Farthest Neighbor Voronoi Diagram in the Presence of Rectangular Obstacles

Boaz Ben-Moshe* Binay Bhattacharya* Qiaosheng Shi*

Abstract

We propose an implicit representation for the farthest Voronoi diagram of a set $\mathcal{P}$ of n points in the plane located outside a set $\mathcal{R}$ of m disjoint axes-parallel rectangular obstacles. The distances are measured according to the L_1 shortest path (geodesic) metric. In particular, we design a data structure of size $O(N^{1.5})$ in $O(N^{1.5}\log^2 N)$ time that supports $O(N^{0.5}\log N)$-time farthest point queries (where $N = m + n$). We avoid computing the more complicated farthest neighbor Voronoi diagram, whose combinatorial complexity is $\Theta(mn)$. This allows one to compute the diameter (and all farthest pairs) of $\mathcal{P}$ in $O(N^{1.5}\log^2 N)$ time. This improves the previous $O(mn\log N)$ bound [1].

1 Introduction

Let $\mathcal{R}$ be a set of m disjoint (axes-parallel) rectangles in the plane. We assume the rectangles in $\mathcal{R}$ are open, and refer to them as *obstacles*. Let $\mathcal{P}$ be a set of n points in *free space*, $\mathcal{F} = \mathcal{R}^2 - \cup\mathcal{R}$. In the following we will assume that m and n are of comparable sizes. In this paper the input is the pair $\mathcal{R}$, $\mathcal{P}$. The distance between two points $a, b \in \mathcal{F}$, denoted $d(a, b)$, is the L_1 geodesic (shortest obstacle-avoiding path) distance between them; i.e., $d(a, b)$ is the (Euclidean) length of a shortest path in $\mathcal{F}$, from a to b, that is comprised of axes-parallel line segments. For extensive background and numerous references, see [6].

In this paper a new result on farthest neighbor queries of $\mathcal{P}$ is presented. This is achieved by first constructing an implicit (compact) representation of the farthest Voronoi diagram. The problem we address is of constructing a data structure for farthest point queries in $\mathcal{P}$, which allows one to determine quickly the point of $\mathcal{P}$ that is farthest from a query point q. Note that the *nearest* neighbor query problem for L_1 geodesic distance in the plane is well understood, such Voronoi diagram has $O(N)$ size, $O(N\log N)$ constructing time, and $O(\log N)$ query time [7]. In contrast, the best results on L_1 *farthest* neighbor Voronoi diagrams requires $O(mn)$ space and $O(mn\log N)$ constructing time, to support $O(\log N)$ time queries [1]. In fact, the (explicit) combinatorial complexity of such Voronoi diagram is $\Omega(mn)$. For our problem we avoid explicitly computing the L_1 farthest neighbor Voronoi diagram. Instead, we use a query mechanism [4] which uses $O(N^{1.5}\log N)$ running time to compute an $O(N^{1.5})$ data structure and supports distance queries (shortest distance between any two corner points) in $O(N^{0.5})$ query time. Using this query mechanism we present a new (compact) representation of farthest Voronoi diagram (FVD) of size $O(N^{1.5})$ that supports $O(N^{0.5}\log N)$-time farthest point queries. The construction time of the representation is $O(N^{1.5}\log^2 N)$. We then use this compact Voronoi diagram to compute the diameter by finding first all the farthest pairs. The proposed algorithm has an $O(N^{1.5}\log^2 N)$ running time. This improves the best known algorithm which runs in $O(mn\log N)$ time [1].

2 Preliminaries

For points $a, b \in \mathcal{F}$, we let $path(a, b)$ denote any path between a and b of length $d(a, b)$. Assume that a lies to the left of (resp., below) b. An *x-monotone* (resp., *y-monotone*) path from a to b is a path in which all of the horizontal (resp., vertical) segments of the path are directed rightwards (resp., upwards). A path that is both x-monotone and y-monotone is *xy-monotone*. Clearly, if there exists an xy-monotone path between $a = (a_x, a_y)$ and $b = (b_x, b_y)$, then this path is a shortest path between a and b and its length is simply $|a_x - b_x| + |a_y - b_y|$.

Lemma 1 *[2] For any two points a and b in the free space, any shortest path between a and b is either x-monotone or y-monotone. Moreover, if both x-monotone and y-monotone paths exist, then there exists an xy-monotone path between a and b.*

Let $a, b \in \mathcal{F}$, with a to the left of b, and assume that there is no xy-monotone path between a and b. Denote *x-first*(a, b) to be a particular shortest path from a to b, obeying the following rule. In traveling from a to b we always prefer to go to the *right*, if possible (i.e., if by moving rightwards from the current location we do not penetrate into an obstacle or lose our ability to obtain a shortest path). We define *y-first*(a, b) similarly.

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

Let $q \in \mathcal{F}$ and let $\mathcal{P}_{left}$ (resp., $\mathcal{P}_{right}$) denote the subset of $\mathcal{P}$ consisting of the points that lie to the left (resp. right) of q, and for which there exists an x-monotone path ending at q. The *farthest point from q on the left* (resp., *right*) is the point in $\mathcal{P}_{left}$ (resp. $\mathcal{P}_{right}$) for which the shortest x-monotone path to q is the longest. Similarly, we define $\mathcal{P}_{below}$ (resp., $\mathcal{P}_{above}$) and the *farthest point from q from below (above).*

Lemma 2 *[1] The point in $\mathcal{P}$ that is the farthest from (query point) q is among the four points that are the farthest from q on the left, right, below, and above.*

Our goal is to construct a data structure for answering farthest neighbor queries: Given a query point q, find which of the points in $\mathcal{P}$ is the farthest from q. The above lemma allows us to construct four separate data structures, one for finding the farthest point from q on the left, one for finding the farthest point from q on the right, etc., and to answer a farthest point query by performing a query in each of the four data structures and selecting the farthest of the four candidates. We shall describe the data structure for finding the farthest point on the left; the other cases are analogous.

Theorem 3 *[1] Let a be a point in the free space, and let b be a point in the free space that lies in the (interior of the) right region of a. Let c be a point in the free space that lies to the right of b (not necessarily in the right region of a). If there exists an x-monotone path from b to c, then $d(a,c) > d(b,c)$ (see Figure 1 left).*

Let $\mathcal{X}^-$ denote the subset of $\mathcal{P}$ consisting of the points that do not lie in the right region of any other point in $\mathcal{P}$. $\mathcal{X}^-$ can be computed in $O(n \log n)$ time. The sets $\mathcal{X}^+$, $\mathcal{Y}^-$, and $\mathcal{Y}^+$ are defined similarly. Theorem 3 allows us, for example, to construct the substructure for finding the farthest neighbor on the left using the subset $\mathcal{X}^-$ rather than $\mathcal{P}$. According to Lemma 2 the farthest point (p) from q is either on the left, on the right, from below, or from above. Assume p is the farthest from q on the left, then p necessarily belongs to $\mathcal{X}^-$.

The sets $\mathcal{X}^-$ and $\mathcal{X}^+$ (alternatively, $\mathcal{Y}^-$ and $\mathcal{Y}^+$) may have some points in common. However, the intersection between $\mathcal{X}^*$ and $\mathcal{Y}^*$, where $^* \in \{-,+\}$, consists of a single point, assuming general position (since any such point must lie on a supporting line of slope ± 45 degrees).

Theorem 4 *[1] Let l be a vertical line, and let p_1, p_2 be two points in $\mathcal{X}^-$ to the left of l, such that $p_{1.y} > p_{2.y}$. If q_1 and q_2 are two points on l such that (i) there exists an x-monotone path from p_i to q_j, $i,j \in 1,2$, (ii) $d(p_1,q_1) > d(p_2,q_1)$, and (iii) $d(p_2,q_2) > d(p_1,q_2)$, then $q_{1.y} < q_{2.y}$.*

We can generalize Theorem 4 to any number of points in $\mathcal{X}^- = \{p_1, \ldots, p_k\}$. Let l be a vertical line to the right of $\mathcal{X}^-$. Then $\mathcal{X}^-$ divides l into ranges where each range is labeled by a point of $\mathcal{X}^-$ from which it is farthest (assuming there are x-monotone paths from each point in $\mathcal{X}^-$ to l). The following two observations are the direct outcomes of Theorem 4: i) any point $p \in \mathcal{X}^-$ can generate at most one single (connected) range on l – from which it is farthest to p, and ii) the y-order of the ranges (on l) is sorted in reverse y-order of their generating points in $\mathcal{X}^-$.

Let $q \in \mathcal{F}$ be an arbitrary point, let $\mathcal{X}^-(q)$ denote the set of all the points in $\mathcal{X}^-$ which are both to the left of q and have an x-monotone path to q. Let x_u^- (x_b^-) be the highest (lowest) point in $\mathcal{X}^-$.
Definition: $p \in \mathcal{X}^-$ is $FVD_{\mathcal{X}^-}(q)$, the restricted farthest point of q in $\mathcal{X}^-(q)$, if the following conditions hold: (i) $p \in \mathcal{X}^-(q)$, (ii) p is the farthest point from q among $\mathcal{X}^-(q)$, and (iii) $d(p,q) \geq max(d(q,x_u^-), d(q,x_b^-))$. In the following we assume that a vertical line passes through $\mathcal{F}$. If the line passes through a rectangle we will assume that it goes around the right side of the rectangle.

Lemma 5 *If p is $FVD_{\mathcal{X}^-}(q)$, for any vertical line l' to the right of q there exist a point $q' \in l'$ for which p is $FVD_{\mathcal{X}^-}(q')$*

Proof. In case q is not on the left boundary of an obstacle, let q' be the point q dragged rightwards till it hits an obstacle. Let dx be $d(q,q')$. Clearly $d(p,q') = d(p,q) + dx$ while for any other point $r \in \mathcal{X}^-$, $d(r,q') \leq d(r,q) + dx$. Therefore p is $FVD_{\mathcal{X}^-}(q')$.
If q is on the left boundary of an obstacle g, let q' be the farthest point from p on the right boundary of g. Let u (b) be the upper (lower) right corner of g (see Figure 1 middle). Let $r \neq p$ be a point in $\mathcal{X}^-$, the following two cases should be considered:
case 1: $r \in \mathcal{X}^-(q)$. Observe that if r is above p, $d(p,u) > d(r,q)$ (by Theorem 4). Therefore, $d(r,q') = d(r,u) + d(u,q') < d(p,u) + d(u,q') = d(p,q')$ (because q' is the farthest point from p on g). Hence $d(p,q') > d(r,q')$. A similar proof holds when r is below p.
case 2: $r \notin \mathcal{X}^-(q)$. Let dx and dy denote x and y coordinates difference between the points q and q' respectively. Observe the following: (i) $d(p,q') \geq d(p,q) + dx + dy$ while for every $r \notin \mathcal{X}^-(q)$ $d(r,q') \leq d(r,q) + dx + dy$, (ii) for every $r \notin \mathcal{X}^-(q)$ and $r \in \mathcal{X}^-(q')$, the path from r to q' is xy-monotone and therefore $d(r,q') \leq max(d(x_u^-,q'), d(x_b^-,q')) \leq d(p,q')$. □

Theorem 6 *Each cell in the farthest Voronoi diagram of $\mathcal{P} \cup \mathcal{R}$ in L_1 metric is infinite.*

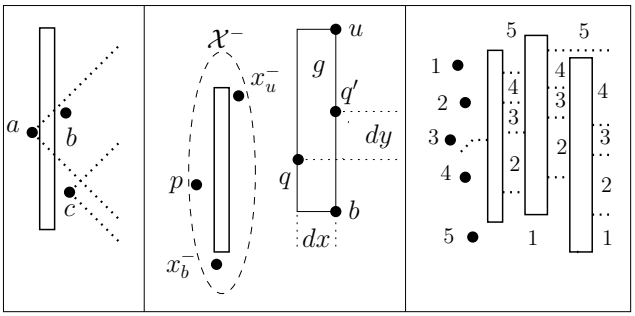

Figure 1: Left: b can not be the farthest point to the left ($b \notin X^-$). Middle: any vertical line (to the right of q) will cut the $FVD_{\mathcal{X}^-}$ cell of p (lemma 5). Right: a construction showing that the farthest neighbor Voronoi diagram can have complexity of $\Omega(mn)$ [1].

3 Farthest neighbor queries

In this section we describe a data structure for *'farthest point on the left'* queries in $\mathcal{X}^-$, denoted by $FVD_{X^-}(.)$. That is, for a query point q, find its farthest point among $\mathcal{X}^-(q)$[1]. Let $d_{\mathcal{X}^-}(p,q)$ denote the following function: $d_{\mathcal{X}^-}(p,q) = d(p,q)$: if $p \in \mathcal{X}^-(q)$, $d_{\mathcal{X}^-}(p,q) = 0$: if $p \notin \mathcal{X}^-(q)$.

3.1 Computing a cut in the FVD

We first develop a method which determines, for an arbitrary line $l(x)$, the cut of $l(x)$ with the $\mathcal{X}^-$ farthest Voronoi diagram ($FVD_{\mathcal{X}^-}$). It is assumed that the intersection of $l(x)$ with the horizontal decomposition of $\mathcal{F}$ is known. This is available in the form of a binary tree which, when traversed in inorder fashion, reports the intersection points on $l(x)$ in increasing y-order. We can use the persistent tree data structure [3] computed by sweeping the horizontal decomposition of $\mathcal{X}^- \cup R$ by a vertical line from left to right. This data structure allows the access of the binary tree of any cut of the decomposition by a vertical line in $O(\log n)$ time.
Given a set of points $\mathcal{X}^-$ and a query vertical line $l(x)$, we want to find the 'cut' that $l(x)$ performs on $FVD_{\mathcal{X}^-}$, more formally: find all $p \in \mathcal{X}^-$ such that there exists a point $q \in l(x)$ for which p is $FVD_{\mathcal{X}^-}(q)$. The algorithm uses the query mechanism developed by ElGindy and Mitra [4]. The query mechanism preprocesses the set of points $\mathcal{X}^-$ and the rectangles $\mathcal{R}$ in $O(N^{1.5}\log N)$ requiring $O(N^{1.5})$ space such that distance query between any corner point of the rectangles and a data point of $\mathcal{X}^-$ can be answered in $O(N^{0.5})$ time.

Let $Z(x)$ denote the set of intersection points of $l(x)$ with the horizontal decomposition of $\mathcal{X}^- \cup \mathcal{R}$. Let $Z_j(x)$ denote the j^{th} intersection point on $l(x)$ from the bottom. The algorithm **FVDsnapshot**(A,k,k'), described below, computes the intersection of the $FVD_{\mathcal{X}^-}$ of the points in A with $l(x)$, between the intersection points $Z_k(x)$ and $Z_{k'}(x)$. Suppose $A = \{x_1, x_2, \ldots, x_t\} \subseteq \mathcal{X}^-$.
Algorithm FVDsnapshot(A,k,k')
step 0: If $|A| = 1$, Return.
step 1: If $Z_k(x)$ and $Z_{k'}(x)$ are consecutive on $l(x)$, compute the intersection of FVD of A on the segment $[Z_k(x), Z_{k'}(x)]$ on $l(x)$. Return.
step 2: $j_m = \lceil (k+k')/2 \rceil$, find[2] $x_{j_m} = FVD_{\mathcal{X}^-}(Z_{j_m}(x))$.
step 3: Compute *FVDsnapshot*$(\{x_1, x_2, \ldots, x_{j_m}\}, j_m, k')$
step 4: Compute *FVDsnapshot*$(\{x_{j_m}, \ldots, x_t\}, k, j_m)$

In order to refine the $FVD_{\mathcal{X}^-}$ cut between each two consecutive points $Z_k(x), Z_{k'}(x) \in l(x)$, we only need to query distances ($d_{\mathcal{X}^-}$) between $Z_k(x)$ and $Z_{k'}(x)$ to only the points of A which falls in the corresponding $\mathcal{X}^-$ range of $[FVD_{\mathcal{X}^-}(Z_k(x)), FVD_{\mathcal{X}^-}(Z_{k'}(x))]$. The correctness of the above divide and conquer algorithm follows from Theorem 4 and Lemma 5. Therefore,

Lemma 7 *The complexity of the call FVDsnapshot*$(\mathcal{X}^-, 1, |Z(x)|)$ *is* $O(|\mathcal{X}^-|N^{0.5}\log N)$.

Notice that besides the per query cost $O(N^{0.5})$, m, the number of obstacles is involved in the term $\log N$ only.

3.2 Computing all cells of the FVD

In this section we address the problem of computing the left most x-coordinate of each Voronoi cell of $FVD_{\mathcal{X}^-}$. As shown in Lemma 5 each cell of such FVD is infinite, and new cells can only start after an *obstacle event*. Hence we only need to attach each cell to the left most obstacle it is starting from.If more than one Voronoi cell is attached after an obstacle event, the list of cells attached to the rectangle are kept in sorted y-order. Therefore, we are interested in computing all the cuts in the FVD determined by the right sides of the obstacles from where at least one new Voronoi cell has started.

Let $\mathcal{X}_e^- \subseteq \mathcal{X}^-$ be the set of points which actually determine a Voronoi cell in $FVD_{\mathcal{X}^-}$. $\mathcal{X}_e^-$ can be easily determined by calling *FVDsnapshot*$(\mathcal{X}^-, 1, |Z(x)|)$ where the vertical line $l(x)$ lies to the right of $\mathcal{R}$ and $\mathcal{P}$. We now look for a vertical line $l(x_j)$ such that the number of Voronoi cells intersecting $l(x_j)$ is at most $|\mathcal{X}_e^-|/2$ and the number of Voronoi cells intersecting the vertical line $l(x_{j'})$, determined by the next obstacle event, is greater than $|\mathcal{X}_e^-|/2$. Using the algorithm *FVDsnapshot* in a binary search mode, $l(x_j)$ and $l(x_{j'})$ can be found using only $O(\log N)$ probes.

Let $< VP(q_0), VP(q_1), \ldots, VP(q_t) >$ be the sequence of Voronoi cells intersecting $l(x_j)$ in increasing y-order. Let $< VP(p_{i_0}), VP(p_{i_1}), \ldots, VP(p_{i_{t'}}) >$ be the sequence of

[1]Note: this data structure might return 'null' in cases it is clear that the farthest point of q is not from $\mathcal{X}^-(q)$.

[2]If $FVD_{\mathcal{X}^-}(Z_{j_m}(x))$ is undefined, find the point q' (which is the Y-projection of $Z_{j_m}(x)$ on $l(x')$ - the next event line to the right of $l(x)$). If q' falls inside an obstacle, or $FVD_{\mathcal{X}^-}(q')$ is still undefined, mark Z_{j_m} to be 'null' and set x_{j_m} to be the extrema point (x_b^- or x_u^-) it is farthest from. Else set x_{j_m} to be $FVD_{\mathcal{X}^-}(q')$.

Voronoi cells intersecting $l(x_{j'})$ in increasing y-order. According to Theorem 4, $\{q_0, q_1, \ldots, q_t\} \subset \{p_{i_0}, p_{i_1}, \ldots, p_{i_{t'}}\}$. Snapshots on $l(x_j)$ and $l(x_{j'})$ partition the problem of computing all the Voronoi cells into the following similar subproblems.

(a) Determine the FVD of $\{q_0, q_1, \ldots, q_t\}$ to the left of $l(x_j)$.

(b) Determine the FVD of $\{p_{i_s}, \ldots, p_{i_{s+1}}\}$, $s = \{1, 2, \ldots, t' - 1\}$ to the right of $l(x_{j'})$.

If p_{i_s} and $p_{i_{s+1}}$ are consecutive in $\mathcal{X}_e^-$ in y-order, no more partition is needed. This is the base case. In case (b) the points are partitioned, and the size of each set $\{p_{i_s}, \ldots, p_{i_{s+1}}\}$ is at most $|\mathcal{X}_e^-|/2 + 2$. Therefore,

Theorem 8 *All the relevant vertical snapshots of FVD of $\mathcal{X}^-$ can be computed in $O(nN^{0.5} \log^2 N)$ time. The storage space requirement is $O(N^{1.5})$.*

3.3 A compact representation of the FVD

In the previous section the starting point of each Voronoi cell has been determined. The starting point of each cell is attached to the right hand side of some rectangle. If more than one cell starts from the same rectangle, these cells are listed in increasing y-order. We are now interested in presenting the $FVD_{\mathcal{X}^-}$. Hence we need to construct a data structure that holds all the vertical *FVDsnapshots* of the farthest Voronoi cells of $\mathcal{X}^-$ (in there y-order). An explicit representation of all these *FVDsnapshots* has an $O(mn)$ size (see Figure 1 right). Therefore, we use a compact implicit representation based on the persistent tree data structure [3]. This data structure allows the access of the binary tree of any 'cut' of the decomposition by a vertical line in $O(\log n)$ time. At first the persistent tree PT is empty, then we go over all the obstacles according to there x-order (left to right, as implied by there right edge x-coordinate). For each obstacle, the list of (new) cells that was attached to it, is inserted to PT.

Now, the persistent tree PT can support in $O(\log n)$ time the following query: given a vertical line $l(x)$ and an integer k, find the farthest Voronoi cell in the $FVDsnapshots(l(x))$ which is in the k^{th} position according to the cells y-order (if undefined return 'null').

Corollary 9 *A persistent tree of size $O(n)$ (which stores the left starting point of each farthest Voronoi cell of $\mathcal{X}_e^-$), can be used to implicitly represent the furthest Voronoi diagram.*

Theorem 10 *The implicit FVD of $\mathcal{X}^-$ can be computed in $O(N^{1.5} \log^2 N)$ time. The storage space requirement is $O(N^{1.5})$.*

3.4 Query mechanism for the FVD

Given a query point q we want to find the farthest point to its left (the farthest point of $\mathcal{X}^-$ from which there is an x-monotone path), More formally, we want to find if there is a point $p \in \mathcal{X}^-$ which is $FVD_{\mathcal{X}^-}(q)$. Observe that if there is no such p, the furthest point of q, is from $\mathcal{X}^+ \cup \mathcal{Y}^- \cup \mathcal{Y}^+$.

FVD query algorithm: find the first obstacle r_q to the left[3] of q. This obstacle can be found in $O(\log m)$ time using a planar point location data structure [5]. Next, use the persistent tree PT, to find a 'middle' farthest cell (p_{mid}) of the *FVDsnapshot* associated with the right edge of r_q. Let u (b) be the up (low) right corner of r_q. Compute $d(u, p_{mid})$ and $d(b, p_{mid})$ using ElGindy and Mitra [4] query mechanism. Let $d_u = d(p_{mid}, u) + d(u, q)$, $d_b = d(c_{mid}, b) + d(b, q)$. If $d_u < d_b$, search PT 'below' p_{mid}, else search PT 'above' p_{mid}.

Using $\log(n)$ iterations of binary search, $FVD_{\mathcal{X}^-}(q)$ can be found in total time of $O(N^{0.5} \log n)$. Similarly we can find the farthest points of q in $\mathcal{X}^+$, $\mathcal{Y}^-$ and $\mathcal{Y}^+$. Let $p' \in \mathcal{P}$ be the maximum distance over the above four farthest points. Let $p'' \in \mathcal{P}$ be the farthest point of q in the L_1 free space (ignoring all obstacles). The farthest point of q is the maximum over p' and p''.

Theorem 11 *One can construct a data structure of size $O(N^{1.5})$ in $O(N^{1.5} \log^2 N)$ time, such that the farthest point of a query point q can be found in $O(N^{0.5} \log n)$ time.*

Using the implicit FVD data structure one can perform a farthest neighbor query from each of the points in $\mathcal{P}$, thereby obtaining all farthest neighbors pairs. This can be done in $O(N^{0.5} \log n)$ time.

Corollary 12 *Once the implicit FVD of $\mathcal{P}$ is known, one can compute all farthest neighbors pairs and hence the diameter in $O(nN^{0.5} \log n)$ time*[4].

References

[1] B. BenMoshe, M. J. Katz, and J. S. B. Mitchell. Farthest neighbors and center points in the presence of rectangular obstacles. In *Proc. 17th Annu. ACM Sympos. Comput. Geom.*, pages 164–171, 2001.

[2] J. Choi and Chee-Keng Yap. Monotonicity of rectilinear geodesics in d-space. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 339–348, 1996.

[3] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. Making data structures persistent. *J. Comput. Syst. Sci.*, 38(1):86–124, 1989.

[4] H. A. ElGindy and P. Mitra. Orthogonal shortest route queries among axis parallel rectangular obstacles. *Int. J. Comput. Geometry Appl.*, 4(1):3–24, 1994.

[5] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.

[6] K. Klamroth. *Single Facility Location Problems with Barriers*. Habilitation thesis, Dept. of Mathematical Sciences, University of Kaiserslautern, Germany, 2000.

[7] J. S. B. Mitchell. L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica*, 8:55–88, 1992.

[3] If there is no such obstacle, the farthest point of q will be found by the other direction queries or the free space query.

[4] in practice we expect the bounds to be much smaller, since after spending $O(n \log n)$ time in computing the sets $\mathcal{X}^-, \mathcal{X}^+, \mathcal{Y}^-, \mathcal{Y}^+$, n in the above bounds can be replaced by $|\mathcal{X}^-| + |\mathcal{X}^+| + |\mathcal{Y}^-| + |\mathcal{Y}^+|$, which is usually much smaller than n, more like $\sqrt{n}$.