Farthest-Point Queries with Geometric and Combinatorial Constraints

Ovidiu Daescu [a], Ningfang Mi [a], Chan-Su Shin [b], and Alexander Wolff [c]

[a] *Department of Computer Science, University of Texas at Dallas, Richardson, TX 75083, USA*
[b] *School of Electronics and Information Engineering, Hankuk University of Foreign Studies, Korea*
[c] *Faculty of Computer Science, Karlsruhe University, Germany. WWW: i11www.ilkd.uka.de/algo/people/awolff*

1. Introduction

In this paper we discuss farthest-point problems, in which a sequence $S = (p_1, p_2, \dots, p_n)$ of n points in the plane is given in advance and can be preprocessed to answer various queries efficiently. We first consider the general setting where query points can be arbitrary, then we investigate a special setting where each point in S is queried exactly once.

To describe our problems, we use the following notation. Given two points $p \neq q$, let pq denote the line through the points p and q, let $\overline{pq}$ denote the line segment joining p and q, and let $|pq|$ be the Euclidean distance of p and q. We will use ε to denote a fixed arbitrarily small positive number.

FarthestPointAboveLine (FPAL): Given a pair (q, l_q), where q is a point and l_q is a line through q, decide whether there is a point in S above l_q, and if yes report the one farthest from q.

Our solution to this problem is a data structure based on [6] that takes $O(n \log n)$ space, $O(n^{1+\varepsilon})$ preprocessing time and $O(n^{1/2+\varepsilon})$ query time. We can do better if S is in convex position:

FPALinConvexPolygon (FPALCP): Given a convex n-gon C and a pair (q, l_q), where q is a point and l_q is a line through q, decide whether there is a point in C above l_q, and if yes report the one farthest from q.

Our solution takes $O(n \log n)$ space, $O(n \log^2 n)$ preprocessing time and $O(\log^2 n)$ query time.

FarthestPointSameSide (FPSS): Given a triplet $(q, \mathcal{L}_q, \Delta)$, where q is a point and $\mathcal{L}_q$ is a line through q such that all points in S are within distance Δ from $\mathcal{L}_q$, decide whether there is a point $p \in S$ such that (i) $|qp| > \Delta$, and (ii) p is above the line l_q orthogonal to $\mathcal{L}_q$ at q. If yes, report the point p farthest from q that fulfills (ii).

Our data structure for this problem has the same time and space bounds as that for FPALCP.

FarthestIndexedPointOtherSide (FIPOS): Given a triplet (i, j, Δ), such that $1 \leq i < j \leq n$ and all points $p_k \in S$ with $i < k < j$ are within distance Δ from $p_i p_j$, decide whether there is a point p_k with $i < k < j$ such that (i) $|p_i p_k| > \Delta$, and (ii) p_k and p_j lie on different sides of the line that goes through p_i and is orthogonal to $p_i p_j$. If yes, report the point p_k farthest from p_i that fulfills (ii).

Our time and space bounds for this problem are by a log-factor above those for FPALCP.

In the special setting where each point in S is queried exactly once we investigate the following problem. We assume the existence of a point $p \notin S$.

BatchedFarthestIndexedPointSameSide (BFIPSS): For each point $p_i \in S$ decide whether there is a point $p_f \in \{p_1, \dots, p_i\}$ that lies on the same side as p with respect to the perpendicular bisector of p and p_i. If yes report the point p_f farthest from p that has the above property.

Our algorithm for this problem runs in $O(n \log^2 n)$ time and uses $O(n \log n)$ space.

The problems we study are related to the nearest-point query problem, to the all-pairs farthest and closest neighbors problem, and to the closest-point-to-line query problem. Although these problems are well understood, we are not aware of work on the problems we consider.

Applications of our data structures are polygonal chain approximation [2], approximate solutions for the one-cylinder problem [1], and geometric spanning trees [5].

Email addresses: daescu@utdallas.edu (Ovidiu Daescu), nxm024100@utdallas.edu (Ningfang Mi), cssin@hufs.ac.kr (Chan-Su Shin).

2. Farthest-Point Queries

We first tackle FPALCP, since it is part of the solutions for the problems FPSS and FIPOS.

In the preprocessing phase, we construct a balanced binary tree T in $O(n \log^2 n)$ time as follows. The vertices of the convex polygon C, in counter-clockwise order from the rightmost vertex, are associated with the leaves of T. At each internal node u, we compute and store the convex hull and the farthest-point Voronoi diagram V_u of the leaf descendants of u from the information available at the children of u. We then preprocess V_u for planar point-location queries, which takes a total of $O(n \log n)$ time for each level of T. Thus, the overall computation of T takes $O(n \log^2 n)$ time.

Given a query pair (q, l_q), we find in $O(\log n)$ time the intersection points of l_q with the boundary ∂C of C by binary search. We assume that l_q has non-empty intersection with the interior of C (the four other cases are trivial). The sought point is either one of the two intersection points of l_q with ∂C or a vertex of C that is above l_q. Without loss of generality, we assume that no vertex of C lies on l_q and that l_q has positive slope.

We query T as follows. We select the two endpoints of the segments intersected by l_q that are above l_q. Let s be the first and t the second endpoint in counter-clockwise order on ∂C. We walk in T from s to t and collect a set $\mathcal{V}$ of $O(\log n)$ farthest-point Voronoi diagrams in two phases. In the ascending phase we go upwards from s. Whenever we get to a node u from its *left* child, we add to $\mathcal{V}$ the Voronoi diagram stored at the right child of u. In the descending phase we go down towards t. Whenever we go to the *right* child of a node u, we add to $\mathcal{V}$ the Voronoi diagram stored at the left child of u. Clearly, all points associated with these Voronoi diagrams are above l_q and thus the sought vertex is either s, t or one of these points. We locate q in $O(\log n)$ time in each Voronoi diagram in $\mathcal{V}$ and keep track of the point farthest from q. Thus we can answer the query in $O(\log^2 n)$ time.

Theorem 1 *There is a data structure for FPALCP that takes $O(n \log n)$ space, $O(n \log^2 n)$ preprocessing time and $O(\log^2 n)$ query time.*

One can answer queries for FPSS using the same approach as for FPAL: construct a partition tree based on a fine simplicial partition in $O(n^{1+\varepsilon})$ time [6] and enhance it with a second-level data structure consisting of the farthest-point Voronoi diagram (of the points at each internal node) preprocessed for planar point location. This data structure can be used to obtain a collection of disjoint subsets of points representing all points in S which are above the line l_q, and then query the farthest-point Voronoi diagrams for the points in each subset in order to answer the query. Note that the point farthest from the query point q may lie *inside* the convex hull C of S, so it seems our solution of FPALCP cannot be applied here. The following lemma, however, does give us a way to use the FPALCP data structures to solve FPSS.

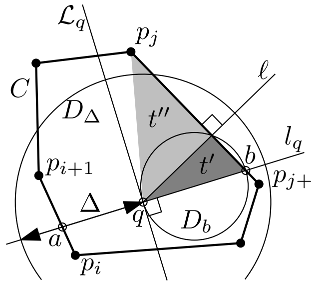

Fig. 1. The point p farthest from q must be a vertex of the convex hull C of S if $|qp| > \Delta$.

Lemma 2 *Given $S \subset \mathbb{R}^2$ and a triplet $(q, \mathcal{L}_q, \Delta)$, where q is a point and $\mathcal{L}_q$ is a line through q such that all points in S are within distance Δ from $\mathcal{L}_q$, if there is a point $p \in S$ such that (i) $|qp| > \Delta$, and (ii) p is above the line l_q orthogonal to $\mathcal{L}_q$ at q, then the point in S farthest from q is a vertex of the convex hull C of S.*

PROOF. Wlog. we assume that l_q intersects ∂C in two line segments $\overline{p_i p_{i+1}}$ and $\overline{p_j p_{j+1}}$ with $1 \leq i < j < n$ and that $p_{i+1}, \ldots, p_j$ all lie above l_q. Let a and b be the intersection points. Clearly for each triangle qp_kp_{k+1} above l_q either p_k or p_{k+1} is farthest from q. We now consider the triangle $t = qp_jb$, the triangle qap_{i+1} is analogous.

Let ℓ be the line through q that is orthogonal to p_jp_{j+1}. Consider the right-angled triangle t' (shaded dark in Figure 1) that is defined by ℓ, p_jp_{j+1} and qb. Due to Thales' theorem t' is contained in the disk D_b whose diameter is $\overline{qb}$. Since $|qb| \leq \Delta$, D_b (and thus t') is contained in the radius-Δ disk D_Δ centered at q.

Now if $q_j \in t'$ then $t \subseteq t' \subseteq D_\Delta$. Otherwise ℓ splits t into t' and another right-angled triangle t'' (shaded lightly in Figure 1) that is defined by qp_j, p_jp_{j+1} and ℓ. Since $\overline{qp_j}$ is the hypothenuse of t'',

p_j is farthest from q in t''. Thus the point farthest from q either lies in $D_\triangle$ or is a vertex of C. □

Theorem 3 *There is a data structure for FPSS that takes $O(n \log n)$ space, $O(n \log^2 n)$ preprocessing time and $O(\log^2 n)$ query time.*

To solve FIPOS, the indexed version of FPSS, in the preprocessing phase we construct an $O(n \log^2 n)$-size balanced binary tree T as follows. Each leaf of T is associated with a point in S such that the point $p_i \in S$ is stored at the i-th leaf of T. We go up the tree T and, at each internal node v, we compute and store the convex hull C_v of the leaf descendants of v. We also compute and store at v a secondary level data structure in the form of a balanced binary tree T_v of the farthest-point Voronoi diagrams on the vertices of C_v, enhanced with a planar point-location data structure, similar to the one used in solving FPSS. Then, the overall computation of T takes $O(n \log^3 n)$ time and requires $O(n \log^2 n)$ space. For a query pair (i, j), let π_{ij} be the path in T from p_i to p_j. We use this path to obtain a set $\mathcal{C}$ of $O(\log n)$ convex hulls whose union contains only the points p_k with $i < k < j$ as follows. In the ascending phase we add to $\mathcal{C}$ the convex hull stored at the right child of v if π_{ij} gets to v from its left child. In the descending phase we add to $\mathcal{C}$ the convex hull stored at the left child of v if π_{ij} goes from v to its right child. For each convex hull $C \in \mathcal{C}$, querying the secondary data structure reduces to FPSS. Let t be the number of vertices on C. Then we can determine in $O(\log^2 t)$ time whether there is a k, $i < k < j$, such that the point $p_k \in S$ associated with C satisfies the two FIPOS conditions. Since the size of the set $\mathcal{C}$ is $O(\log n)$, the overall query time is $O(\log^3 n)$.

Theorem 4 *There is a data structure for FIPOS that takes $O(n \log^2 n)$ space, $O(n \log^3 n)$ preprocessing time and $O(\log^3 n)$ query time.*

Building on our solutions of FPALCP, FPSS, and FIPOS, we extend a recent result in [3].

Theorem 5 *Given a polygonal chain $P = (p_1, p_2, \ldots, p_n)$ in the plane, the min-# problem with the tolerance zone criterion and L_2 distance metric can be solved in $O(F(m)\, n \log^3 n)$ time with $O(n \log^2 n)$ space, where $F(m)$ is the number of vertices of the path approximation graph reachable from p_1 with at most $m - 2$ edges and m is the number of vertices of an optimal approximating path.*

A version of BFIPSS without the index restriction has been considered in [5]. There the problem was to report for each point $p_i \in S$ a point farthest from the fixed point $p \notin S$ that lies on the same side as p with respect to the perpendicular bisector of p and p_i. In [5] the problem is reduced to the problem of finding for each $p_i \in S$ the first disk in a sequence of disks that does *not* contain p_i. This problem has been addressed in [2] under the name *off-line ball exclusion search (OLBES)*. The authors set up a tree data structure with a space requirement of $O(n \log n)$ and then query this structure with each point in S. This results in a total running time of $O(n \log n)$ for OLBES in dimension $d = 2$. For $d > 2$ the problem is solved differently in $O(n^{2-2/(\lfloor d/2 \rfloor)+1})$ time. In [5], a version of OLBES where all disks intersect a common point has been solved for $d = 2$ in $O(n \log n)$ time and $O(n)$ space by sweeping an arrangement of circular arcs. To solve BFIPSS in dimension 2 we set up a tree data structure similar to [4, proof of Lemma 2]. Here, however, we must solve a different OLBES problem in each query and thus need to modify our tree successively.

Theorem 6 *BFIPSS can be solved in $O(n \log^2 n)$ time and $O(n \log n)$ space for a sequence S of n points and a point $p \notin S$ in the plane.*

PROOF. Let $D_{n+1} = \emptyset$ and let $D_1, \ldots, D_n$ be the sequence of disks in order of non-increasing radius that are centered on the points in S and touch p. We build a binary tree $\mathcal{B}$ that we query with the points in S, and the answer of a query will correspond to the index of the first disk in the sequence $D_1, \ldots, D_{n+1}$ that does *not* contain the query point. The leaves of $\mathcal{B}$ correspond to these answers from left to right. Each inner node v stores the intersection I_v of all disks (except D_{n+1}) that correspond to the leaves in the subtree rooted at v. We label each node v with a pair $[a_v, b_v]$ encoding the set $S_v = \{a_v, \ldots, b_v\}$ of consecutive indices that correspond to these disks. In Figure 2 a tree with $n = 13$ is depicted. We build $\mathcal{B}$ in a bottom-up fashion. Each inner node has two children in the previous level, except possibly a level's rightmost node that can have a right child in an earlier level, see the node with label $[9, 13]$ in Figure 2.

Querying $\mathcal{B}$ with a query point q means to follow a path from the root to a leaf. In each inner node v with left child ℓ the test $q \in I_\ell$ is performed. If $q \in$

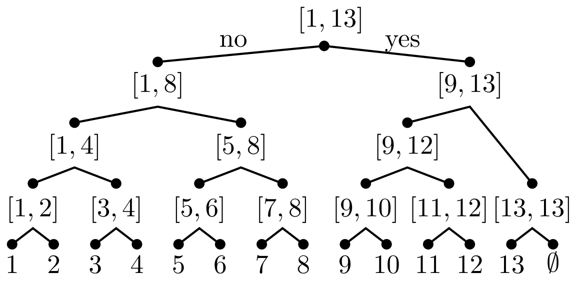

Fig. 2. The tree data structure $\mathcal{B}$ for $n = 13$.

I_ℓ, the query continues with the right, otherwise with the left child of v.

Other than in [2,4], we start with an empty skeleton of $\mathcal{B}$, i.e. all inner nodes v are labeled by $[a_v, b_v]$, but all intersections I_v are set to $\mathbb{R}^2$. Also other than in [4], the order in which we query becomes crucial. We go through the points $p_1, \ldots, p_n \in S$ in order of increasing index. Before querying $\mathcal{B}$ with p_i we update $\mathcal{B}$ by adding the new disk D_k centered on p_i (note that usually $k \neq i$) to the intersection I_v for each node v on the path from the root to the leaf that corresponds to D_k.

The result of a query with p_i is the disk D_j that corresponds to the leaf at the end of the query path π. If $j = n + 1$ then π is the rightmost root–leaf path. Consider the left children of the nodes on π. The sets S_ℓ that belong to these left children partition $\{1, \ldots, n\}$. In other words, the intersection of I_ℓ over these children is $D_1 \cap \cdots \cap D_n$. Since π is the rightmost root–leaf path, the containment queries in all nodes on π were answered positively. Thus q_i is contained in all disks currently in $\mathcal{B}$, i.e. $q_i \in D(q_1, p) \cap \cdots \cap D(q_i, p)$, where $D(a, b)$ is the disk centered at a that touches b. This means that none of $q_1, \ldots, q_i$ lies in the halfplane $h(p, q_i)$ that contains p and whose boundary is the perpendicular bisector of p and q_i. Otherwise [5, Lemma 1] would guarantee that $q_i \notin D(q_k, p)$ for the point q_k in $\{q_1, \ldots, q_i\}$ farthest from p in $h(p, q_i)$.

If $j \leq n$ we again consider the left children of the nodes on the query path π of q_i. The sets S_ℓ partition $\{1, \ldots, j - 1\}$ if we take only those left children ℓ into account that do not themselves lie on π. Similarly to above, the intersection of I_ℓ over these children is $D_1 \cap \cdots \cap D_{j-1}$. Thus q_i is contained in all D_k with $k < j$ that are currently in $\mathcal{B}$. On the other hand, since π is not the rightmost root–leaf path, there must be left children that lie on π. The last such left child v is the root of the subtree whose rightmost leaf corresponds to D_j. Thus v is associated with some set $S_v = \{i_v, \ldots, j\}$, where $1 \leq i_v \leq j$. Since we have already observed that q_i is contained in all D_k with $k < j$ that are currently in $\mathcal{B}$, but π came to v via a "no"-branch, we now know that $q_i \notin D_j$. Let m be such that $D_j = D(q_m, p)$. Note that $q_i \notin D(q_m, p)$ means that $D(q_m, p)$ was inserted in $\mathcal{B}$ before querying with q_i, and thus $m \leq i$. Since $q_i \notin D(q_m, p)$, and $q_i \in D(q_r, p)$ for all $r \leq i$ with $|pq_r| > |pq_m|$, [5, Lemma 1] yields that q_m is farthest from p in $\{q_1, \ldots, q_i\} \cap h(p, q_i)$.

The running time is as follows. Querying $\mathcal{B}$ takes $O(\log^2 n)$ time since the height of $\mathcal{B}$ is $O(\log n)$ and in each node of the query path the query point has to be located in the intersection I_v of some disks, which can be done in $O(\log n)$ time.

When we update $\mathcal{B}$ by adding a new disk D_j, we have to go from the root to the leaf that corresponds to D_j. In each node on this path we must compute $I_v \cap D_j$ and update our data structure for I_v. This can be done in $O(\log n)$ time per node by a procedure detailed in [5, proof of Lemma 4]. Thus each update also takes $O(\log^2 n)$ time.

Now the running time of $O(n \log^2 n)$ is obvious. The space consumption is $O(n \log n)$ since a) each disk contributes only to intersections stored on the path from the root to "its" leaf, and b) a disk that contributes to some intersection I_v adds at most one arc to the boundary of I_v [5, Fact 2]. $\square$

References

[1] M. Bădoiu, S. Har-Peled, and P. Indyk. Approximate clustering via core-sets. In *Proc. STOC'02*, pages 250–257, 2002.

[2] D. Z. Chen, O. Daescu, J. Hershberger, P. M. Kogge, and J. Snoeyink. Polygonal path approximation with angle constraints. In *Proc. SODA'01*, pages 342–343, Washington, D.C., 2001.

[3] O. Daescu and N. Mi. Polygonal path approximation: A query based approach. In *Proc. ISAAC'03*, volume 2906 of *LNCS*, pages 36–46, 2003. Springer.

[4] J. Gudmundsson, H. Haverkort, S.-M. Park, C.-S. Shin, and A. Wolff. Facility location and the geometric minimum-diameter spanning tree. In *Proc. APPROX '02*, vol. 2462 of *LNCS*, pages 146–160, 2002. Springer.

[5] J. Gudmundsson, H. Haverkort, S.-M. Park, C.-S. Shin, and A. Wolff. Facility location and the geometric minimum-diameter spanning tree. *Computational Geometry: Theory and Appl.*, 27(1):87–106, 2004.

[6] J. Matoušek. Efficient partition trees. *Discrete and Computational Geometry*, 8:315–334, 1992.