The Discrete Median and Center Line Segment Problems in the Plane

Ovidiu Daescu* Ka Yaw Teo†

Abstract

Let \( P \) be a set of \( n \) points in the plane. The discrete median line segment of \( P \) is the line segment with both its endpoints in \( P \) such that the sum of the distances from \( P \) to the line segment is minimized. Similarly, the discrete center line segment of \( P \) is the line segment bounded by two points of \( P \) such that the maximum of the distances from \( P \) to the line segment is minimized. We present exact algorithms for computing the discrete median and center line segments of \( P \). Our algorithms run in \( O(n^2) \) time and use \( O(n^2) \) space.

1 Introduction

In this paper, we consider the following two problems.

Discrete median line segment problem Given a set \( P \) of \( n \) points in \( \mathbb{R}^2 \), locate a line segment bounded by two points of \( P \) such that the sum of the Euclidean distances from \( P \) to the line segment is minimized.

Discrete center line segment problem Given a set \( P \) of \( n \) points in \( \mathbb{R}^2 \), locate a line segment bounded by two points of \( P \) such that the maximum of the Euclidean distances from \( P \) to the line segment is minimized.

It appears, as far as the authors are aware, that the problems defined above, albeit interesting and geometric in nature, have been seemingly overlooked in facility location theory. The proposed problems are closely related to a class of “discrete” problems in facility location theory, where the goal is to select one point (or several) from a given set of points \( P \) so as to minimize an objective function that is distance-dependent with respect to \( P \).

In general, there are two types of problems in facility location theory depending on the objective function used – i) center (minimax) and ii) median (minsum). In regards to the discrete point facility location problems aforementioned, the discrete center problem asks to locate a point in \( P \) that minimizes the maximum of the distances between the points of \( P \) and the located point. This is analogous to finding the smallest disk centered at a point of \( P \) and containing \( P \). The discrete center problem can be solved in \( O(n \log n) \) time using the farthest-neighbor Voronoi diagram of \( P \) [5, Chapter 7]. The discrete median, which is commonly known as the medoid, is a point in \( P \) that has the minimal sum of distances to \( P \). One can find the medoid of \( P \) by simply computing all \( O(n^2) \) pairwise distances. However, it has been argued that no exact algorithm exists for solving the medoid problem in \( o(n^2) \) time [8].

2 Our results

We begin in Section 3 by addressing the discrete median line segment problem. First, we solve the problem in \( O(n^2 \log n) \) time by using enumeration enhanced with logarithmic query-time data structures (Section 3.1). We then improve the time complexity of our algorithm to \( O(n^2) \) by reducing the query time of our data structures to amortized \( O(1) \) (Section 3.2).

In the process of deriving our solution to the discrete median line segment problem, we develop efficient data structures supporting half-plane distance-sum queries (see Subproblems 1 and 2 for detailed definitions), which happen to be more general than required for solving our problem.

We proceed to solve the discrete center line segment problem in Section 4 along similar lines. We obtain an \( O(n^2 \log n) \)-time algorithm for the problem based on the query data structures proposed in [1, 4] (Section 4.1). We then follow by deriving an \( O(n^2) \)-time algorithm, which requires a data preprocessing approach different from that in the prior algorithm (Section 4.2).

In Section 5, we show that, against intuition, the discrete median or center line segment of \( P \) does not necessarily have an endpoint at a vertex on the convex hull of \( P \). This rules out any algorithm whose time complexity depends on the number of vertices on the convex hull of \( P \), which is typically smaller than \( n \) for random (and many practical) sets of points.

We end the paper with some brief concluding remarks in Section 6.

3 Discrete median line segment

One can find the discrete median line segment in \( O(n^3) \) time using a brute-force method – namely, by enumerating all \( \binom{n}{2} \) candidate line segments (i.e., distinct pairs
of points in $P$) and computing the corresponding sum of $n - 1$ distances for each candidate line segment.

### 3.1 An $O(n^2 \log n)$-time algorithm

In this section, we derive an $O(n^2 \log n)$-time algorithm for the discrete median line segment problem. The idea is to preprocess $P$ into some data structures of logarithmic query time for use in computing the sum of distances for each candidate line segment. We begin by addressing the required query data structures, which are derived from solving the following two subproblems (refer to Figure 1).

**Subproblem 1**

Given a set $P$ of $n$ points in the plane, let $H$ be a query half-plane bounded by a line $L$ containing a point $p \in P$. Preprocess $P$ so that, for a point $p \in P$ and a half-plane $H$ given at query time, one can efficiently report the sum of the distances from $P \cap H$ to $p$.

**Subproblem 2**

Given a set $P$ of $n$ points in the plane, let $H$ be a query half-plane bounded by a line $L$ containing a point $p \in P$. Let $\rho$ be the ray emanating from $p$, perpendicular to $L$, and contained in $H$. Preprocess $P$ so that, for a point $p \in P$ and a half-plane $H$ given at query time, one can efficiently report the sum of the orthogonal distances from $P \cap H$ to $p$.

Here we give a brief description of the preprocessing procedure for solving the subproblems. For each point $p \in P$, we i) sort the points of $P \setminus \{p\}$ around $p$ in $O(n \log n)$ time, ii) define a sequence of $O(n)$ intervals in the sorted order such that $P \cap H$ remains constant within each interval, iii) enumerate the intervals in the sorted order so that it takes $O(1)$ time to evaluate the sum of distances in each interval, and iv) store the distance sums computed for the intervals in an $O(\log n)$-query time data structure. The details of the solutions to Subproblems 1 and 2 are presented in the following two subsections.

**Subproblem 1**

Recall that $L$ denotes a line passing through a point $p \in P$. Without loss of generality, let $H$ be one of the two half-planes bounded by $L$ (the other half-plane can be handled similarly due to symmetry). As line $L$ rotates around point $p$, a point $q \in P \setminus \{p\}$ may enter and leave $H$. These point-entering and -leaving events can be determined in $O(1)$ time each by computing the line passing through $p$ and each point $q \in P \setminus \{p\}$. Since a point $q \in P \setminus \{p\}$ can enter and leave $H$ at most once during a full rotation of line $L$ around point $p$, the total number of point-entering and -leaving events is bounded by $2n - 2$. These events can be sorted in counterclockwise order, according to the slopes of their corresponding lines $L$, in $O(n \log n)$ time.

Let $q_i$ and $q_j$ be the points in $P \setminus \{p\}$ associated with any two consecutive events in the sorted order. Note that, within the event interval bounded by $q_i$ and $q_j$, the subset of points $P \setminus \{p\}$ contained in $H$ remains constant, and so does the sum of their distances to $p$. The said distance sum for the set of points $(P \cap H) \setminus \{p\}$ for each of these event intervals can be computed as follows. We begin with the event interval with the smallest slope of $L$ (in the sorted order), for which we determine the set of points $(P \cap H) \setminus \{p\}$ and calculate the sum of their distances to $p$ in $O(n)$ time. For each subsequent event interval in the sorted order, the distance sum can be computed in constant time using that of the preceding event interval – that is, by adding to (or subtracting from) the current distance sum the distance between $p$ and the point entering (or leaving) $H$. Therefore, for a given point $p \in P$, the total time for the computation of the distance sums for all event intervals is bounded by $O(n)$.

A data structure $D_1$ can then be built as follows. For each point $p \in P$, we construct a simple logarithmic query-time data structure to store the distance sums computed for the event intervals. Given a query point $p \in P$ and a query half-plane $H$ defined by a line $L$ passing through $p$, we can use data structure $D_1$ to look up the event interval that contains the slope of line $L$, and report the sum of the distances from the points of $P \cap H$ to $p$ in $O(\log n)$ time.

**Lemma 1**

In Subproblem 1, a set $P$ of $n$ points can be preprocessed in $O(n^2 \log n)$ time into a data structure of size $O(n^2)$ so that, given a query point $p \in P$ and a query half-plane $H$, one can report the sum of the distances from $P \cap H$ to $p$ in $O(\log n)$ time.

**Subproblem 2**

The line containing ray $\rho$ can be described by $y = mx + c$. Let $x_i$ and $y_i$ be the $x$- and $y$-coordinates, respectively, of a point $p_i \in P$, where $1 \leq i \leq n$. Define $T^+ = \{i : y_i - mx_i > c\}$ and $T^- = \{i : y_i - mx_i < c\}$. 

![Figure 1: Illustration for Subproblems 1 and 2.](image-url)
The sum of the distances from \( P \cap H \) to \( \rho \) can then be expressed as
\[
\sum_{i \in T^+} \left( \frac{y_i - mx_i - c}{(m^2 + 1)^{1/2}} \right)
\]
\[
= (m^2 + 1)^{-1/2} \left[ \sum_{i \in T^+} (y_i - mx_i - c) \right]
\]
\[
- \sum_{i \in T^-} (y_i - mx_i - c)
\]
\[
= (m^2 + 1)^{-1/2} \left[ \sum_{i \in T^+} y_i + m \sum_{i \in T^-} x_i - \sum_{i \in T^+} 1 - \sum_{i \in T^-} 1 \right]
\]

In order to solve Subproblem 2, we will follow a strategy similar to that in Subproblem 1. Assume, without loss of generality, that \( H \) is one of the two half-plane bounded by \( L \) (the other case can be handled symmetrically). Observe that, when \( L \) rotates counterclockwise around \( p \), a point \( q \in P \setminus \{p\} \) may enter \( H \), leave \( H \), or move from set \( T^+ \) to \( T^- \). Each of these point-entering and -leaving events can be determined in constant time by computing i) the line \( L \) passing through \( p \) and each point \( q \in P \setminus \{p\} \), and ii) the ray \( \rho \) emanating from \( p \) and passing through each point \( q \in P \setminus \{p\} \). The total number of events is at most \( 3n - 3 \). These events can be sorted in counterclockwise order, according to the slopes of their respective lines \( L \), in \( O(n \log n) \) time.

Notice that, within any interval delimited by two consecutive events in the sorted order, sets \( T^+ \) and \( T^- \) remain constant. We will determine, for each event interval, the following set \( Q \) of values (from Equation 1)
\[
\sum_{i \in T^+} x_i, \sum_{i \in T^-} x_i, \sum_{i \in T^+} y_i, \sum_{i \in T^-} y_i, \sum_{i \in T^+} 1, \sum_{i \in T^-} 1
\]
by keeping track of \( T^+ \) and \( T^- \) as we sweep the event intervals in the counterclockwise order. We begin with the first event interval in the ordering by determining its corresponding sets \( T^+ \) and \( T^- \), as well as the respective set of values \( Q \), in \( O(n) \) time. For each subsequent event interval, we can determine the corresponding set of values \( Q \) by updating those of the preceding event interval in \( O(1) \) time. Hence, for a point \( p \in P \), it takes \( O(n) \) time to compute \( Q \) for all event intervals.

We can then build a query data structure \( D_2 \) as follows. For each point \( p \in P \), we construct a linear-size data structure with a logarithmic query time to store the sets of values \( Q \) computed for the event intervals. Given a query point \( p \in P \) and a query half-plane \( H \) bounded by a line \( L \) passing through \( p \) (along with the calculated values of parameters \( m \) and \( c \) associated with the line supporting ray \( \rho \)), by employing data structure \( D_2 \), we retrieve in \( O(\log n) \) time the corresponding set of values \( Q \), from which we can calculate the sum of the distances from the points of \( P \cap H \) to \( \rho \) using Equation 1 in constant time.

**Lemma 2** In Subproblem 2, a set \( P \) of \( n \) points can be preprocessed in \( O(n^2 \log n) \) time into a data structure of size \( O(n^2) \) so that, given a query point \( p \in P \) and a query half-plane \( H \), one can report the sum of the orthogonal distances from \( P \cap H \) to \( \rho \) in \( O(\log n) \) time.

![Figure 2: Computing the distances from P to s.](image)
The discrete median line segment problem can be solved in \(O(n^2 \log n)\) time using \(O(n^2)\) space.

Remark 1 It is worth noting that the logarithmic query-time data structures given in Lemmas 1 and 2 are in fact more general than necessary to solve our problem. Namely, using those data structures, the distance sum can be reported for any query half-plane defined by a line passing through a point \(p \in P\), whereas each query half-plane in our problem is always associated with a line passing through a pair of points in \(P\). This observation would become one of the keys in improving the time bound of our algorithm to \(O(n^2)\), as detailed in the next section.

3.2 An \(O(n^2)\)-time algorithm

We can reduce the running time of the algorithm above by an \(O(\log n)\) factor as follows.

Using the point-line duality transform, point set \(P\) can be mapped into a set of \(n\) lines, whose arrangement can be constructed in \(O(n^2)\) time using \(O(n^2)\) space \([2, 6]\). For any point \(p \in P\) in the primal plane, let \(p^*\) denote its corresponding line in the dual plane. Observe that the line containing \(p\) and a point \(q \in P \setminus \{p\}\) in the primal plane corresponds to the intersection point of lines \(p^*\) and \(q^*\) in the dual plane. By the properties of the duality transform, the ordering of slopes of the lines passing through \(p\) and every point \(q \in P \setminus \{p\}\) in the primal plane is equivalent to the ordering of \(x\)-coordinates of the intersections between line \(p^*\) and each line \(q^*\) in the dual plane. Using this trait, for any point \(p \in P\), the set of points \(P \setminus \{p\}\) can be obtained in sorted order around \(p\) by simply traversing the vertices along the dual line of point \(p\) in \(O(n)\) time. Notice that these sorted points correspond to the endpoints of the sorted event intervals in Subproblems 1 and 2. Consequently, the preprocessing time of the query data structures given in Lemmas 1 and 2 is reduced to \(O(n^2)\).

In addition, we note that each query half-plane in our problem is associated with a pair of points in \(P\). We can predetermine and index each of these query half-planes, and use the static indices to create perfect hash tables \([3, \text{Chapter 11}]\) for constant-time look-ups (in the worst case) in place of the current logarithmic-time query data structures \(D_1\) and \(D_2\).

Specifically, for a candidate line segment \(s\) bounded by a pair of points \(a\) and \(b\) in \(P\), the respective set of query half-planes consists of \(H_a\), \(H_b\), and \(H_a^* = \mathbb{R} \setminus H_a\) (Figure 2). In the ensuing discussion, we will give an argument for query half-plane \(H_a\), and an analogous argument can be made about query half-planes \(H_b\) and \(H_b^*\).

Recall that a query half-plane \(H_a\) is uniquely defined by \(a\) and \(b\) (i.e., a pair of points in \(P\)) as being i) delimited by the line passing through \(a\) and orthogonal to \(s_1\), and ii) not containing \(b\). For a given point \(a \in P\), let \(L\) denote any line passing through \(a\), and let \(H\) be one of the two half-planes bounded by \(L\). In Subproblem 1, a point-entering or leaving event is indicated by line \(L\) passing through a point \(q \in P \setminus \{a\}\). Let \(r\) be the ray emanating from \(a\), perpendicular to \(L\), and not contained in \(H\) (see Figure 3). Notice that a query half-plane \(H_a\) is equivalent to \(H\) when ray \(r\) passes through a point \(b \in P \setminus \{a\}\). Thus, for a candidate point \(a \in P\), there exists a set of \(n - 1\) query half-planes \(H_a\), due to \(n - 1\) other possible points \(b \in P \setminus \{a\}\). For each point \(a \in P\), by computing the ray \(r\) passing through each point \(b \in P \setminus \{a\}\), we can obtain in advance the set of all \(O(n)\) possible query half-planes \(H_a\) according to the counterclockwise ordering of points \(b \in P \setminus \{a\}\) around point \(a\).

In Subproblem 1, when we perform the sweep procedure for a point \(p = a \in P\) by rotating line \(L\) counterclockwise around \(a\), in addition to processing each point-entering or leaving event due to line \(L\) passing through a point of \(P \setminus \{a\}\), we record the distance sum for each query half-planes \(H_a\) as ray \(r\) passes through a point \(b \in P \setminus \{a\}\). For each point \(a \in P\), we create a linear-size (perfect) hash table that maps each query half-plane \(H_a\) to its corresponding sum of distances. As a result, we can perform each query in \(O(1)\) time.

Theorem 4 The discrete median line segment problem can be solved in \(O(n^2)\) time using \(O(n^2)\) space.

Figure 3: Keeping track of query half-plane \(H_a\) in Subproblem 1 as line \(L\) and its perpendicular ray \(r\) rotate around a point \(a \in P\) in a counterclockwise direction (illustrations from left to right).
4 Discrete center line segment

Naively, we can find the discrete center line segment for point set \( P \) in \( O(n^3) \) time (with a linear space usage) by simply enumerating all \( O(n^2) \) candidate line segments and determining the farthest point of \( P \) from each candidate line segment in \( O(n) \) time.

Before proceeding any further, we denote by i) \( |P| \) the convex hull of \( P \), ii) \( bd[P] \) the boundary of \( |P| \), and iii) \( int|P| \) the interior of \( |P| \).

4.1 An \( O(n^2 \log n) \)-time algorithm

We can derive an \( O(n^2 \log n) \)-time algorithm for the discrete center line segment problem based on the previous results given by Aronov et al. [1] and Daescu et al. [4]. According to [4, Theorem 6], after a preprocessing of \( P \) that takes \( O(n \log n) \) time and space, the farthest point of \( P \) from a query line segment can be determined in \( O(\log^2 n) \) time. As a direct consequence, we can compute the discrete center line segment for \( P \) in \( O(n^2 \log^2 n) \) time.

As shown in [4, Theorems 5 and 6], the time and space complexities of finding the farthest point of \( P \) from a query line segment are dominated by those of solving the following key subproblem (half-plane farthest point queries):

**Theorem 5** The discrete center line segment problem can be solved in \( O(n^2 \log n) \) time using \( O(n \log^3 n) \) space.

4.2 An \( O(n^2) \)-time algorithm

In this section, we derive an \( O(n^2) \)-time algorithm for computing the discrete center line segment of \( P \).

Let \( s \) denote a line segment bounded by two points of \( P \). We begin with the following observation.

**Observation 1** The point of \( P \) farthest from \( s \) is a vertex of \( |P| \).

**Proof.** Since \( s \) is a line segment with both its endpoints in \( P \), \( s \) must lie within \( |P| \); that is, \( s \subset bd|P| \cup int|P| \). Note that, for any point \( p \in bd|P| \cup int|P| \), the point of \( P \) farthest from \( p \) is a vertex of \( |P| \). Thus, the farthest point of \( P \) from any point on \( s \) is a vertex of \( |P| \). We conclude that the point of \( P \) farthest from \( s \) must be a vertex of \( |P| \). □

Let \( \alpha \) be the point of \( P \) farthest from \( s \). Let \( \beta \) be the closest point on \( s \) to \( \alpha \). The Euclidean distance between \( \alpha \) and \( s \) is defined as the distance between \( \alpha \) and \( \beta \).

Let \( a \) and \( b \) denote the two endpoints of \( s \). Recall that \( H_a \) (resp. \( H_b \)) is the half-plane i) bounded by the line passing through \( a \) (resp. \( b \)) and perpendicular to \( s \), and ii) not containing \( s \). In addition, \( H_{ab} = \mathbb{R}^2 \setminus (H_a \cup H_b) \).

If \( \alpha \in H_a \), then the closest point on \( s \) to \( \alpha \) is \( a \). Similarly, if \( \alpha \in H_b \), then the closest point on \( s \) to \( \alpha \) is \( b \). If \( \alpha \in H_{ab} \), then the closest point on \( s \) to \( \alpha \) is an interior point of \( s \). Furthermore, if \( \alpha \in H_{ab} \), then \( \alpha \) is also the point of \( P \) farthest from the line containing \( s \).

Based on the observations above, for line segment \( s \), we can find the farthest point of \( P \) from \( s \) using the following approach. We determine

I. the point of \( |P| \cap H_a \) farthest from \( a \),

II. the point of \( |P| \cap H_b \) farthest from \( b \), and

III. the point of \( |P| \) farthest from the line containing \( s \).

Then, the farthest of the three is the farthest point of \( P \) from \( s \). Since there are \( O(n^2) \) candidate line segments, in order to obtain an \( O(n^2) \)-time algorithm, we have to address each of parts I, II, and III above in constant time (on average) for each candidate line segment.

Parts I and II

The subproblem of interest associated with parts I and II can be stated as follows.

**Subproblem 3** Preprocess \( P \) into a data structure so that, given a point \( p \in P \) and a directed line \( L \) passing through \( p \), one can efficiently report the point of \( P \) farthest from \( p \) among those located to the left of \( L \).

Consider the following approach for solving Subproblem 3.

Let \( H \) denote the half-plane to the left of directed line \( L \). As line \( L \) rotates around point \( p \), a point \( q \in P \setminus \{p\} \) may enter and leave half-plane \( H \). Each of these point-entering and -leaving events can be determined in \( O(1) \) time by computing the line passing through \( p \) and each point \( q \in P \setminus \{p\} \). Given that a point \( q \in P \setminus \{p\} \) can enter and leave \( H \) only once, there exist \( 2n - 2 \) point-entering and -leaving events. These events can be obtained in counterclockwise order in \( O(n) \) time by employing the point-line duality transform – that is, mapping \( P \), using \( O(n^2) \) time and space, into an arrangement of \( n \) lines, through which the set of points \( P \setminus \{p\} \) can be determined in sorted order around \( p \) in \( O(n) \) time.

Within an interval bounded by any two consecutive events in the sorted order, the subset of points \( P \setminus \{p\} \) contained in \( H \) remains unchanged, and so does the maximum of the distances from points \( (P \cap H) \setminus \{p\} \)
to p. As we sweep the event intervals in the sorted (counterclockwise) order, we can keep track of the maximum of the distances from points \((P \cap H) \setminus \{p\}\) to p by maintaining a monotonic double-ended queue \(Q\), whose elements are a subset of \(P \setminus \{p\}\), as follows.

For a point \(q_i \in P \setminus \{p\}\), let \(d_i\) be the distance from \(q_i\) to \(p\). Let \(q_f\) denote the first element in \(Q\), and \(q_l\) be the last element of \(Q\). Upon a point-entering event, in which a point \(q_i\) enters half-plane \(H\), if \(d_i < d_f\), then \(q_i\) is added to the back of \(Q\); otherwise, we keep removing the last element of \(Q\) until the condition \(d_i < d_f\) is satisfied, and \(q_i\) is appended to the back of \(Q\). For a point-leaving event, where a point \(q_i\) leaves half-plane \(H\), if \(q_i = q_f\) (i.e., \(q_i\) is the first element in \(Q\)), then we remove \(q_f\) from the front of \(Q\); otherwise, no update is made to \(Q\) (see Figure 4 for an illustrative example).

For the first event interval in the sorted order, we simply treat each of the points \((P \cap H) \setminus \{p\}\) as a point entering \(H\), and process the points in counterclockwise order according to the rules above. Note that, within an event interval, \(Q\) always contains a subset of \((P \cap H) \setminus \{p\}\) such that the points, from the front to the back of \(Q\), form a monotonic sequence with strictly decreasing distances from \(p\). Thus, the first element of \(Q\) always corresponds to the point of \((P \cap H) \setminus \{p\}\) with the farthest distance from \(p\) for a given event interval. Since there are \(O(n)\) point-entering and -leaving events, and each point of \(P \setminus \{p\}\) can only be inserted into or removed from \(Q\) at most once, it takes a total of \(O(n)\) time to determine the point of \((P \cap H) \setminus \{p\}\) farthest from \(p\) in all event intervals.

For each point \(p \in P\), we then construct an \(O(n)\)-size data structure with a logarithmic query time to store the farthest point of \((P \cap H) \setminus \{p\}\) from \(p\) for each event interval. Given a point \(p \in P\) and a directed line \(L\) passing through \(p\), by using the associated query data structure, we can retrieve in \(O(\log n)\) time the farthest point of \(P\) from \(p\) among those situated to the left of \(L\).

**Lemma 6** In Subproblem 3, a set \(P\) of \(n\) points can be preprocessed using \(O(n^2)\) time and space so that, given a point \(p \in P\) and a directed line \(L\) passing through \(p\), one can report, in \(O(\log n)\) time, the point of \(P\) farthest from \(p\) among those located to the left of \(L\).

As with the queries required in solving the discrete median line segment problem (Section 3.2), for any point \(p \in P\), a query line \(L\) passing through \(p\) must be perpendicular to the line passing through \(p\) and another point \(q \in P\). Thus, we can pre-compute and index these \(O(n)\) query lines, and use the indices to create a perfect hash map [3, Chapter 11] for \(O(1)\)-time searches instead of constructing the logarithmic query-time data structure above for each point \(p \in P\).

Ergo, with respect to parts I and II of our current approach, it takes \(O(n^2)\) time total to find i) the point of \([P] \cap H_a\) farthest from \(a\), and ii) the point of \([P] \cap H_b\) farthest from \(b\), for all \(O(n^2)\) candidate line segments.

**Part III**

First, we note that the convex hull \([P]\) of \(P\) can be computed in \(O(n \log n)\) time [5, Chapter 11]. Let \(L\) be a line passing through any two points of \(P\), and let \(H\) denote either of the two half-planes bounded by \(L\). Since the orthogonal distances from the vertices of \([P] \cap H\) to line \(L\) are unimodal (as the vertices are traversed in order) [9, Theorem 1], we can find the farthest point of \([P]\) to line \(L\) using two binary searches in \(O(\log n)\) time. Thus, we can compute, for all lines \(L\) in \(O(n^2 \log n)\) time total, the farthest point of \([P]\) from line \(L\).

In order to achieve \(O(n^2)\) time, we propose the following approach.

We begin by using the point-line duality transform to map \(P\) into a set \(P^*\) of \(n\) lines, whose arrangement can be computed in \(O(n^2)\) time [2, 6]. Specifically, a point \(p = (x_p, y_p) \in P\) in the primal plane is transformed into a line \(p^*\) represented by \(y = x_p x - y_p\) in the dual plane. Notice that the \(x\)-coordinate \(x_p\) of point \(p\) in the primal plane is equivalent to the slope of line \(p^*\) in the dual plane.

Let \(V_u\) and \(V_l\) denote the upper and lower envelopes of \(P^*\) in the dual plane. Note that \(V_u\) and \(V_l\) correspond to the lower and upper hulls of \([P]\), respectively, in the primal plane. We can compute \(V_u\) and \(V_l\) in \(O(n \log n)\) time using \(O(n)\) space [5, Chapter 11].
For a point \( p \in P \), we can obtain the sequence of \( n-1 \) lines, each of which passes through \( p \) and a point \( q \in P \setminus \{p\} \), in increasing order of their slopes in \( O(n) \) time by simply traversing the vertices along the dual line \( p^* \). We denote the sequence of lines as \( S = \{L_1, L_2, \ldots, L_{n-1}\} \).

For each line \( L_i \) in the primal plane, we designate its dual point as \( L_i^* \). Note that the slope of line \( L_i \) in the primal plane corresponds to the \( x \)-coordinate of point \( L_i^* \) in dual plane.

In the ensuing description, for conciseness, we present the arguments only for upper envelope \( V_u \) (i.e., the lower hull of \([P]\)), and the same arguments can be similarly applied to lower envelope \( V_l \) (i.e., the upper hull of \([P]\)) due to symmetry.

First, we observe the following. For a line \( L_i \in S \), let \( \rho_i \) be the vertical upward ray emanating from \( L_i^* \) in the dual plane. The line containing the edge of \( V_u \) intersected by \( \rho_i \) corresponds to the farthest point of the lower hull of \([P]\) from \( L_i \) in the primal plane.

Consequently, we can find the farthest point in the lower hull of \([P]\) from each line \( L_i \in S \) — that is, \( L_1, L_2, \ldots, L_{n-1} \) in increasing order of their slopes — by simply traversing along \( V_u \) in the positive \( x \)-direction, while keeping track of the edge of \( V_u \) intersected by each \( \rho_i \) (see Figure 5).

![Diagram](image)

Figure 5:Traversal of upper envelope \( V_u \) in part III. The polygonal chain \( q_k, q_{k+1}, q_{k+2} \) belongs to the lower hull of \([P]\) in the primal plane. The dual lines of \( q_k, q_{k+1}, \) and \( q_{k+2} \) are denoted by \( q_k^*, q_{k+1}^*, \) and \( q_{k+2}^* \), respectively. The farthest point in the lower hull of \([P]\) from \( L_i \) in the primal plane is given by line \( q_k^* \) containing the edge of \( V_u \) bounded to the right by \( v \) in the dual plane.

Specifically, let \( u \) and \( v \) be any pair of adjacent vertices in \( V_u \), where the \( x \)-coordinate of \( u \) is smaller than that of \( v \). For every line \( L_i \in S \) whose slope lies between the \( x \)-coordinates of \( u \) and \( v \), the farthest point in the lower hull of \([P]\) from \( L_i \) is given by (the line supporting) the edge connecting \( u \) and \( v \). Thus, we only have to keep track of each vertex encountered in our traversal of \( V_u \) and process the sequence of lines \( S \) in order accordingly. That is, upon encountering vertex \( v \) in the traversal of \( V_u \), for the contiguous subsequence of lines in \( S \) whose slopes fall between the \( x \)-coordinates of \( u \) and \( v \), we record, as the farthest point in lower hull of \([P]\) from each of those lines, the point dual to the line containing the edge bounded on the right by \( v \).

Note that, for the first line \( L_1 \) in \( S \), it takes \( O(\log n) \) time (i.e., a binary search on \( V_u \)) to locate the edge intersected by \( \rho_1 \). From there on, by traversing along \( V_u \) (and \( S \) in order), for each successive line \( L_i \in S \) (in increasing order of slope), where \( 2 \leq i \leq n-1 \), it requires only a constant number of operations to determine the edge of \( V_u \) intersected by \( \rho_i \) and thus the farthest point in the lower hull of \([P]\) from \( L_i \).

Since \( S \) and \( V_u \) are bounded by \( O(n) \) in size, the entire process above takes \( O(n) \) time. In other words, for each point \( p \in P \), it takes \( O(n) \) time to determine the farthest point of the lower hull of \([P]\) from \( L_i \) for all \( i \in [1, n-1] \).

The same traversal procedure can be performed on lower envelope \( V_l \) to find the farthest point of the upper hull of \([P]\) from each line \( L_i \), where \( 1 \leq i \leq n-1 \).

Recall that \( L \) denotes any line containing two points of \( P \). Since \( |P| = n \), we conclude that, in \( O(n^2) \) time total, we can find the farthest point of \([P]\) from each of the \( O(n^2) \) lines \( L \). That is, in part III, it takes a total of \( O(n^2) \) time to find the point of \([P]\) farthest from the line containing a candidate line segment \( s \) for all \( O(n^2) \) candidates.

Finally, based on the results obtained for parts I, II, and III, we arrive at the following conclusion.

**Theorem 7** The discrete center line segment problem can be solved in \( O(n^2) \) time using \( O(n^2) \) space.

## 5 A remark on discrete median and center line segments and convex hull of points

Recall that, for any given set \( P \) of points, \([P]\) denotes the convex hull of \( P \).

**Observation 2** The discrete median line segment of \( P \) does not necessarily have an endpoint at a vertex of \([P]\).

**Proof.** It suffices to disprove the statement “if \( s \) is a discrete median line segment of \( P \), then \( s \) has an endpoint at a vertex of \([P]\)” by giving a counterexample. Consider a set of six points \( P = \{p_1, p_2, \ldots, p_6\} \) in the plane with the following coordinates (see Figure 6): \( p_1 = (0, 0), p_2 = (1, 0), p_3 = (1, 1), p_4 = (0, 1), p_5 = (x, \frac{1}{2}), \) and \( p_6 = (1 - x, \frac{1}{2}) \).
Figure 6: Illustrative example used in proving Observations 2 and 3.

It can be verified, using algebraic geometry, that $p_5p_6$ is the discrete median line segment of $P$ for any $0 < x < \frac{1}{14}$ (note that, when $\frac{1}{14} < x < \frac{1}{2}$, either $p_1p_3$ or $p_2p_4$ is the discrete median line segment of $P$). Since the vertices of $[P]$ consist of $p_1, p_2, p_3,$ and $p_4$, the discrete median line segment of $P$ (i.e., $p_5p_6$) does not have an endpoint at a vertex of $[P]$. □

**Observation 3** The discrete center line segment of $P$ does not necessarily have an endpoint at a vertex of $[P]$.

**Proof.** Using the same counterexample as in the proof of Observation 2, with $0 < x < \frac{1}{2}$, we can show that the statement “if $s$ is a discrete center line segment of $P$, then $s$ has an endpoint at a vertex of $[P]$” is false, thus proving Observation 3. □

With respect to solving the discrete median and center line segment problems, Observations 2 and 3 essentially preclude the potential of finding efficient algorithms with a time complexity dependent on the number of vertices on the convex hull of $P$, which may be significantly smaller than $n$ in practice.

6 Concluding remarks

We have described an $O(n^2)$-time algorithm for computing the discrete median line segment as well as the discrete center line segment for a set $P$ of $n$ points in the plane.

Our $O(n^2)$-time algorithm for solving the discrete median line segment problem matches in time complexity the lower bound of the medoid problem, as well as the fastest known algorithm for finding the median line in $\mathbb{R}^2$ (i.e., the line having the minimal sum of distances from $P$) [7]. Hence, we conjecture that our algorithm for the discrete median line segment problem is optimal. Nevertheless, we have no reason to believe that our $O(n^2)$ time bound is tight for the discrete center line segment problem.

By allowing each point of $P$ to be associated with a positive weight, we can generalize our problems to those of minimizing the sum and maximum of weighted distances. The algorithms proposed herein can be directly extended to solve the weighted problems with the same time and space bound.

We end our paper with the following open questions. Can we reduce the quadratic space usage of our $O(n^2)$-time algorithms to $O(n)$? Can we obtain efficient (subcubic-time) algorithms for solving the discrete median and center line segment problems in $\mathbb{R}^d$ for $d \geq 3$?

References


