Farthest-Point Queries with Geometric and Combinatorial Constraints

Ovidiu Daescu[1]*, Ningfang Mi[2], Chan-Su Shin[3]**, and Alexander Wolff[4]***

[1] Department of Computer Science, University of Texas at Dallas, Richardson, TX 75083, USA. Email: daescu@utdallas.edu
[2] Department of Computer Science, College of William and Mary, P.O. Box 8795, Williamsburg, VA 23187-8795, USA. Email: ningfang@cs.wm.edu
[3] School of Electronics and Information Engineering, Hankuk University of Foreign Studies, Korea. Email: cssin@hufs.ac.kr
[4] Department of Computer Science, Karlsruhe University, P.O. Box 6980, D-76128 Karlsruhe, Germany. http://i11www.ira.uka.de/people/awolff

Abstract. In this paper we discuss farthest-point problems in which a set or sequence S of n points in the plane is given in advance and can be preprocessed to answer various queries efficiently. First, we give a data structure that can be used to compute the point farthest from a query line segment in $O(\log^2 n)$ time. Our data structure needs $O(n \log n)$ space and preprocessing time. To the best of our knowledge no solution to this problem has been suggested yet. Second, we show how to use this data structure to obtain an output-sensitive query-based algorithm for polygonal path simplification. Both results are based on a series of data structures for fundamental farthest-point queries that can be reduced to each other.

1 Introduction

Proximity problems are fundamental in computational geometry and have been studied intensively since Knuth [15] posed the post-office problem about three decades ago. In this paper we discuss farthest-point problems in which a set or sequence S of n points in the plane is given in advance and can be preprocessed to answer various queries efficiently. Our main results are the following.

First, we present a data structure that can be used to compute the point farthest from a query line segment in $O(\log^2 n)$ time. Our data structure needs $O(n \log n)$ space and preprocessing time. To the best of our knowledge no solution to this problem has been suggested yet.

Second, we design a data structure that can be used to simplify polygonal paths in the following sense: given a path $P = (p_1, \dots, p_n)$ and a real $\Delta > 0$ we want to find a subpath P' of P that goes from p_1 to p_n and consists exclusively of *Δ-approximating segments* according to the *tolerance-zone criterion*, i.e. a sequence of line segments $\overline{p_i p_k}$ with the property that each p_j with

* Supported by NSF grant CCF-0430366.
** Supported by Hankuk University of Foreign Studies Research Fund of 2005.
*** Supported by grant WO 758/4-1 of the German Science Foundation (DFG).

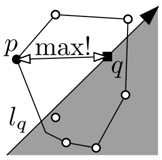

Fig. 1.
FV-halfplane.

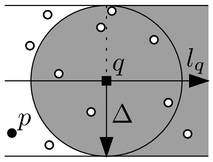

Fig. 2.
FP-halfstrip.

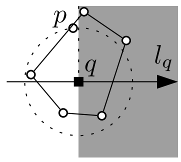

Fig. 3.
Counterexample.

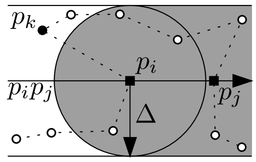

Fig. 4.
FIP-halfstrip.

$i < j < k$ is at most Δ away from $\overline{p_i p_k}$. We are interested in a min-# subpath, i.e. a subpath with the minimum number of vertices. This is motivated by data reduction (e.g. in geographic information systems) and considered an important problem—finding a near-linear solution is listed as problem 24 in the *Open Problems Project* [17]. Our query-based algorithm finds a min-# subpath in $O(n^2 \log^3 n)$ worst-case running time. This is slightly worse than the quadratic running time of the best incremental algorithm [6], but much better in practice since, as we will see later, the running time of our algorithm is output sensitive. Our algorithm has the same structure as a query-based algorithm [11] for the weaker *infinite-beam criterion* which requires that a vertex p_j of P that is short-cut by an edge $\overline{p_i p_k}$ of P' must be within distance Δ from the *line* through p_i and p_k. The algorithm [11] outperformed an incremental algorithm similar to [6] in an experimental evaluation.

Both main results are based on our solution of the following basic problem:

FarthestVertexInHalfplane (FV-halfplane): Preprocess a convex n-gon C for queries of the following type. Given (q, l_q), where q is a point and l_q is a directed line through q, decide whether there is a vertex of C to the left of l_q. If yes, report the one farthest from q. (See Figure 1.)

Other than one might think at first glance, this problem cannot be solved simply by binary search on the vertices of C since the distance from the query point q is not unimodal on the boundary of C. Our data structure for FV-halfplane answers queries in $O(\log^2 n)$ time given $O(n \log n)$ space and preprocessing time.

Next we address a problem whose solution yields our first main result, an efficient data structure for finding points farthest from query line segments.

FarthestPointInHalfstrip (FP-halfstrip): Preprocess a set S of n points for queries of the following type. Given a triplet (q, l_q, Δ), where q is a point and l_q is a directed line through q such that all points in S are within distance Δ from l_q, decide whether there is a point $p \in S$ such that (i) $\lvert qp \rvert \geq \Delta$, and (ii) the projection of p on l_q lies before q. If yes, report the point farthest from q that fulfills conditions (i) and (ii). (See Figure 2.)

We prove that if there are points fulfilling conditions (i) and (ii), then among these the one farthest from q among them lies on the convex hull of S. Note that this statement does not hold if we drop condition (i): in Figure 3 the point p is farthest from q among all points in S that fulfill condition (ii), but p does not lie on the convex hull of S. Thanks to condition (i), our data structure for FV-halfplane in fact solves FP-halfstrip within the same asymptotic bounds.

This in turn yields our first main result: we can preprocess a set S of n points in $O(n \log n)$ time and space such that the point in S farthest from a query line segment s can be reported in $O(\log^2 n)$ time.

For our second main result, which deals with polygonal path simplification, point order is important. Thus we consider an indexed version of FP-halfstrip:

> FARTHESTINDEXEDPOINTINHALFSTRIP (FIP-halfstrip): Preprocess a sequence $S = (p_1, \ldots, p_n)$ of points for queries of the following type. Given a triplet (i, j, Δ) such that all points p_k with $i < k < j$ are within distance Δ from the line $p_i p_j$, decide whether there is a point p_k with $i < k < j$ such that (i) $|p_i p_k| \geq \Delta$, and (ii) the projection of p_k on $p_i p_j$ lies before p_i. If yes, report the point p_k farthest from p_i that fulfills (i) and (ii). (See Figure 4.)

Our time and space bounds for FIP-halfstrip are a log-factor above those for FV-halfplane. The data structure for FIP-halfstrip yields an output-sensitive query-based algorithm for polygonal path simplification. Given a polygonal path $P = (p_1, \ldots, p_n)$ in $\mathbb{R}^2$ and a real $\Delta > 0$, the algorithm computes a subpath of P with the minimum number m_{tz} of vertices among all subpaths satisfying the tolerance-zone criterion. The algorithm runs in $O(F_{\text{tz}}(m_{\text{tz}})\, n \log^3 n)$ time and uses $O(n \log^2 n)$ space, where $F_{\text{tz}}(m_{\text{tz}}) \leq n$ is the number of vertices that can be reached from p_1 with at most $(m_{\text{tz}} - 2)$ Δ-approximating segments.

Next we look at a batched version of an indexed farthest-point problem. Given a sequence S of points, we want to observe how the point farthest from a fixed point p changes over time while we insert the points of S one after the other. In each round we ignore all those points that lie in a halfplane determined by the newly inserted point. Our solution assumes knowledge of S before the observation starts.

> BATCHEDFARTHESTINDEXEDPOINTINHALFPLANE (BFIP-halfplane): Given a sequence $S = (p_1, \ldots, p_n)$ of points and a point $p \notin S$, decide for each $i \in \{1, \ldots, n\}$ whether there is a point $p_f \in \{p_1, \ldots, p_i\}$ 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 takes $O(n \log^2 n)$ time and $O(n \log n)$ space.

Our paper is structured as follows. In Section 2 we briefly review related work. In Section 3 we first consider the problem FP-halfplane, a generalization of FV-halfplane where points do not have to be in convex position. In Section 4 we solve the convex case, i.e. FV-halfplane. In Section 5 we show that FP-halfstrip can be reduced to FV-halfplane and how this helps to solve the farthest-point-to-line-segment problem. In Section 6 we show how the data structure for FV-halfplane can be used to solve the indexed problem FIP-halfstrip. Section 7 settles the connection between FIP-halfstrip and polygonal path simplification. In Section 8 we address the batched problem BFIP-halfplane.

2 Previous Work

The problems we study are related to the nearest-point query problem [9, 18, 19] and to the all-pairs farthest- and closest-neighbors problem [21, 1, 3]. Cole and Yap [9] consider closest-point-to-line queries and present a data structure with $O(\log n)$ query time that needs $O(n^2)$ preprocessing time and space. A data structure with $O(n^{0.695})$ query time that needs $O(n \log n)$ preprocessing time and $O(n)$ space is presented by Mitra and Chaudhuri [18]. Using simplicial partitions, Mukhopadhyay [19] constructs in $O(n^{1+\varepsilon})$ time a data structure of size $O(n \log n)$ that finds a point *closest* to a query line in $O(n^{\frac{1}{2}+\varepsilon})$ time for arbitrary $\varepsilon > 0$. Finding a point *farthest* from a query line seems to be easier: it can be done by $O(\log n)$ time given $O(n \log n)$ preprocessing and $O(n)$ space, see Section 5. This data structure helps us to show how to find a point farthest from a query line segment in $O(\log^2 n)$ time given $O(n \log n)$ preprocessing and space. Bespamyatnikh and Snoeyink [5] show how to preprocess a set S of n points in $O(n \log n)$ time using $O(n)$ space such that the point closest to a query line segment *outside* the convex hull of S can be reported in $O(\log n)$ time. Using this data structure, Bespamyatnikh [4] shows how to solve in $O(n \log^2 n)$ time a batched problem where n points and n *disjoint* line segments are given and for each segment the *closest* point has to be determined. In contrast, our data structure for FP-halfstrip (see Section 5) answers *farthest*-point queries for an *arbitrary* line segment in $O(\log^2 n)$ time each, given $O(n \log n)$ space and preprocessing time.

While the all-pairs nearest neighbors of n points in a fixed dimension can be computed in optimal $O(n \log n)$ time [21], no algorithm is known to compute the all-pairs farthest neighbors of n points within the same time bound. Agarwal et al. [1] show that the all-pairs farthest neighbors in $\mathbb{R}^3$ can be computed in $O(n^{4/3} \log^{4/3} n)$ time. If the points are the vertices of a convex polygon in $\mathbb{R}^2$, the all-pairs farthest neighbors can be computed in linear time, even though the problem has a complexity of $\Omega(n \log n)$ for arbitrary points [3]. In $\mathbb{R}^3$ the convex case can be solved in $O(n \log^2 n)$ expected time [8].

Although the closest-point-to-line query problem and the all-pairs farthest neighbors problem are well understood, we are not aware of any published work on the farthest-point problems we consider.

3 Farthest Point In Halfplane

In this section, for completeness we address the following natural generalization of FV-halfplane.

> FARTHESTPOINTINHALFPLANE (FP-halfplane): Preprocess a set S of n points for queries of the following type. Given (q, l_q), where q is a point and l_q is a directed line through q, decide whether there is a point in S to the left of l_q. If yes, report the one farthest from q.

We use the following structure: a *simplicial partition* for a set S of n points in the plane is a collection of pairs $\Psi(S) = \{(S_1, t_1), (S_2, t_2), \ldots, (S_r, t_r)\}$, where the sets of type S_i partition S, and t_i is a triangle that contains S_i for $i = 1, \ldots, r$. An example of a point set S and a simplicial partition of S of size 4 are given in Figure 5. For a given simplicial partition $\Psi(S)$, the *crossing number* of a line l is the number of triangles of $\Psi(S)$ that l intersects. For example, the line l in Figure 5 has crossing number 3. The crossing number of $\Psi(S)$ is the maximum crossing number over all possible lines l. We say that a simplicial partition $\Psi(S)$ is *fine* if $|S_i| \leq 2n/r$, for every $1 \leq i \leq r$. Matoušek [16] showed the following important result on the construction of fine simplicial partitions with low crossing number:

Theorem 1 ([16]). *Let S be a set of n points in the plane, and let r be an integer with $1 \leq r \leq n/2$. Then a fine simplicial partition $\Psi(S)$ of size r with crossing number $O(\sqrt{r})$ exists. If r is constant, $\Psi(S)$ can be constructed in $O(n)$ time and space.*

Simplicial partitions are the basis of an efficient search data structure, called *partition tree*. The root of a partition tree of S has r children $v_1, \ldots, v_r$ that correspond one-to-one to the sets S_i in $\Psi(S)$. Each child v_i is the root of a recursively defined partition tree of S_i. The partition tree of n points can be computed in $O(n \log n)$ time and uses $O(n)$ space [16].

To solve the problem FP-halfplane we will take advantage of the *farthest-point Voronoi diagram*. Given a set of n sites in the plane, the farthest-point Voronoi diagram is a partition of the plane into cells, each of which is associated with a site and contains all the points in the plane that are *farther* from that site than from any other site. Unlike the *nearest-point* Voronoi diagram, in the farthest-point Voronoi diagram only the sites on the convex hull have a non-empty Voronoi region associated with them. In the plane, the farthest-point Voronoi diagram can be constructed in $O(n \log n)$ time if the sites are in general position [20]. When the sites are the vertices of a convex polygon, the diagram can be constructed [2] and preprocessed for planar point-location queries [12] in linear time.

We start by constructing the partition tree of S. Recall that for a node v_i of the tree, S_i is the subset of S stored at v_i, and t_i is the triangle of $\Psi(S)$ that contains S_i. Let $n_i = |S_i|$. For each node v_i we compute and store the farthest-point Voronoi diagram of S_i and preprocess it for planar point-location queries. This takes $\tau(n_i) = O(n_i \log n_i)$ time and uses $\sigma(n_i) = O(n_i)$ space. Let $T(n)$ and $S(n)$ be the total construction time and space consumption of these secondary data structures, respectively. They satisfy the following recurrences: $T(n) \leq \tau(n) + \sum_{i=1}^{r} T(n_i)$ and $S(n) = \sigma(n) + r + \sum_{i=1}^{r} S(n_i)$ for $n > 1$, and $T(1) = S(1) = 1$. Since we have that $\sum_{i=1}^{r} n_i = n$, that $n_i \leq 2n/r$, and that r is a constant, the general version of the Master theorem [10] yields that $T(n) = O(n \log^2 n)$. Similar arguments as for $T(n)$ show that $S(n) = O(n \log n)$.

When we query the partition tree, we want to find the point in S farthest from the query point q that is left of the directed line l_q. We have to consider two different kinds of point sets S_i. First we consider the $O(\sqrt{r})$ point sets S_i with $t_i \cap l_q \neq \emptyset$. For each such point set S_i, we recursively search in its simplicial partition $\Psi(S_i)$. Second we have to consider those point sets S_i that lie left of the line l_q. For each of these at most $r - O(\sqrt{r})$ point sets, we locate the query point q in the farthest-point Voronoi diagram to find the point farthest from q. Point location takes time logarithmic in the size of the partition. Therefore, we

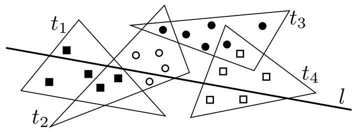

Fig. 5. A simplicial partition.

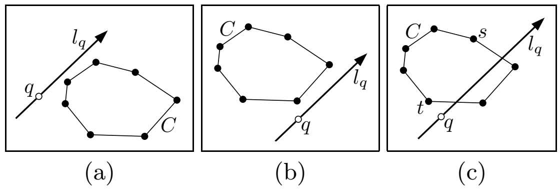

Fig. 6. Cases for the intersection of l_q with C.

get the following recurrence for the query time: $Q(1) = 1$ and for $n > 1$

$$Q(n) \quad \leq \quad r + \sum_{t_i \cap l_q = \emptyset} O(\log n_i) + \sum_{t_i \cap l_q \neq \emptyset} Q(n_i). \tag{1}$$

Let $c\sqrt{r} = O(\sqrt{r})$ be the crossing number of $\Psi(S)$. Given an arbitrary $\varepsilon > 0$, we can set $r = \lceil 2(c\sqrt{2})^{1/\varepsilon} \rceil$, which makes r a constant and yields $Q(n) = O(n^{1/2+\varepsilon})$ for n large enough, i.e. $n \geq 2r$. This can be seen by bounding the first sum in Inequality 1 by $O(r \log n)$ and the second sum by $c\sqrt{r} \cdot Q(2n/r)$. We sum up:

Theorem 2. *There is a data structure for FP-halfplane that answers queries in $O(n^{1/2+\varepsilon})$ time given $O(n \log^2 n)$ preprocessing time and $O(n \log n)$ space.*

4 Farthest Vertex in Halfplane

We now tackle FV-halfplane, the convex case of FP-halfplane. It is the basis of our solutions for the problems FP-halfstrip and FIP-halfstrip. The problem is to preprocess a convex n-gon C such that for a query pair (q, l_q), where q is a point and l_q is a directed line through q, one can efficiently decide whether there is a vertex of C left of l_q and if yes, report the one farthest from q.

Given a query pair (q, l_q), we first compute potential intersection points of l_q with the boundary ∂C of C. This can be done by binary search in $O(\log n)$ time since the distance from l_q is a unimodal function on ∂C. There are three possible cases, see Figure 6: (a) $l_q \cap C = \emptyset$ and C lies to the right of l_q; (b) $l_q \cap C = \emptyset$ and C lies to the left of l_q; (c) l_q has nonempty intersection with C. Knowing that $l_q \cap C = \emptyset$, case (a) can be handled in constant time. Case (b) reduces to finding the point on C farthest from l_q. This can be achieved in $O(\log n)$ time by locating the query point in the farthest-point Voronoi diagram of the vertices of C. In the remainder of this section we show how to handle case (c).

In the preprocessing phase, we construct a balanced binary tree T in $O(n \log n)$ 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 farthest-point Voronoi diagram V_u of the leaf descendants of u. This takes linear time for each level of T since all point sets are in convex position [2]. Within the same asymptotic time bound we then preprocess V_u for planar point-location queries [12]. Thus the computation of T takes $O(n \log n)$ time in total.

We query T as follows. Consider the edges of C intersected by l_q. If these edges are incident to the same vertex v of C to the left of l_q then we report v. Otherwise the edges have two different endpoints to the left of l_q. Let s be the first and t the second endpoint in counter-clockwise order on C, see Figure 6. We assume that the sequence of points on C that lie to the left of l_q does not contain both the rightmost vertex and its counter-clockwise predecessor. Otherwise the words left and right in the following description have to be exchanged.

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 until we reach the least common ancestor a of s and t. Whenever we get to a node $u \neq a$ 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 from a towards t. Whenever we go to the *right* child of a node $u \neq a$, we add to $\mathcal{V}$ the Voronoi diagram stored at the left child of u. Clearly, all points associated with these Voronoi diagrams are to the left of 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 farthest-point Voronoi diagram in $\mathcal{V}$ and keep track of the point farthest from q. This answers a query in $O(\log^2 n)$ time.

Theorem 3. *There is a data structure for FV-halfplane that answers queries in $O(\log^2 n)$ time given $O(n \log n)$ space and preprocessing time.*

5 Farthest Point in Halfstrip

In this section we want to preprocess a set S of n points for queries of the following type. Given a triplet (q, l_q, Δ), where q is a point and l_q is a directed line through q such that all points in S are within distance Δ from l_q, decide whether there is a point $p \in S$ such that (i) $|qp| \geq \Delta$, and (ii) the projection of p on l_q lies before q. If yes, report the point farthest from q that fulfills (i) and (ii). (See Figure 2.)

FP-halfstrip can be solved by the same approach as for FP-halfplane: construct a partition tree based on a fine simplicial partition in $O(n^{1+\varepsilon})$ time [16] and enhance it with a second-level data structure. For the points at each internal node of the partition tree, the second-level structure consists of the farthest-point Voronoi diagram preprocessed for planar point location.

We would prefer to use the faster solution for FV-halfplane, i.e. for the convex case. At first glance it seems that this is not possible, since among the points that fulfill condition (ii), the point p' farthest from the query point q may lie *inside* the convex hull C of S, see Figure 3. Condition (i), however, does in fact give us a way to use the data structure for FV-halfplane to solve FP-halfstrip. For a point q and a directed line l_q with $q \in l_q$ let l'_q be the directed line that results from turning l_q around q by $+90°$. Then the points whose projection on l_q lies before q are exactly the points to the left of l'_q.

Lemma 1. *Given a set $S \subset \mathbb{R}^2$ and a triplet (q, l_q, Δ), where q is a point and l_q is a directed line through q such that all points in S are within distance Δ from l_q, if there is a point $p \in S$ such that (i) $|qp| \geq \Delta$, and (ii) p lies to the left of l'_q,* *then among all points in S to the left of l'_q the point farthest from q is a vertex of the convex hull C of S.*

Proof. Let Σ be the closed strip that is bounded by the two lines at distance Δ from l_q and let H be the part of S to the right of l'_q. In Figure 7, Σ is the whole shaded area, H is the darker part. Let p be the point farthest from q to the left of l'_q, let D be a disk centered at q that touches p, and let $D' = D \cap \Sigma$. In Figure 7, the boundary of D is dotted, that of D' is bold solid. Finally let $U = D' \cup H$. Then p lies on the boundary of U. If $|pq| \geq \Delta$, U is convex. Thus for any (finite) set F with $p \in F \subset U$ it holds that p is a vertex of the convex hull of F. □

Since the convex hull of S can be computed in $O(n \log n)$ time, we have:

Theorem 4. *There is a data structure for FP-halfstrip that answers queries in $O(\log^2 n)$ time given $O(n \log n)$ space and preprocessing time.*

This yields our first main result, a data structure for finding the point farthest from a query segment.

Theorem 5. *Given a set S of n points, we can construct in $O(n \log n)$ space and preprocessing time a data structure that for any line segment s determines in $O(\log^2 n)$ time the point in S farthest from s.*

Proof. Let $s = \overline{uv}$ and let $\ell = uv$ be the line that is directed from u to v. There are two mutually exclusive cases. In the first case the point farthest from s is also the point farthest from ℓ. For this case we preprocess S by computing in $O(n \log n)$ time the convex hull C of S. Then this case can be solved by binary search in $O(\log n)$ time since the distance from ℓ is unimodal on C. Note that the point farthest from ℓ also gives us the smallest value Δ such that S lies within a Δ-strip around ℓ. For the second case, let S_w ($w \in \{u, v\}$) be the set of all points in S that are separated from s by the line orthogonal to s in w. In this case the point farthest from s is the point in S_u farthest from u or the point in S_v farthest from v. These two points can be determined within the desired time and space bounds by querying a data structure for FP-halfstrip with the triplets (u, uv, Δ) and (v, vu, Δ). □

6 Farthest Indexed Point in Halfstrip

We solve FIP-halfstrip, the indexed version of FP-halfstrip, in a way similar to FV-halfplane. At the same time we use the data structure for FV-halfplane as a plug-in. Let the points in the input sequence S be denoted by $p_1, \ldots, p_n$. In the preprocessing phase we construct a balanced binary tree $\mathcal{T}$ of the same structure as for FV-halfplane. The i-th leaf of $\mathcal{T}$ is associated with the point $p_i \in S$. We build the tree $\mathcal{T}$ bottom-up. At each internal node v, we compute and store the convex hull C_v of the leaf descendants $p_{i(v)}, \ldots, p_{j(v)}$ of v. We also compute and store at v a secondary level data structure, namely the tree described in

Section 4 that solves FV-halfplane (i.e. FP-halfstrip) for the vertices of C_v. The overall computation of $\mathcal{T}$ requires $O(n \log^2 n)$ time and space.

A query is also very similar to FV-halfplane: for a query (i, j, Δ), we follow the unique path from p_i to p_j in $\mathcal{T}$ collecting a set $\mathcal{C}$ of $O(\log n)$ convex hulls whose union contains all points p_k with $i < k < j$. This is done in the same way as with the set of farthest-point Voronoi diagrams in Section 4. For each convex hull $C_v \in \mathcal{C}$, we solve FP-halfstrip for the triplet $(p_i, p_i p_j, \Delta)$ using the secondary data structure stored at vertex v of $\mathcal{T}$. (Compare the situations in Figures 2 and 4!) Thus we can decide in $O(\log^2 |C_v|)$ time whether there is a k, $i(v) \leq k \leq j(v)$, such that the point p_k satisfies the two FIP-halfstrip conditions. Since the size of the set $\mathcal{C}$ is $O(\log n)$, the overall query time is $O(\log^3 n)$.

Theorem 6. *There is a data structure for FIP-halfstrip that answers queries in $O(\log^3 n)$ time given $O(n \log^2 n)$ space and preprocessing time.*

7 Polygonal Path Simplification and FIP-halfstrip

In this section we use our solution of FIP-halfstrip to extend a recent result of Daescu and Mi [11] for the min-# version of the polygonal path simplification problem: Given a polygonal path $P = (p_1, p_2, \ldots, p_n)$, with n vertices, and an error tolerance Δ, find a subpath $P' = (p_{i_1} = p_1, p_{i_2}, \ldots, p_{i_m} = p_n)$ of P such that the vertices of P' are an ordered subset of the vertices of P, each line segment $\overline{p_{i_j} p_{i_{j+1}}}$ of P' is a Δ-approximation of the corresponding subpath $(p_{i_j}, p_{i_j+1}, \ldots, p_{i_{j+1}})$ of P, and the number of vertices m of P' is minimized.

To decide whether a line segment of P' is a Δ-approximation of the corresponding subpath of P, two error criteria are commonly used: the *tolerance-zone* criterion and the *infinite-beam* criterion. The first criterion produces a compressed version that better captures the features of the original path, while the second gives a better degree of compression. According to the tolerance-zone criterion, all vertices of the approximated subpath of P must be within distance Δ from the approximating line segment of P', while according to the infinite-beam criterion all vertices of the approximated subpath must be within distance Δ from the line supporting the approximating line segment.

In [11] an output sensitive, query based algorithm is presented for solving the min-# problem according to the infinite-beam criterion. There, it is also shown that the algorithm is very fast in practice and outperforms previous algorithms. Since a Δ-approximation segment according to the tolerance-zone criterion is also a Δ-approximation segment according to the infinite-beam criterion, extending the algorithm in [11] to the tolerance-zone criterion reduces to answering queries on indexed points, as formulated in problem FIP-halfstrip. More precisely, if the line segment $\overline{p_{i_j} p_{i_{j+1}}}$ of P', for $j \in \{1, \ldots, m_{\text{tz}} - 1\}$, approximates the subpath $P[i_j, i_{j+1}] = (p_{i_j}, p_{i_j+1}, \ldots, p_{i_{j+1}})$ of P according to the infinite-beam criterion, then all the vertices of the subpath $P[i_j, i_{j+1}]$ are within distance Δ from the line $p_{i_j} p_{i_{j+1}}$. Let l_j and l'_j denote the lines orthogonal to $p_{i_j} p_{i_{j+1}}$ in p_{i_j} and $p_{i_{j+1}}$, respectively. If for each vertex p_k of P, $i_j \leq k \leq i_{j+1}$, with the property that l_j separates p_k and $p_{i_{j+1}}$ or l'_j separates p_k and p_{i_j}, we have that p_k is within distance Δ from p_{i_j} or $p_{i_{j+1}}$, respectively, then every vertex on the subpath $P[i_j, i_{j+1}]$ is within distance Δ from the line segment $\overline{p_{i_j} p_{i_{j+1}}}$. Clearly, this reduces to solving FIP-halfstrip, see Section 6. The result is an output sensitive, query-based algorithm for solving the min-# problem under the tolerance-zone criterion.

Theorem 7. *Given a polygonal path $P = (p_1, p_2, \ldots, p_n)$ in the plane, the min-# problem under the tolerance-zone criterion can be solved in $O(F_{\mathrm{tz}}(m_{\mathrm{tz}})\, n \log^3 n)$ time using $O(n \log^2 n)$ space, where $F_{\mathrm{tz}}(m_{\mathrm{tz}}) \leq n$ is the number of vertices that can be reached from p_1 with at most $(m_{\mathrm{tz}} - 2)$ Δ-approximating segments, and m_{tz} is the number of vertices on an optimal approximating path.*

Proof. The algorithm is similar to the query-based algorithm in [11], except that now each query takes $O(\log n + \log^3 n)$ time instead of $O(\log n)$ time: as in [11] we first spend $O(\log n)$ time to decide whether some segment $\overline{p_i p_j}$ is a Δ-approximating segment according to the infinite-beam criterion. If the answer is positive, we now use Theorem 6 and spend additional $O(\log^3 n)$ time to decide whether $\overline{p_i p_j}$ is a Δ-approximating segment according to the tolerance-zone criterion. □

8 Batched Farthest Indexed Point In Halfplane

In this section we consider the problem BFIP-halfplane: given a sequence $S = (p_1, \ldots, p_n)$ of points and a point $p \notin S$, decide for each $i \in \{1, \ldots, n\}$ whether there is a point $p_f \in \{p_1, \ldots, p_i\}$ 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.

A version of BFIP-halfplane without the index restriction has been considered in [14]. There the problem of computing the minimum-sum dipolar spanning tree (MSST) is considered. The MSST of a point set S is a tree with vertex set S and two non-leaf nodes $x, y \in S$ that minimizes $|xy| + \max\{r_x, r_y\}$, where r_x and r_y are the radii of two disks centered at x and y whose union covers S. In the computation of the MSST the following subproblem shows up: 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 [14] 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 [7] 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 the plane. In [14], a version of OLBES where all disks intersect a common point is solved in $O(n \log n)$ time and $O(n)$ space by sweeping an arrangement of circular arcs. The same problem is solved in [13] in $O(n \log n)$ time and space using a tree data structure and fractional cascading.

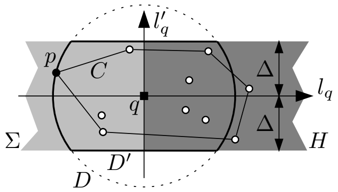

Fig. 7. Proof of Lemma 1.

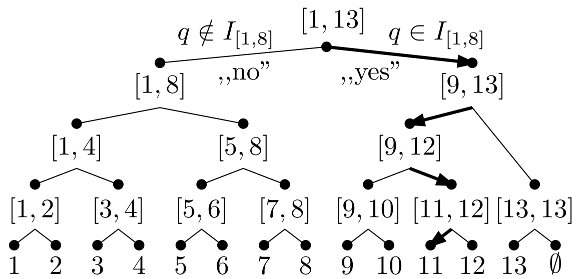

Fig. 8. The tree T for $n = 13$. Bold arrows indicate the search path for a point $q \in (D_1 \cap \cdots \cap D_{10}) \setminus D_{11}$.

We are interested in the problem BFIP-halfplane since it adds a time component to the pure query problem FIP-halfplane. We want to keep track on how the point farthest from the fixed point p changes over time while we insert the points of S one after the other and ignore all those points that lie in a halfplane determined by the newly inserted point. We solve this problem by setting up a tree data structure similar to that in the proof of Lemma 2 in [13]. Here, however, we must solve a different OLBES problem in each query and thus need to modify our tree successively.

We need some notation. Let $D(x, p)$ be the open disk centered at x that touches p and let $h(p, q)$ be the closed halfplane that contains p and whose boundary is the perpendicular bisector of p and q. Note that $x \in h(p, q)$ is equivalent to $|xp| \leq |xq|$, which in turn is equivalent to $q \notin D(x, p)$.

Now we give our algorithm for BFIP-halfplane. Let $p_1, \ldots, p_n$ be the sequence of input points. We first sort the n disks $D(p_i, p)$ in order of non-increasing radius. Let $D_1, \ldots, D_n$ be the resulting sequence. Thus, a disk D_k in this sequence corresponds to some disk $D(p_i, p)$ and in general $k \neq i$. Let $D_{n+1} = \emptyset$. We build a binary tree T as follows. The leaves of T correspond to the sequence $D_1, \ldots, D_{n+1}$, from left to right. Each inner node v stores the intersection I_v of the leaf descendants of v (excluding D_{n+1}). 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 the disks associated with the leaves of the subtree of T rooted at v. In Figure 8 a tree with $n = 13$ is depicted. We build T in a bottom-up fashion. Each inner node has two children in the previous level, except possibly a level's rightmost node, which can have a right child in an earlier level, see the node with label [9, 13] in Figure 8. We query T 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.

Unlike [7, 13] we start with an empty skeleton of T, i.e. all inner nodes v are labeled by $[a_v, b_v]$, but all leaves and all intersections I_v are set to $\mathbb{R}^2$. 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. When we query with the point p_i, only the disks $D(p_1, p), \ldots, D(p_{i-1}, p)$ have been inserted in T. Before querying T with p_i we update T by adding the new disk $D_k = D(p_i, p)$ (recall 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. Querying T with p_i amounts to following a path from the root to a leaf. In each inner node v with left child ℓ, the test $p_i \in I_\ell$ is performed. If $p_i \in I_\ell$, the query continues with the right, otherwise with the left child of v, see Figure 8. The leaf at the end of the query path π determines what our algorithm reports. Let D_j be the disk corresponding to that leaf. If $j \leq n$, then we report that p_j is the point farthest from p in $\{p_1, \ldots, p_i\} \cap h(p, p_i)$. Otherwise (i.e. if $j = n+1$) we report that $\{p_1, \ldots, p_i\} \cap h(p, p_i)$ is empty. This algorithm yields the following.

Theorem 8. *Given a sequence S of n points and a point $p \notin S$, BFIP-halfplane can be solved in $O(n \log^2 n)$ time and $O(n \log n)$ space.*

Proof. We first show the correctness of the above algorithm. Depending on the index of the disk D_j we consider two cases. The first case is that $j = n + 1$. 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 p_i is contained in all disks currently in T, i.e. $p_i \in D(p_1, p) \cap \cdots \cap D(p_i, p)$. This means that none of $p_1, \ldots, p_i$ lies in $h(p, p_i)$. Otherwise Lemma 1 in [14] would guarantee that $p_i \notin D(p_k, p)$ for the point $p_k \in \{p_1, \ldots, p_i\}$ farthest from p in $h(p, p_i)$.

The second case is $j \leq n$. Again we consider the left children of the nodes on the query path π of p_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, p_i is contained in all D_k with $k < j$ that are currently in T. On the other hand, since π is not the rightmost root–leaf path, π contains at least one node that is a left child of a node 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 p_i is contained in all D_k with $k < j$ that are currently in T, but π came to v via a "no"-branch ($p_i \notin I_v$), we now know that $p_i \notin D_j$. Let m be such that $D_j = D(p_m, p)$. Note that $p_i \notin D(p_m, p)$ means that $D(p_m, p)$ was inserted in T before querying with p_i, and thus $m \leq i$. Since $p_i \notin D(p_m, p)$, and $p_i \in D(p_r, p)$ for all $r \leq i$ with $|pp_r| > |pp_m|$, Lemma 1 in [14] yields that p_m is farthest from p in $\{p_1, \ldots, p_i\} \cap h(p, p_i)$.

The total running time is $O(n \log^2 n)$ since both querying and updating T take $O(\log^2 n)$ time for each p_i. The space consumption is $O(n \log n)$ since each disk contributes at most one arc to each intersection stored on the path from the root to "its" leaf. For details refer to the full version of this paper. □

The definition of BFIP-halfplane and Theorem 8 can be generalized without much effort as follows. Instead of insisting that the separator of p and p_i splits $\overline{pp_i}$ in a ratio of 1:1, any other ratio can be used as long as the split is orthogonal.

References

[1] P. K. Agarwal, J. Matoušek, and S. Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Computational Geometry: Theory and Applications*, 1(4):189–201, 1992.

[2] A. Aggarwal, L. J. Guibas, J. B. Saxe, and P. W. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete & Computational Geometry*, 4(6):591–604, 1989.

[3] A. Aggarwal and D. Kravets. A linear time algorithm for finding all farthest neighbors in a convex polygon. *Information Processing letters*, 31(1):17–20, 1989.

[4] S. Bespamyatnikh. Computing closest points for segments. *Int. J. Comput. Geom. Appl.*, 13(5):419–438, 2003.

[5] S. Bespamyatnikh and J. Snoeyink. Queries with segments in Voronoi diagrams. *Comput. Geom. Theory Appl.*, 16(1):23–33, 2000.

[6] D. Z. Chen and O. Daescu. Space-efficient algorithms for approximating polygonal curves in two dimensional space. *International Journal of Computational Geometry and Applications*, 13(2):95–112, 2003.

[7] D. Z. Chen, O. Daescu, J. Hershberger, P. M. Kogge, and J. Snoeyink. Polygonal path approximation with angle constraints. In *Proc. 12th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'01)*, pp. 342–343, 2001.

[8] O. Cheong, C.-S. Shin, and A. Vigneron. Computing farthest neighbors on a convex polytope. *Theoretical Computer Science*, 296:47–58, 2003.

[9] R. Cole and C.-K. Yap. Geometric retrieval problems. In *Proc. 24th Ann. IEEE Symposium on Foundations of Computer Science (FOCS'83)*, pp. 112–121, 1983.

[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[11] O. Daescu and N. Mi. Polygonal path approximation: A query based approach. *Computational Geometry: Theory and Applications*, 30(1):41–58, 2005.

[12] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15:317–340, 1986.

[13] J. Gudmundsson, H. Haverkort, S.-M. Park, C.-S. Shin, and A. Wolff. Facility location and the geometric minimum-diameter spanning tree. In *Proc. 5th Int. Workshop on Approx. Algorithms for Combinatorial Optimization (APPROX'02)*, vol. 2462 of *Lecture Notes Comput. Sci.*, pp. 146–160. Springer-Verlag, 2002.

[14] 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 Applications*, 27(1):87–106, 2004.

[15] D. E. Knuth. *The Art of Computer Programming*, volume 3, chapter Sorting and Searching. Addison-Wesley, Reading, MA, 1973.

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

[17] J. S. B. Mitchell and J. O'Rourke. Computational geometry column 42. *SIGACT News*, 32(3):63–72, 2001.

[18] P. Mitra and B. Chaudhuri. Efficiently computing the closest point to a query line. *Pattern Recognition Letters*, 19(11):1027–1035, 1998.

[19] A. Mukhopadhyay. Using simplicial partitions to determine a closest point to a query line. In *Proc. Canadian Conf. Comp. Geom. (CCCG'02)*, pp. 10–12, 2002.

[20] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1990.

[21] P. M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete and Computational Geometry*, 4:101–115, 1989.