a (b) is maintained using an incremental constant amortized time convex-hull algorithm [6] (the points are inserted according to their radial order around a (b)).

3.1 Case 1

The initial position of the sweep ray is pointed downwards. Let p_1 be the first point the sweep ray encounters. The points are relabeled such that the sweeping ray hits p_i immediately after p_{i-1} is encountered. A similar algorithm is applied for computing the critical radii for the segment anchored at b. Suppose the sweeping ray stops at the point $p_i \in S$ (Fig. 2 left). We compute $CH_{a\vec{p}_i}$ which is the convex hull of the points $\{p_1, p_2, \ldots, p_{i-1}\}$. Let $[c_{p_i}, c'_{p_i}]$ be the counterclockwise chain of $CH_{a\vec{p}_i}$ such that any point q on $[c_{p_i}, c'_{p_i}]$ satisfies the necessary conditions for q to be a critical point for the ray $a\vec{p}_i$. Clearly, $c_a = p_{i-1}$. c'_{p_i} can be easily computed in logarithmic time once $CH_{a\vec{p}_i}$ is known. Once the chain $[c_{p_i}, c'_{p_i}]$ is identified, the vertex q of the chain such that the points p_i and q determine a critical radius can be identified. Since the critical radii ranges of the vertices of the chain $[c_{p_i}, c'_{p_i}]$ are monotonic, the point q can easily be identified in $O(\log n)$ time. Once q is known, the critical radius, determined by p_i and q, is easy to compute. Note that no other point of $[c_{p_i}, c'_{p_i}]$ can determine a critical radius with p_i. Thus the total time required to compute all $O(n)$ critical-radii of case 1 type is $O(n \log(n))$.

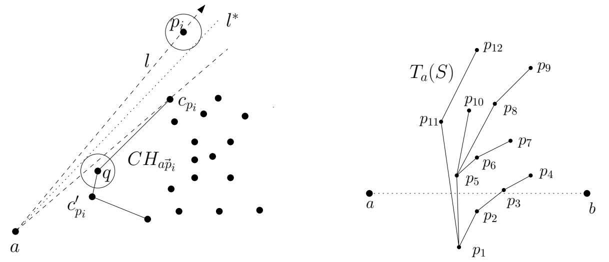

Figure 2: Left: binary search for the vertex (q) for which $dist(p_i, l^*) = dist(q, l^*) = dist(CH_{a\vec{p}_i}, l^*) = r^*$, where r^* is the critical radius. Right: the tree like structure $T_a(S)$ of the incremental (a-angle ordered) convex hull.

3.2 Case 2

In this case we are interested in computing the critical radius of type case 2, if any, for each point p_i. Three critical points that determine the critical radius are needed to be identified. One of them is p_i. When processing p_i, the convex-hull of the vertices that lie to the right of $a\vec{p}_i$ and to the left of $b\vec{p}_i$ is needed in order to determine the critical points. Let S_{p_i} denote the points whose convex hull is needed. We can compute the left visible convex chains of the points to the right of $a\vec{p}_i$ in an incremental fashion as follows. Let $< u_0 = p_{i-1}, u_1, \ldots, u_k >$ be the left visible convex chain in counterclockwise order when the sweep ray at a visits p_i. A simple linked list data structure is used to store the chain. When the sweep ray visits p_i, the left visible convex chain can be obtained by adding the tangent edge from p_i to the chain $< u_0 = p_{i-1}, u_1, \ldots, u_k >$ at some u_j. The updated left convex visible chain is $< p_i, u_j, u_{j+1}, \ldots, u_k >$. The amortized cost of computing the tangent edge $\overline{p_i u_j}$ is constant. We can access the left visible convex chain when the sweep line is next at p_{i+1} by just following the counterclockwise neighbor link starting from p_i. The union of all the left visible convex chains after processing $p_i, i = 1, 2, \ldots, n$ form a tree like structure $T_a(S)$ as shown in Figure 2 (right). This tree structure $T_a(S)$ can be preprocessed in linear time [1] so that the following operations can be performed on $T_a(S)$ (details are omitted due to space constraints):
1. Any node in the tree can be accessed from the root in logarithmic steps.
2. For an arbitrary node q, the left visible convex hull chain can be partitioned into $O(\log n)$ pieces.
3. Any node in the tree can be accessed from any other node in the tree in logarithmic time.
4. For an arbitrary node q, any $O(\log n)$ binary search query nodes on the path from q to the root can be identified in $O(\log n)$ time.
A similar tree structure $T_b(S)$ is constructed by sweeping a ray in clockwise order with the origin at b. This tree structure is also preprocessed for efficient queries.

Once we have preprocessed the tree structures $T_a(S)$ and $T_b(S)$, each data point p_i is treated in the following way: (Let S_{p_i} denote the set of points that are processed by both the sweep rays anchored at a and b before processing p_i.)
1. Compute the point of S_{p_i}, say c_a, which determines the largest angle at a.
2. Compute the point of S_{p_i}, say c_b, which determines the largest angle at b.
3. Determine the left visible convex chain of S_{p_i} starting from c_a.
4. Determine the right visible convex chain of S_{p_i} starting from c_b.
5. Determine c_a^* in the left visible chain and c_b^* in the right visible chain such that c_a^*, c_b^* and p_i together realize a critical radius of Case 2 type. Note that if c_a^* and c_b^* are the same, we have the case 2(a).

A simple way to implement the first two steps can be done by using a $2D$-range query data structure. We transform each point $p_i \in S$ to a point (α_i, β_i) where α_i is the angle of p_i at a β_i is the angle of p_i at b. Given a point (α_i, β_i), transformed from p_i, c_a (c_b) is the right most (top most) point to the left of α_i and below β_i. Similarly c_b is the top most point below β_i and to the left of α_i. Using the priority search tree [5] of S, both c_a and c_b can be computed in $O(\log n)$ time. The preprocessing time taken is $O(n \log n)$. The storage space requirement is linear. Steps 3 and 4 can be implemented in $O(\log n)$ time using spine tree decomposition [1]. The pairs c_a^* and c_b^* can be found by using two-stage binary search one on the left visible convex chain of S_{p_i} and the other on the right visible convex chain S_{p_i}. We first query a middle point q_a on the left visible convex chain and test if q_a satisfies the necessary conditions for a critical point. If not, the chain from q_a to the root in $T_a(S)$ can be ignored. Suppose q_a satisfies the necessary conditions for a critical point. We then determine the minimum critical-radius r_{q_a} allowed by q_a and search for the corresponding q_b whose critical radius range contains r_{q_a}. Last, we test the distance between the corresponding intersection (t) of the tangents and p_i (see figure 3). If this distance is bigger then r_{q_a}, the critical radius if exists must be larger than r_{q_a}. Therefore, the left visible convex chain $[c_a, q)$ and the right visible convex chain $[c_b, q_b)$ can be ignored. Otherwise (the critical radius is smaller than r_{q_a}) chains $(q_a, c_a']$ and $(q_b, c_b']$ can be eliminated. Once q_a, q_b and r_{q_a} are known, t can be computed easily in $O(1)$ time.

Observe that there could be at most one critical radius of type case 2 for each point p_i. The spine tree decomposition of $T_a(S)$ and $T_b(S)$ allows one to generate each query probe in amortized constant time. The total cost of generating probes on the left chain is $O(\log n)$. On the surface, it looks as if there will be $O(\log^2 n)$ query probes on the right chain. However, using the the decomposition tree, one can show that there are only $O(\log n)$ query probes for the right convex chain as well. Therefore,

Lemma 5 *The set of all* critical radii *of case 2 is of linear size, and can be computed in $O(n \log n)$.*

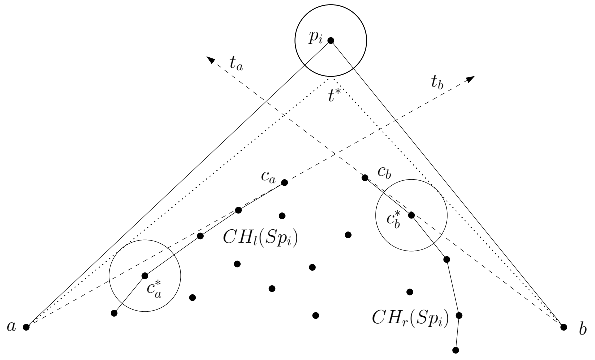

Figure 3: case 2: Two steps: first find the points c_a and c_b, then use a binary search to find pairs (c_a^*, c_b^*).

4 Determining the optimal radius

In this section we use the set (R) of all computed radii, to search (binary) the biggest radius $r^* \in R$ for which an 1-corner polygonal chain exists. Since each decision problem can be solved using the linear time decision algorithm (see lemma 2), therefore

Theorem 6 *The widest empty 1-corner polygonal chain problem can be solved in $O(n \log(n))$ time.*

Corollary 7 *Given an ellipse E centered at a and b, the suggested algorithm can be modified to find the widest 1-corner polygonal chain within E in $O(n \log(n))$ time. This improves the previous $O(n^2)$ result [3].*

References

[1] R. R. Benkoczi and B. K. Bhattacharya. Spine tree decomposition. Technical Report TR 1999-09, SFU-CS-School, 10 1999.

[2] S. W. Cheng. Widest empty L-shaped corridor. *Internat. J. Comput. Geom. Appl.*, 58(6):277–283, 1996.

[3] J. M. Díaz-Báñez and F. Hurtado. Computing obnoxious 1-corner polygonal chains. *Computers & Operations Research*, to appear, 2005.

[4] A. Glozman, K. Kedem, and G. Shpitalnik. Computing a double-ray center for a planar point set. *Internat. J. Comput. Geom. Appl.*, 9(2):109–124, 1999.

[5] E. McCreight. Priority search trees. *SIAM J. Computing*, 14, 1985.

[6] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.