Closest-Pair Queries and Minimum-Weight Queries are Equivalent for Squares[*]

Abrar Kazi[†]   Michiel Smid[‡]

May 22, 2021

Abstract

Let S be a set of n weighted points in the plane and let R be a query range in the plane. In the range closest pair problem, we want to report the closest pair in the set $R \cap S$. In the range minimum weight problem, we want to report the minimum weight of any point in the set $R \cap S$. We show that these two query problems are equivalent for query ranges that are squares, for data structures having $\Omega(\log n)$ query times. As a result, we obtain new data structures for range closest pair queries with squares.

1   Introduction

Let S be a set of n points in the plane. In the *range closest pair problem*, we want to store S in a data structure, such that for any axes-parallel query rectangle R, the closest pair in the point set $R \cap S$ can be reported. This problem has received considerable attention; see [1, 2, 3, 6, 7, 9, 10, 11, 12]. The best known result is by Xue *et al.* [12], who obtained a query time of $O(\log^2 n)$ using a data structure of size $O(n \log^2 n)$. For the special case when the query range R is a *square* (or, more generally, a fat rectangle), Bae and Smid [2] showed that a query time of $O(\log n)$ is possible, using $O(n \log n)$ space.

Assume that each point p of S has a real weight $\omega(p)$. In the *range minimum weight problem*, we want to store S in a data structure, such that for any axes-parallel query rectangle R, the minimum weight of any point in $R \cap S$ can be reported. Using a standard range tree of size $O(n \log n)$, such queries can be answered in $O(\log^2 n)$ time; see, e.g., de Berg *et al.* [5]. Chazelle [4] showed the following results for such queries on a RAM: (i) for every constant $\varepsilon > 0$, $O(\log^{1+\varepsilon} n)$ query time using $O(n)$ space, (ii) $O(\log n \log \log n)$ query time using $O(n \log \log n)$ space, and (iii) for every constant $\varepsilon > 0$, $O(\log n)$ query time using $O(n \log^\varepsilon n)$ space. We are not aware of better solutions for query squares.

1.1 Our Results

We show that the range closest pair problem and the range minimum weight problem are equivalent for query *squares*[1], for data structures having $\Omega(\log n)$ query times. We say that a function f is *smooth*, if $f(O(n)) = O(f(n))$. Our main results are as follows:

Theorem 1 *Let M and Q be smooth functions such that $M(n) \geq n$ and $Q(n) = \Omega(\log n)$. Assume there exists a data structure of size $M(n)$ that answers a range minimum weight query, for any query square, in $Q(n)$ time. Then there exists a data structure of size $O(M(n))$ that answers a range closest pair query, for any query square, in $O(Q(n))$ time.*

Theorem 2 *Let M and Q be smooth functions such that $M(n) \geq n$ and $Q(n) = \Omega(\log n)$. Assume there exists a data structure of size $M(n)$ that answers a range closest pair query, for any query square, in $Q(n)$ time. Then there exists a data structure of size $O(M(n))$ that answers a range minimum weight query, for any query square, in $O(Q(n))$ time.*

Theorem 1, together with the above mentioned results of Chazelle, imply the following:

Corollary 1 *Let S be a set of n points in the plane and let $\varepsilon > 0$ be a constant. Range closest pair queries, for any query square, can be answered*

1. *in $O(\log^{1+\varepsilon} n)$ time using $O(n)$ space,*

2. *in $O(\log n \log \log n)$ time using $O(n \log \log n)$ space,*

3. *in $O(\log n)$ time using $O(n \log^{\varepsilon} n)$ space.*

Observe that the third result in Corollary 1 improves the space bound in Bae and Smid [2] from $O(n \log n)$ to $O(n \log^{\varepsilon} n)$.

Our proofs of Theorems 1 and 2 are based on the approach of Bae and Smid [2] for range closest pair queries with squares. Their solution uses data structures for (i) deciding whether a query square contains at most c points of S, for some fixed constant c, (ii) computing the smallest square that has a query point as its bottom-left corner and contains c' points of S, for some fixed constant c', and (iii) range minimum weight queries with squares. They showed that the queries in (i) and (ii) can be answered in $O(\log n)$ time using $O(n \log n)$ space. We will improve the space bound for both these queries to $O(n)$.

If p is a point in the plane, then we denote its x- and y-coordinates by p_x and p_y, respectively. The *north-east quadrant* of p is defined as $NE(p) = [p_x, \infty) \times [p_y, \infty)$. Similarly, the *south-west quadrant* of p is defined as $SW(p) = (-\infty, p_x] \times (-\infty, p_y]$. The Manhattan distance between two points p and q is given by $d_1(p, q) = |p_x - q_x| + |p_y - q_y|$. Observe that, for $q \in NE(p)$, $d_1(p, q) = (q_x + q_y) - (p_x + p_y)$.

Definition 1 *Let S be a set of n points in the plane, let c be an integer with $1 \leq c \leq n$, and let p be a point in the plane.*

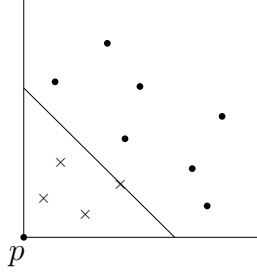[1]throughout this paper, squares are always axes-parallel

Figure 1: The set $closest_c(p)$, where $c = 4$, is the set of four points indicated by $\times$.

1. Assume that $|NE(p) \cap S| \geq c$. We define $closest_c(p)$ to be the set of c points in $NE(p) \cap S$ that are closest (with respect to d_1) to p.

2. Assume that $|NE(p) \cap S| < c$. We define $closest_c(p)$ to be $NE(p) \cap S$.

The set $closest_c(p)$ can equivalently be described as follows. Consider a line with slope -1 through p. We move this line to the right until it has encountered c points of $NE(p) \cap S$ or it has encountered all points in $NE(p) \cap S$, whichever occurs first. The set $closest_c(p)$ is the subset of $NE(p) \cap S$ that are encountered during this process. See Figure 1 for an example.

We will see in Section 3 that data structures answering the queries (i) and (ii) in the approach of Bae and Smid [2] in $O(\log n)$ time, while using $O(n)$ space, can be obtained from the following result:

Theorem 3 *Let S be a set of n points in the plane and let c be an integer with $1 \leq c \leq n$. There exists a data structure of size $O(c^2 n)$ such that for any query point p, the set $closest_c(p)$ can be computed in $O(\log n + c)$ time.*

The proof of Theorem 3 will be given in Section 2. In Section 4, we will reduce range closest pair queries with squares, to range minimum weight queries, again with squares, and the queries of Section 3. Finally, in Section 5, we will present our reduction in the other direction.

2   Answering $closest_c(p)$ Queries

In this section, we will prove Theorem 3. Throughout this section, S denotes a set of n points in the plane and c denotes an integer with $1 \leq c \leq n$. We assume for simplicity that no two points in S are (i) on a vertical line, (ii) on a horizontal line, and (iii) on a line with slope -1. We will use the notion of a staircase polygon, as illustrated in Figure 2.

Definition 2 (Staircase polygon) *A staircase polygon consists of (i) a horizontal edge AB, where A is to the left of B, (ii) a vertical edge CB where C is below B, and (iii) a polygonal path consisting of alternating vertical and horizontal edges, where the leftmost edge is vertical with top endpoint A and the rightmost edge is horizontal with right endpoint C.*
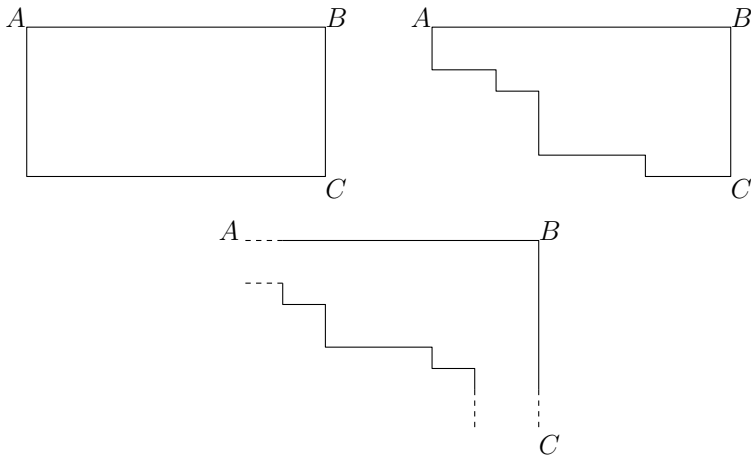
3

Figure 2: Three examples of staircase polygons. In the two examples on top, all points A, B, and C have finite coordinates. In the example at the bottom, B has finite coordinates, A has an x-coordinate of $-\infty$, and C has a y-coordinate of $-\infty$.
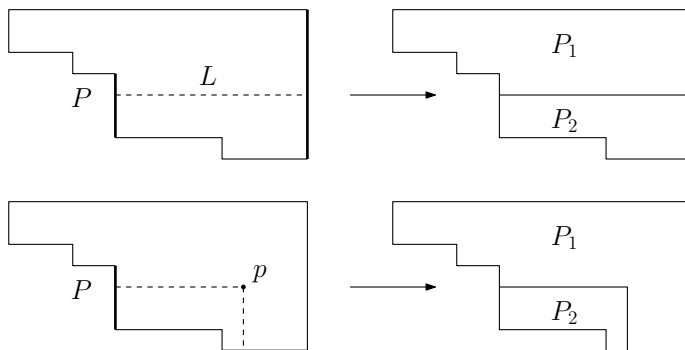

Figure 3: The top part illustrates Observation 1.1. The bottom part illustrates Observation 1.2. Each thick edge is divided into two new edges.

In the two staircase polygons at the top of Figure 2, the vertices A, B, and C have finite x- and y-coordinates. In the staircase polygon at the bottom of this figure, the vertex A can be thought of having an x-coordinate of $-\infty$ and the same y-coordinate as B, and the left-most edge as being infinitely far off to the left. Similarly, the vertex C has a y-coordinate of $-\infty$ and its x-coordinate is the same as that of B; the bottom-most edge is infinitely far off in the downward direction. The vertex B may have x- and y- coordinates of ∞. In particular, the entire plane is considered a staircase polygon.

The following observation is illustrated in Figure 3.

Observation 1 *Let P be a staircase polygon.*

    1. If L is a horizontal or vertical line that intersects P, then L divides P into two staircase polygons, P_1 and P_2. The total number of edges of P_1 and P_2 (counting shared edges

<div align="center">4</div>

only once) is at most 3 *more than the number of edges belonging to* P.

2. *Let* p *be a point in the interior of* P. *The boundary of* $SW(p)$ *divides* P *into two staircase polygons,* P_1 *and* P_2. *The total number of edges of* P_1 *and* P_2 *(counting shared edges only once) is at most* 4 *more than the number of edges belonging to* P.

2.1 Constructing the Data Structure

We order the points p in S by their $p_x + p_y$ values and use $p^{(k)}$ to denote the kth point in this ordering. For any real number α, all points p in the plane for which $p_x + p_y = \alpha$ are on the line ℓ_α having equation $y = -x + \alpha$. Increasing the value of α is equivalent to moving the lint ℓ_α to the right. Thus, the above order of the points in S is the same order in which the points of S are visited when moving a line with slope -1 from left to right.

For each $k = 0, 1, \ldots, n$, we will construct a subdivision $SD^{(k)}$ of the plane into staircase polygons. We will refer to each such polygon as a *cell*.

The subdivision $SD^{(0)}$ consists of one single cell, the plane itself. Let k be an integer with $0 \le k < n$ and consider the subdivision $SD^{(k-1)}$. To obtain the next subdivision $SD^{(k)}$, we add the point $p^{(k)}$ to $SD^{(k-1)}$: from the point $p^{(k)}$, we extend a ray horizontally to the left until it has encountered c vertical edges of $SD^{(k-1)}$ or reaches $-\infty$, whichever occurs first. (For example, in Figure 4, for the case when $k = 5$ and $c = 2$, these edges are thick.) For $i = 1, \ldots, c - 1$, the part of the ray between the ith and $(i + 1)^{th}$ vertical edges divides a cell of $SD^{(k-1)}$ into two cells. We also extend a ray from $p^{(k)}$ vertically downward until it has encountered c horizontal edges of $SD^{(k-1)}$ or reaches $-\infty$, whichever occurs first. (For example, in Figure 4, for the case when $k = 6$ and $c = 2$, these edges are thick.) For $i = 1, \ldots, c - 1$, the part of the ray between the ith and $(i + 1)^{th}$ horizontal edges divides a cell of $SD^{(k-1)}$ into two cells. Finally, the boundary of $SW(p^{(k)})$ divides the cell of $SD^{(k-1)}$ that contains $p^{(k)}$ into two cells. The resulting subdivision is $SD^{(k)}$. The entire construction is illustrated in Figure 4.

The following lemma follows, by induction on k, from Observation 1.

Lemma 1 *For every* k *with* $0 \le k \le n$, *every cell of the subdivision* $SD^{(k)}$ *is a staircase polygon.*

Consider the final subdivision $SD^{(n)}$. With each cell C of this subdivision, we store the set $S_c(C) := closest_c(z)$, where z is the top-right vertex of C. Finally, we build a point location data structure for the subdivision $SD^{(n)}$; see Kirkpatrick [8]. This completes the description of the data structure.

Definition 3 *Let* C *be a cell in* $SD^{(k)}$. *The* northeast closure *of* C, $NEC(C)$, *consists of its interior, the topmost edge of* C *(without its leftmost point), and the rightmost edge of* C *(without its lowest point). See Figure 5 for an example.*

For the query algorithm, consider a query point p. We first locate p in the subdivision $SD^{(n)}$, and find the (unique) cell C such that $p \in NEC(C)$. The query algorithm returns the set $S_c(C)$.
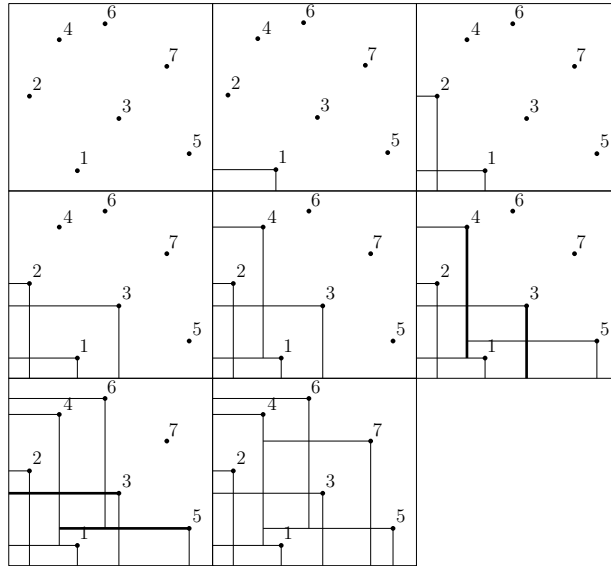
5

Figure 4: Constructing the sequence of subdivisions for $n = 7$ and $c = 2$.

Figure 5: Illustrating the northeast closure of a cell. It includes the shaded grey region and the thick edges, but not the thin edges or the circled vertices.

The following lemma proves the correctness of this query algorithm.

Lemma 2 *For any query point p in the plane, let C be the cell of $SD^{(n)}$ that is returned by the point location query. Then $S_c(C) = closest_c(p)$.*

Proof. Let z be the top-right vertex of C. We have to show that $closest_c(p) = closest_c(z)$. Consider the triangle T that is bounded by the boundary of $NE(p)$ and the line through z with slope -1. It suffices to show that the interior of T does not contain any point of S. Assume this interior does contain a point of S. Let q be such a point that is closest (with respect to d_1) to p, and let k be such that $q = p^{(k)}$. Then, after the subdivision $SD^{(k)}$ has been constructed, the points p and z are in different cells. As a result, in the final subdivision $SD^{(n)}$, the points p and z are still in different cells. This is a contradiction. ∎

2.2 Space Requirement and Query Time

We start by bounding the number of cells of the final subdivision $SD^{(n)}$. Clearly, $SD^{(0)}$ consists of only one cell. For each k, during the construction of the subdivision $SD^{(k)}$ from $SD^{(k-1)}$, at most $2c - 1$ cells are divided into two new cells and, thus, the total number of cells increases by at most $2c - 1$. It follows that the number of cells in $SD^{(n)}$ is at most $1 + n(2c - 1) = O(cn)$.

Each cell C of $SD^{(n)}$ stores a set $S_c(C)$ of size at most c. Therefore, the total size of all these sets $S_c(C)$ is $O(c^2 n)$.

Next, we bound the number of edges of $SD^{(n)}$. The initial subdivision $DS^{(0)}$ is the entire plane, which we regard to be an infinite rectangle consisting of four edges. By Lemma 1, each cell in each subdivision $SD^{(k)}$ is a staircase polygon. Thus, by Observation 1, at most 4 new edges are added when such a cell is divided. Therefore, the number of edges increases by at most $4(2c - 1)$ when constructing $SD^{(k)}$ from $SD^{(k-1)}$. Thus, the total number of edges in the final subdivision $SD^{(n)}$ is at most $4 + n \cdot 4(2c - 1) = O(cn)$. It follows that the point location data structure uses $O(cn)$ space.

We have shown that the space used by the entire data structure is $O(c^2 n)$.

The query algorithm, with query point p, first performs point location, which takes $O(\log(cn)) = O(\log n)$ time since $c \leq n$. Reporting the set $closest_c(p)$ takes $O(c)$ time. Thus, the total query time is $O(\log n + c)$.

This completes the proof of Theorem 3.

3 Some Related Queries

In this section, we use the data structure of Theorem 3 to solve several related query problems.

Definition 4 *Let p be a point in the plane and consider the line with slope 1 through p. This line divides $NE(p)$ into two cones, each one having an angle of $45°$. We denote the upper cone by $NNE(p)$ and the lower cone by $ENE(p)$. See Figure 6.*
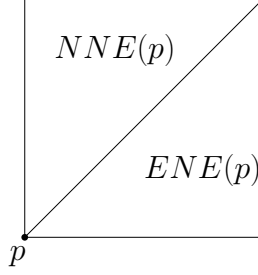
Figure 6: The line with slope 1 through p divides $NE(p)$ into $NNE(p)$ and $ENE(p)$.
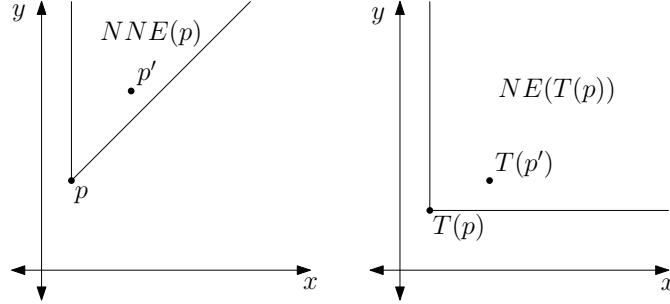

Figure 7: T transforms $NNE(p)$ into $NE(T(p))$.

Lemma 3 *Let S be a set of n points in the plane and let c be an integer with $1 \leq c \leq n$. There exists a data structure of size $O(c^2 n)$ which can perform the following query in $O(\log n + c)$ time: Given a query point p, find the smallest square that has p as its bottom-left corner and contains c points of S.*

Proof. Assume we know the set L_1 consisting of the c lowest points of $NNE(p) \cap S$ and the set L_2 consisting of the c leftmost points of $ENE(p) \cap S$. Then we obtain the answer to the query in $O(c)$ time by selecting the cth smallest element in the sequence $d_\infty(p, q)$, $q \in L_1 \cup L_2$, where $d_\infty(p, q) = \max\{|p_x - q_x|, |p_y - q_y|\}$.

We will describe how the data structure of Theorem 3 can be used to find the set L_1 in $O(\log n + c)$ time. Finding the set L_2 can be done in a symmetric way.

Consider the transformation T that maps any point $q = (q_x, q_y)$ to the point $T(q) = (q_x, q_y - q_x)$. We compute the set $S' = \{T(q) : q \in S\}$ and construct the data structure of Theorem 3 for S'.

Observe that $p' \in NNE(p)$ if, and only if, $T(p') \in NE(T(p))$; refer to Figure 7. Furthermore, if $p' \in NNE(p)$, then $d_1(T(p), T(p')) = d_1((p_x, p_y - p_x), (p'_x, p'_y - p'_x)) = (p'_x + (p'_y - p'_x)) - (p_x + (p_y - p_x)) = p'_y - p_y$. Thus, p' is one of the c lowest points in $NNE(p) \cap S$ if and only if $T(p')$ is one of the c points in $NE(T(p)) \cap S'$ that is closest (with respect to d_1) to $T(p)$.

Thus, for a given query point p, by querying the data structure for S' with $T(p)$, we obtain the set L_1. By Theorem 3, the amount of space used is $O(c^2 n)$ and the query time is $O(\log n + c)$. ∎

8

Lemma 4 *Let S be a set of n points in the plane and let c be an integer with $0 \leq c \leq n-1$. There exists a data structure of size $O(c^2 n)$ which can perform the following query in $O(\log n + c)$ time: Given a query square R, decide whether $|R \cap S| \leq c$, and if so, report the points of $R \cap S$.*

Proof. We store the set S in the data structure of Lemma 3, with c replaced by $c + 1$.

Let p be the bottom-left corner of the query square R. By querying the data structure, we obtain the smallest square R' that has p as its bottom-left corner and contains $c + 1$ points of S. It is clear that one of these $c + 1$ points is on the top or right edge of R'; let this point be p'.

If $p' \notin R$ then R is properly contained in R' and, thus, $|R \cap S| \leq c$. In this case, since $R \cap S \subset (R' \cap S)$, the points of $R \cap S$ can be reported in $O(c)$ time.

If $p' \in R$ then $|R \cap S| > c$. This fact is reported. ∎

4   From Minimum Weight Queries to Closest-Pair Queries

In this section, we prove Theorem 1. Let S be a set of n points in the plane.

We assume that, for any set V of m weighted points in the plane, we can construct a data structure $DS_{MW}(V)$ that can report, for any query square R, the minimum weight of any point in $R \cap V$. We denote the space and query time of this data structure by $M(m)$ and $Q(m)$, respectively. We assume that both functions M and Q are smooth, $M(m) \geq m$, and $Q(m) = \Omega(\log m)$.

We will show that DS_{MW} and the results from the previous sections can be used to obtain a data structure that supports range closest pair queries on S for ranges that are squares.

Let R be a query square and let ℓ be the length of its sides. Bae and Smid [2] have shown that the closest pair in $R \cap S$ is obtained by performing the following six steps.

Step 1: Decide whether $|R \cap S| \leq 9$.[2] If this is the case, find the points in $R \cap S$, compute and return the closest-pair distance in this set, and terminate the query algorithm. Otherwise, i.e., if $|R \cap S| \geq 10$, proceed with Step 2.

- We implement this step by storing the points of S in the data structure of Lemma 4, where $c = 9$. This uses $O(n)$ space and supports Step 1 in $O(\log n)$ time.

- Assume that $|R \cap S| \geq 10$. By dividing R into 9 subsquares with sides of length $\ell/3$, the Pigeonhole Principle implies that the closest-pair distance in $R \cap S$ is at most $\sqrt{2} \cdot \ell/3$, which is less than $\ell/2$.

Step 2: Write R as the Cartesian product $[a_x, b_x] \times [a_y, b_y]$; observe that $\ell = b_x - a_x = b_y - a_y$. Compute the following four squares:

---

[2]In [2], the value 16 is used instead of 9.

1. The smallest square that has (a_x, a_y) as its bottom-left corner and contains at least 5 points of S.

2. The smallest square that has (b_x, a_y) as its bottom-right corner and contains at least 5 points of S.

3. The smallest square that has (b_x, b_y) as its top-right corner and contains at least 5 points of S.

4. The smallest square that has (a_x, b_y) as its top-left corner and contains at least 5 points of S.

Let ℓ' be the side length of the smallest of these four squares. If $\ell' > \ell/2$, set $\delta = \ell/2$. Otherwise, set $\delta = \ell'$.

- We implement Item 1 of this step by storing the points of S in the data structure of Lemma 3, where $c = 5$. This uses $O(n)$ space and supports this part of Step 2 in $O(\log n)$ time.

- We implement Items 2, 3, and 4 of Step 2 by storing the points of S in a symmetric variant of the data structure of Lemma 3, again with $c = 5$.

Step 3: Consider the value δ obtained in Step 2. Observe that $0 < \delta \leq \ell/2$. Partition the square R into (i) the squares $C_1, C_2, C_3,$ and C_4 with sides of length δ, and (ii) the rectangles $A_1, A_2, \ldots, A_5$, as indicated in Figure 8. Define

$$
\begin{aligned}
B_1 &= C_3 \cup A_2 \cup A_3 \cup A_5, \\
B_2 &= C_4 \cup A_3 \cup A_4 \cup A_5, \\
B_3 &= C_1 \cup A_1 \cup A_3 \cup A_4, \\
B_4 &= C_2 \cup A_1 \cup A_2 \cup A_3.
\end{aligned}
$$

Observe that $B_1, B_2, B_3,$ and B_4 are squares with sides of length $\ell - \delta$.

Clearly, this step of the query algorithm takes $O(1)$ time.

Step 4: For each $k = 1, 2, 3, 4$, find the points of the set $C_k \cap S$ and compute the closest-pair distance w_k in this set; if $|C_k \cap S| \leq 1$, then we set $w_k = \infty$. Compute the value $\delta_1 = \min\{w_k : 1 \leq k \leq 4\}$.

- Since each C_k is a square containing at most 5 points of S, we implement this step by storing the points of S in the data structure of Lemma 4, where $c = 5$. This uses $O(n)$ space and supports Step 4 in $O(\log n)$ time.

Step 5: During preprocessing, we compute four (possibly overlapping) subsets $S_1, \ldots, S_4$ of S: For any point $p = (p_x, p_y)$ in the plane, define its four quadrants by

$$
\begin{aligned}
Q_1(p) &= [p_x, \infty) \times [p_y, \infty), \\
Q_2(p) &= (-\infty, p_x] \times [p_y, \infty), \\
Q_3(p) &= (-\infty, p_x] \times (-\infty, p_y], \\
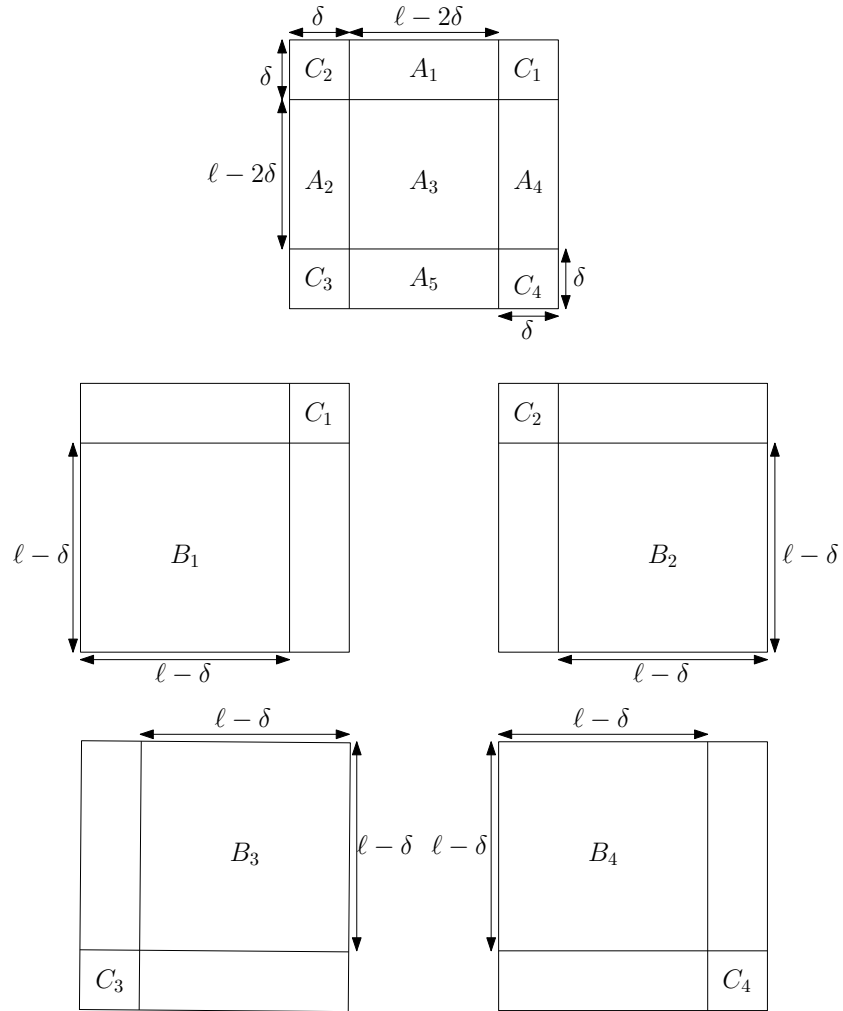Q_4(p) &= [p_x, \infty) \times (-\infty, p_y].
\end{aligned}
$$

10

Figure 8: On the top, the partition of the query square R into $C_1, \ldots, C_4$ and $A_1, \ldots, A_5$ is shown. The parts at the bottom illustrate $B_1, \ldots, B_4$.

For each $k = 1, 2, 3, 4$ and each point p of S, if $Q_k(p) \cap (S \setminus \{p\}) \neq \emptyset$, then we add p to the subset S_k. We give p (as an element of S_k) a weight which is equal to the distance between p and its nearest neighbor in $Q_k(p) \cap (S \setminus \{p\})$. Note that if p is in S_k, then p is *not* a maximal point of S under vector dominance with respect to the k-th quadrant. Moreover, note that these weights are the lengths of the edges in the *Yao-graph* that uses four cones of angle $\pi/2$; see Yao [13].

In this fifth step of the query algorithm, we find, for each $k = 1, 2, 3, 4$, the minimum weight of any point in $B_k \cap S_k$. If this minimum weight is less than δ, then we set w'_k to this minimum weight; otherwise, we set $w'_k = \infty$. Finally, we compute the value $\delta_2 = \min\{w'_k : 1 \leq k \leq 4\}$.

- We implement this step by storing, for each $k = 1, \ldots, 4$, the weighted point set S_k in the data structure $DS_{MW}(S_k)$. Since S_k has size at most n and since B_k is a square, this uses $O(M(n))$ space and supports Step 5 in $O(Q(n))$ time.

Step 6: In this last step of the query algorithm, we return the minimum of δ_1 and δ_2. Clearly, this takes $O(1)$ time.

For the correctness of this query algorithm, we refer the reader to Bae and Smid [2]. The total amount of space used is $O(M(n) + n) = O(M(n))$ and the total query time is $O(Q(n) + \log n) = O(Q(n))$. This proves Theorem 1.

5 From Closest-Pair Queries to Minimum Weight Queries

In this final section, we prove Theorem 2. Let S be a set of n weighted points in the plane. For each point p in S, we denote its weight by $\omega(p)$.

We assume that, for any set V of m points in the plane, we can construct a data structure $DS_{CP}(V)$ that can report, for any query square R, the closest pair in $R \cap V$. We denote the space and query time of this data structure by $M(m)$ and $Q(m)$, respectively. We assume that both functions M and Q are smooth, $M(m) \geq m$, and $Q(m) = \Omega(\log m)$.

We will show that DS_{CP} and the data structure of Lemma 4 can be used to obtain a data structure that supports range minimum weight queries on S for ranges that are squares.

We may assume, without loss of generality, that all weights $\omega(p)$ are positive, pairwise distinct, and strictly less than 1. (If this is not the case, then we sort the sequence of weights, breaking ties arbitrarily, and replace each weight by $1/(2n)$ times its position in the sorted order.)

Let δ be the closest pair distance in the set S. For each point p in S, define the points

$$p^+ = (p_x + \delta \cdot \omega(p)/3, p_y)$$

and

$$p^- = (p_x - \delta \cdot \omega(p)/3, p_y),$$

and let $S' = \{p^+ : p \in S\} \cup \{p^- : p \in S\}$.

Our data structure for minimum weight queries consists of the following:

1. We store the points of S in the data structure of Lemma 4, where $c = 1$.

2. We store the points of $S \cup S'$ in the data structure $DS_{CP}(S \cup S')$.

The query algorithm is as follows. Let R be a query square. First, we decide whether $|R \cap S| \leq 1$. If this is the case, then we obtain the set $R \cap S$. If this set contains one point, say p, then we return $\omega(p)$; otherwise, we return the fact that $R \cap S$ is empty.

Assume that $|R \cap S| \geq 2$. Then we query $DS_{CP}(S \cup S')$ for the closest pair in $R \cap (S \cup S')$. Let (p, a) be this closest pair. In Lemma 7, we will prove that $p \in R \cap S$ and $a \in R \cap \{p^+, p^-\}$. We return $\omega(p)$.

Since $|S| = n$ and $|S'| = 2n$, the total amount of space used by the data structure is $O(n) + M(3n) = O(M(n))$ and the total query time is $O(\log n) + Q(3n) = O(Q(n))$.

To complete the proof of Theorem 2, it remains to prove the correctness of the query algorithm. We will present this proof in the next subsection.

5.1   Correctness of the Query Algorithm

We denote the Euclidean distance between two points a and b by $d(a, b)$. We start with two preliminary lemmas.

Lemma 5 *Let R be a square such that $|R \cap S| \geq 2$. Then for each point p in $R \cap S$, at least one of the points p^+ and p^- is in R.*

Proof. Let ℓ be the side length of R. The distance between any two distinct points of $R \cap S$ is at least δ and at most $\ell \cdot \sqrt{2}$. It follows that $\delta \leq \ell \cdot \sqrt{2}$.

Let p be an arbitrary point in $R \cap S$. We may assume, without loss of generality, that p is in the left half of R, i.e., the distance between p and the right boundary of R is at least $\ell/2$. Since $\omega(p) < 1$,
$$d(p, p^+) = \delta \cdot \omega(p)/3 < \delta/3 < \ell/2$$
and, thus, the point p^+ is in R. ■

Lemma 6 *Let p and q be two distinct points in S, and let $a \in \{p^+, p^-\}$ and $b \in \{q^+, q^-\}$. Then the following inequalities hold:*

1. *Both $d(p, a)$ and $d(q, b)$ are less than $\delta/3$.*

2. *$d(p, q) \geq \delta$.*

3. *Both $d(p, b)$ and $d(a, q)$ are larger than $2\delta/3$.*

4. *$d(a, b) > \delta/3$.*

13

Proof. Recall that the weights of all points in S are less than 1. Since $d(p, a) = \delta \cdot \omega(p)/3 < \delta/3$ and $d(q, b) = \delta \cdot \omega(q)/3 < \delta/3$, the first claim holds. The second claim follows from the definition of δ. The third claim holds because

$$\delta \leq d(p, q) \leq d(p, b) + d(b, q) < d(p, b) + \delta/3$$

and

$$\delta \leq d(p, q) \leq d(p, a) + d(a, q) < \delta/3 + d(a, q).$$

The fourth claim holds because

$$\delta \leq d(p, q) \leq d(p, a) + d(a, b) + d(b, q) < \delta/3 + d(a, b) + \delta/3.$$

∎

The next lemma states that the output of the query in $DS_{CP}(S \cup S')$ consists of one point p in S and one point in $\{p^+, p^-\}$.

Lemma 7 *Let R be a square such that $|R \cap S| \geq 2$. The closest pair distance in $R \cap (S \cup S')$ is attained by a pair (p, a), for some $p \in R \cap S$ and $a \in R \cap \{p^+, p^-\}$.*

Proof. We consider the three possible cases, depending on whether the closest pair distance in $R \cap (S \cup S')$ is attained by two points of S (Case 1), two points of S' (Case 2), or one point of S and one point of S' (Case 3). As we will see, neither of the first two cases can happen.

Case 1: The closest pair distance in $R \cap (S \cup S')$ is attained by a pair (p, q), where p and q are distinct points in $R \cap S$.

By Lemma 5, there exist points $a \in \{p^+, p^-\}$ and $b \in \{q^+, q^-\}$, such that both a and b are in R. Therefore, the closest pair distance in $R \cap (S \cup S')$ is at most the closest pair distance in $\{p, q, a, b\}$, which, by Lemma 6, is less than $d(p, q)$. This is a contradiction. Thus, this case cannot happen.

Case 2: The closest pair distance in $R \cap (S \cup S')$ is attained by a pair (a, b), where a and b are distinct points in $R \cap S'$.

Let p and q be the points in S such that $a \in \{p^+, p^-\}$ and $b \in \{q^+, q^-\}$. Note that p or q may be outside R.

First assume that $p = q$. Then, $\{a, b\} = \{p^+, p^-\}$ and, thus, $p \in R$. But then $d(p, a) < d(a, b)$, which is a contradiction.

Thus, $p \neq q$. By Lemma 6, $d(a, b) > \delta/3$. Let r be the point in $R \cap S$ whose weight is minimum. By Lemma 5, there exists a point $c \in \{r^+, r^-\}$, such that c is in R, and, by Lemma 6, $d(r, c) < \delta/3$. It follows that $d(r, c) < d(a, b)$, which is a contradiction. Thus, Case 2 cannot happen.

Case 3: The closest pair distance in $R \cap (S \cup S')$ is attained by a pair (a, q), where a is a point in $R \cap S'$ and q is a point in $R \cap S$.

Let p be the point in S such that $a \in \{p^+, p^-\}$. The claim in the lemma follows if we can show that $p = q$.

Assume that $p \neq q$. By Lemma 5, there exists a point $b \in \{q^+, q^-\}$, such that b is in R. We obtain a contradiction, because, by Lemma 6, $d(q, b) < \delta/3$ and $d(a, q) > 2\delta/3$. ■

The next lemma will complete the correctness proof of our query algorithm.

Lemma 8 *Let R be a square such that $|R \cap S| \geq 2$. Let p be a point in $R \cap S$ and let a be a point in $\{p^+, p^-\}$, such that the closest pair distance in $R \cap (S \cup S')$ is attained by (p, a). (By Lemma 7, p and a exist.) Then the minimum weight of any point in $R \cap S$ is equal to $\omega(p)$.*

Proof. Let q be the point in $R \cap S$ whose weight is minimum. By Lemma 5, there exists a point $b \in \{q^+, q^-\}$, such that b is in R. If $q \neq p$, then

$$d(q, b) = \delta \cdot \omega(q)/3 < \delta \cdot \omega(p)/3 = d(p, a),$$

which is a contradiction. Thus, $q = p$. ■

References

[1] M. A. Abam, P. Carmi, M. Farshi, and M. Smid. On the power of the semi-separated pair decomposition. *Computational Geometry: Theory and Applications*, 46:631–639, 2013.

[2] S. W. Bae and M. Smid. Closest-pair queries in fat rectangles. *Computational Geometry: Theory and Applications*, 83:1–8, 2019.

[3] T. M. Chan, S. Rahul, and J. Xue. Range closest-pair search in higher dimensions. In *Proceedings of the 16th Algorithms and Data Structures Symposium*, volume 11646 of *Lecture Notes in Computer Science*, pages 269–282. Springer, 2019.

[4] B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17:427–462, 1988.

[5] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 3rd edition, 2008.

[6] P. Gupta. Algorithms for range-aggregate query problems involving geometric aggregation operations. In *Proceedings of the 16th Annual International Symposium on Algorithms and Computation*, volume 3827 of *Lecture Notes in Computer Science*, pages 892–901. Springer, 2005.

[7] P. Gupta, R. Janardan, Y. Kumar, and M. Smid. Data structures for range-aggregate extent queries. *Computational Geometry: Theory and Applications*, 47:329–347, 2014.

[8] D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12:28–35, 1983.

[9] J. Shan, D. Zhang, and B. Salzberg. On spatial-range closest-pair query. In *Proceedings of the 8th International Symposium on Spatial and Temporal Databases*, volume 2750 of *Lecture Notes in Computer Science*, pages 252–269. Springer, 2003.

[10] R. Sharathkumar and P. Gupta. Range-aggregate proximity queries. Technical Report IIIT/TR/2007/80, International Institute of Information Technology Hyderabad, 2007.

[11] J. Xue, Y. Li, and R. Janardan. Approximate range closest-pair search. In *Proceedings of the 30th Canadian Conference on Computational Geometry*, pages 282–287, 2018.

[12] J. Xue, Y. Li, S. Rahul, and R. Janardan. New bounds for range closest-pair problems. In *Proceedings of the 34th International Symposium on Computational Geometry*, volume 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 73:1–73:14, 2018.

[13] A. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11:721–736, 1982.