Data Structures for Range-Aggregate Extent Queries[☆]

Prosenjit Gupta[a,1], Ravi Janardan[*,b,2], Yokesh Kumar[b,2], Michiel Smid[c,3]

[a] *Mentor Graphics, Hyderabad 5000082, India and International Institute of Information Technology, Gachibowli, Hyderabad 500032, India.*

[b] *Department of Computer Science & Engineering, University of Minnesota, Minneapolis, MN 55455, U.S.A.*

[c] *School of Computer Science, Carleton University, Ottawa, Ontario, K1S 5B6, Canada.*

Abstract

A fundamental and well-studied problem in computational geometry is *range searching*, where the goal is to preprocess a set, S, of geometric objects (e.g., points in the plane) so that the subset $S' \subseteq S$ that is contained in a query range (e.g., an axes-parallel rectangle) can be reported efficiently. However, in many situations, what is of interest is to generate a more informative "summary" of the output, obtained by applying a suitable *aggregation function* on S'. Examples of such aggregation functions include *count*, *sum*, *min*, *max*, *mean*, *median*, *mode*, and *top-k* that are usually computed on a set of weights defined suitably on the objects. Such *range-aggregate* query problems have been the subject of much recent research in both the database and the computational geometry communities.

In this paper, we further generalize this line of work by considering aggregation functions on point-sets that measure the extent or "spread" of the objects in the retrieved set S'. The functions considered here include *closest*

[☆] A preliminary version of this paper appeared in the Proceedings of the 20th Canadian Conference on Computational Geometry, Montreal, Aug. 13–15, 2008, pp. 7–10.

[*] Corresponding author

Email addresses: prosenjit_gupta@acm.org (Prosenjit Gupta), janardan@cs.umn.edu (Ravi Janardan), kumaryo@cs.umn.edu (Yokesh Kumar), michiel@scs.carleton.ca (Michiel Smid)

[1] Research supported in part by grants SR/S3/EECE/22/2004 and DST/INT/US/NSF-RPO-0155/04 from the Department of Science and Technology, Government of India.

[2] Research supported, in part, by the National Science Foundation under grants INT–0422775 and CCF-0514950.

[3] Research supported by NSERC.

pair, *diameter*, and *width*. The challenge here is that these aggregation functions (unlike, say, *count*) are not efficiently decomposable in the sense that the answer to S' cannot be inferred easily from answers to subsets that induce a partition of S'. Nevertheless, we have been able to obtain space- and query-time-efficient solutions to several such problems including: closest pair queries with axes-parallel rectangles on point sets in the plane and on random point-sets in $\mathbb{R}^d$ ($d \geq 2$), closest pair queries with disks on random point-sets in the plane, diameter queries on point-sets in the plane, and guaranteed-quality approximations for diameter and width queries in the plane. Our results are based on a combination of geometric techniques, including multilevel range trees, Voronoi Diagrams, Euclidean Minimum Spanning Trees, sparse representations of candidate outputs, and proofs of (expected) upper bounds on the sizes of such representations.

Key words: computational geometry, data structures, closest pair, diameter, width

---

1. Introduction

Range searching is an important and well-studied class of problems in computational geometry. In a typical instance of this problem, called *range reporting*, we are given a set, S, of geometric objects (say, points in the plane) that we wish to preprocess into a data structure, so that given any query object Q (say, an axes-parallel rectangle), the subset $S' \subseteq S$ that is contained in Q can be reported efficiently. (Thus, $S' = S \cap Q$.) The paper by Agarwal and Erickson [2] provides a comprehensive survey of geometric range searching.

There are situations where it is not sufficient to merely report the objects of S'; instead, what is desired is a more informative "summary" of the output, such as an order-statistic on S'. For instance, a realtor would be interested in knowing the average or the median price of homes (the "objects") in different neighborhoods (the "queries") of a large city. This can be accomplished by applying a suitable function, called an *aggregation function*, on S'. Examples of aggregation functions include *count*, *sum*, *min*, *max*, *mean*, *median*, *mode*, and *top-k* that are usually computed on a set of weights defined suitably on the objects (in the above example, the weights are house prices). Such *range-aggregate* query problems have been the subject of much recent research in both the database and the computational geometry communities; see, for

instance, [4, 9, 12, 16, 22, 24, 25, 26, 27].

In this paper, we make further contributions to range-aggregate query processing by considering aggregation functions that measure the extent or "spread" of the objects in S'. These functions include *closest pair*, *diameter* (or *farthest pair*), and *width*. Extent measures find applications in collision detection, shape-fitting, clustering etc. [3]. Often, instead of computing the measure on the entire set—which can be both expensive and unnecessary—it is more useful to "zoom in" on a region of interest that is specified by a query range and compute quickly the desired measure only for this region. (For example, given the instantaneous positions of all aircraft over a busy airport, it is important that an air-traffic controller be able to determine rapidly the closest pair within any prescribed region of airspace, in order to identify potential collisions.)

A major challenge in working with these aggregation functions is that they are not decomposable efficiently, in the sense that the answer for S', under one of these functions, cannot be inferred quickly from answers for subsets that form a partition of S'. (For instance, given a partition of S' into sets S'_1 and S'_2, the closest pair in S' cannot be inferred in sublinear time from the closest pair information for S'_1 and S'_2.) Despite this, however, we have been able to obtain space- and query-time-efficient solutions (either exact or approximate solutions with guaranteed error bounds) to several range-aggregate extent queries, as summarized in Table 1. Our results are based on a combination of techniques, including the use of multilevel range trees, Voronoi Diagrams, Euclidean Minimum Spanning Trees, generating sparse representations of candidate output sets, and establishing proofs of (expected) upper bounds on the sizes of such representations.

Shan *et al.* [19] were among the first to consider the range-aggregate closest pair problem. For axes-parallel query rectangles, they gave a solution based on R-trees and showed that this performed well in practice; however, they did not provide a theoretical analysis of their method. Gupta [10] obtained a solution to this problem in one-dimension (resp., two-dimensions), again for axes-parallel query ranges, where the query time was $O(1)$ using $O(n)$ space (resp., $O(\log^3 n)$ query time using $O(n^2 \log^3 n)$ space). The two-dimensional result was improved recently by Sharathkumar and Gupta [21] to $O(\log^3 n)$ query time using $O(n \log^3 n)$ space. In [20, 21], they also considered a variant (motivated by applications in VLSI design rule checking [23]), where the goal is to determine if the closest pair in an axes-parallel query rectangle is within a user-specified tolerance; their approach answered queries in

$O(\log^2 n)$ time using $O(n \log^{2+\epsilon} n)$ space, for any constant $\epsilon > 0$. To the best of our knowledge, there has been no previous work on the range-aggregate versions of the diameter or width problems.

The rest of the paper is organized as follows. In Section 2 we describe our solution to the range-aggregate closest pair problem in the plane for axes-parallel query rectangles. In Section 3 we discuss an efficient and practical solution for the same problem, with the points being chosen independently and uniformly at random in the unit-square. We also discuss how this approach can be extended to higher dimensions. Section 4 gives a solution to the planar range-aggregate closest pair problem for query disks. Section 5 considers a variation of the range-aggregate closest pair problem, where one point of the closest pair is required to be inside the query object and the other is required to be outside. A solution is given in two and higher dimensions for axes-parallel query hyper-rectangles and it is shown how this method also extends to certain other types of queries, such as, for instance, query halfspaces and query balls. In Section 6, the planar range-aggregate farthest pair (i.e., diameter) problem is considered and an exact solution is given that provides a trade-off between storage and query time. Section 7 describes a different and simpler approach to the diameter problem, which computes an approximation to the diameter with a guaranteed error bound. Section 8 gives an approximate solution, again with a guaranteed error bound, for the planar range-aggregate width problem. Section 9 offers concluding remarks and directions for further work.

2. Computing the closest pair in a query rectangle

Let S be a set of n points in the plane. We will show how to preprocess the points of S into a data structure such that queries of the following form can be answered: Given an axes-parallel rectangle Q, report the closest pair in $S \cap Q$. We will solve this problem using a multi-level tree structure, where higher levels are used to solve simpler variants of this query problem. We will describe this structure in a bottom-up fashion. Thus, we start with the simplest version of the query problem, and then successively solve more general versions of the problem.

For ease of exposition, we assume in this section that no two points of S are on a horizontal line, no two points of S are on a vertical line, and all $\binom{n}{2}$ distances defined by the pairs of points in S are distinct. We denote the Euclidean distance between any two points p and q by $d(p, q)$.

Objects	Query	Aggregation function	Query time	Space	Sec.
Points in $\mathbb{R}^2$	Rect.	Closest pair	$\log^2 n$	$n \log^5 n$	2
Random points in $\mathbb{R}^d$ $(d \geq 2)$	Hyper-rect.	Closest pair	$\log^{2d} n$	$n \log^{3d-2} n$ (expected)	3
Random points in $\mathbb{R}^2$	Disk	Closest pair	$n^{2/3+\epsilon}$ (expected)	$n^{1+\epsilon}$ (expected)	4
Points in $\mathbb{R}^d$ $(d \geq 2)$	Hyper-rect.	Closest pair ("partly in" query)	$\log^d n$	$n \log^d n$	5
	Half-space		$n^{1-1/d+\epsilon}$	$n^{1+\epsilon}$	
	Ball		$n^{1-1/(d+1)+\epsilon}$	$n^{1+\epsilon}$	
Points in $\mathbb{R}^2$	Rect.	Farthest pair	$k \log^5 n$ $1 \leq k \leq n$	$(n + (n/k)^2) \log^2 n$	6
Points in $\mathbb{R}^2$	Rect.	$(1-\delta)$-farthest pair	$\frac{1}{\sqrt{\delta}} \log^2 n$	$\frac{1}{\sqrt{\delta}} n \log n$	7
		$(1-\delta)$-farthest pair; variable δ	$\frac{1}{\sqrt{\delta}} \log n + \log^3 n$	$n \log^2 n$	
Points in $\mathbb{R}^2$	Rect.	$(1+\delta)$-width	$\frac{1}{\sqrt{\delta}} \log^3 n$	$\frac{1}{\sqrt{\delta}} n \log^2 n$	8

Table 1: *Summary of results. All results are big-O and, unless noted otherwise, worst-case. Query rectangles are axes-parallel. By "random points" we mean that the points are chosen independently and uniformly at random in the unit-hypercube. For the result in Section 5, by "partly in" we mean that one point in the closest pair is in the query and the other is outside. Here k is a tunable integer-valued parameter, $1 \leq k \leq n$, δ is a real-valued error tolerance parameter $0 < \delta < 1$, $\epsilon > 0$ is a real-valued constant, and $d \geq 2$ is an integer constant. By "variable δ", we mean that it is part of the query; otherwise, it is fixed.*

Our data structures will use balanced binary search trees that store the points of S (or a subset of S) at their leaves. If T is such a binary tree and u is a node of T, then we denote by S_u the subset of S stored at the leaves of the subtree rooted at u.

2.1. Computing the closest pair in a vertical strip or quadrant

A *vertical strip* is the closed region consisting of all points in the plane that are on or between two vertical lines. For a given point a in the plane, the *north-east quadrant* of a is the set $\{p \in \mathbb{R}^2 : p_x \geq a_x, p_y \geq a_y\}$.

Our final data structure uses solutions to the problems of computing, for any vertical query strip Q or for any north-east quadrant Q, the closest pair in $S \cap Q$. For the first problem, Sharathkumar and Gupta [21] have given the data structure stated in the lemma below, which we will use:

Lemma 1. *A set S of n points in the plane can be preprocessed into a data structure of size $O(n \log^2 n)$ such that for any vertical query strip Q, the closest pair in $S \cap Q$ can be reported in $O(\log n)$ time.*

In the rest of this section, we will show how to answer queries of the form: Given a north-east query quadrant Q, report the closest pair in $S \cap Q$.

Let G be the graph with vertex set S in which any two points p and q are connected by an edge if and only if there exists a north-east quadrant Q such that (p, q) is the closest pair in $S \cap Q$. The following lemma states that the number of edges in this graph is $O(n)$. In other words, there are only $O(n)$ distinct closest pairs taken over all $\Theta(n^2)$ combinatorially distinct north-east query quadrants.

Lemma 2. *The graph G defined above is plane and, thus, contains at most $3n - 6$ edges.*

Proof. Since any plane graph with n vertices has at most $3n - 6$ edges, it suffices to show that G is plane. The proof is by contradiction. Let (p, q) and (r, s) be two (straight-line) edges of G and assume that they cross at a point c. By the triangle inequality, we have $d(p, r) < d(p, c) + d(c, r)$ and $d(s, q) < d(s, c) + d(c, q)$. By adding these two inequalities, we obtain

$$d(p, r) + d(s, q) < d(p, q) + d(r, s). \tag{1}$$

Let Q be a north-east quadrant such that (p, q) is the closest pair in $S \cap Q$, and let Q' be a north-east quadrant such that (r, s) is the closest pair

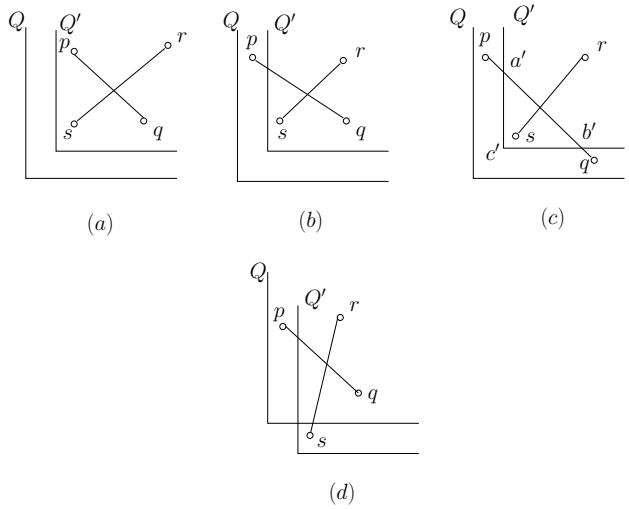

Figure 1: *The different cases for the proof of Lemma 2. Top row: (a) Case 1, (b) Subcase 1.1, and (c) Subcase 1.2. Bottom row: Case 2.*

in $S \cap Q'$. (Q and Q' exist, by definition of G.) We distinguish two main cases (see Figure 1).

Case 1: One of the quadrants Q and Q' is contained in the other.

We may assume without loss of generality that Q' is contained in Q. Since p, q, r, and s are in Q, and since (p, q) is the closest pair in $S \cap Q$, we have

$$d(p, r) > d(p, q) \tag{2}$$

and

$$d(s, q) > d(p, q). \tag{3}$$

Observe that p and q cannot both be contained in Q'. Otherwise, if $d(p, q) < d(r, s)$, the answer to the query Q' cannot be (r, s), whereas if $d(r, s) < d(p, q)$, the answer to the query Q cannot be (p, q). Therefore, there are two subcases.

Subcase 1.1: Exactly one of p and q is in Q'.

We may assume without loss of generality that $p \notin Q'$ and $q \in Q'$. Since q, r, and s are in Q', and since (r, s) is the closest pair in $S \cap Q'$, we have $d(s, q) > d(r, s)$. This, together with (2) implies that $d(p, r) + d(s, q) > d(p, q) + d(r, s)$, which contradicts (1).

Subcase 1.2: Neither p nor q is in Q'.

We may assume without loss of generality that p is to the left of q. Let c' be the bottom-left corner of Q'. Since the edges (p, q) and (r, s) cross, p must be to the north-west of c' and q must be to the south-east of c'. Let a' and b' be the two intersection points between the line segment pq and the boundary of Q', where a' is on the left boundary of Q' and b' is on the bottom boundary of Q'. Then one of r and s is contained in the triangle with vertices a', b', and c'. We may assume without loss of generality that s is in this triangle. Thus, p is above the line through s and q. Since the horizontal and vertical distances between s and q are less than the horizontal and vertical distances between p and q, respectively, it follows that $d(s, q) < d(p, q)$, contradicting (3).

Case 2: The quadrants Q and Q' partially overlap.

We may assume without loss of generality that the bottom-left corner of Q is to the left of (and, thus, above) the bottom-left corner of Q'. At least one of p and q must be contained in Q', because otherwise the edges (p, q) and (r, s) cannot cross. For the same reason, at least one of r and s must be contained in Q. We may assume without loss of generality that q is in Q' and r is in Q. Thus, both lie in $Q \cap Q'$.

Since p, q, and r are in Q, and since (p, q) is the closest pair in $S \cap Q$, we have $d(p, r) > d(p, q)$. Since q, r, and s are in Q', and since (r, s) is the closest pair in $S \cap Q'$, we have $d(s, q) > d(r, s)$. It follows that $d(p, r) + d(s, q) > d(p, q) + d(r, s)$, which contradicts (1). □

For each edge $e = (p, q)$ of the graph G, we define the following point r_e in the plane:

$$r_e = (\min(p_x, q_x), \min(p_y, q_y)).$$

Let

$$R = \{r_e : \ e \text{ is an edge in } G\}.$$

We give each point r_e of R a weight which is defined to be the distance between the endpoints of e.

Lemma 3. *Let Q be a north-east query quadrant, let (p, q) be the closest pair in $S \cap Q$, and let r_e be the point in $R \cap Q$ whose weight is minimum. Then $e = (p, q)$.*

Proof. By the definition of the graph G, (p, q) is an edge of G. The claim follows from the fact that for any edge $f = (u, v)$ in G, both u and v are in Q if and only if the point r_f is in Q. □

Thus, answering closest pair queries in a north-east quadrant is equivalent to answering queries of the following form: Given a north-east query quadrant Q, report the point of minimum weight in $R \cap Q$. Using a two-dimensional range tree and fractional cascading (as explained in Chapter 5 in de Berg *et al.* [6]), we obtain the following result:

Lemma 4. *A set S of n points in the plane can be preprocessed into a data structure of size $O(n \log n)$ such that for any north-east query quadrant Q, the closest pair in $S \cap Q$ can be reported in $O(\log n)$ time.*

Remark 1. In order to construct the data structure of Lemma 4, we need the graph G. Unfortunately, even though this graph has only $O(n)$ edges, it is not clear if it can be obtained in subquadratic time.

2.2. The opposite-quadrant lemma

In this section, we assume that the set S does not contain the origin. Recall that the L_∞-distance between two points p and q is defined to be

$$d_\infty(p, q) := \max(|p_x - q_x|, |p_y - q_y|).$$

Let A be the set consisting of all points of S whose x- and y-coordinates are less than zero, and let B be the set consisting of all points of S whose x- and y-coordinates are larger than zero. Let A_5 be the subset of A consisting of the $\min(5, |A|)$ points that are L_∞-closest to the origin, and let B_5 be the subset of B consisting of the $\min(5, |B|)$ points that are L_∞-closest to the origin; refer to Figure 2.

Lemma 5. *Let (p, q) be the closest pair in S and assume that $p \in A$ and $q \in B$. Then, $p \in A_5$ and $q \in B_5$.*

Proof. Assume the claim is not true. Then we may assume without loss of generality that $p \notin A_5$. Observe that this implies that A_5 consists of five elements. Let δ be the largest L_∞-distance between the origin and any point in A_5. Since $p \notin A_5$, and p and q are in opposite quadrants, we have $d(p, q) \geq \delta$. The box $[-\delta, 0]^2$ contains all points of A_5. If we partition this box into four subboxes with sides of length $\delta/2$, then one of the subboxes contains two points a and a' of A_5. Since

$$d(a, a') \leq \sqrt{2} \cdot \delta/2 < \delta \leq d(p, q),$$

it follows that (p, q) is not a closest pair in S. This is a contradiction. □

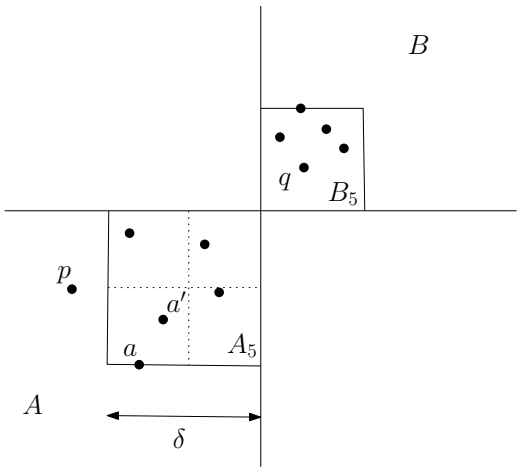

Figure 2: *Illustrating Lemma 5.*

2.3. Computing L_∞-neighbors in a quadrant

In this section, we assume that the x- and y-coordinates of all points of S are positive. We want to preprocess S such that queries of the following form can be answered: Given a query point q with positive coordinates, let Q_q be the *south-west quadrant* of q, i.e., $Q_q := \{p \in \mathbb{R}^2 : p_x \leq q_x, p_y \leq q_y\}$. Report the $\min(5, |S \cap Q_q|)$ points in $S \cap Q_q$ that are L_∞-closest to the origin; refer to Figure 3. We show how such queries can be answered for the case when the query point q is on or above the diagonal $y = x$. (The case when q is below this diagonal can be solved in a symmetric way.)

Let A be the set of all points of S that are on or below the diagonal $y = x$, and let $B := S \setminus A$. Our data structure consists of the following:

1. An array storing the points of A sorted by their x-coordinates. For ease of notation, we denote this array by A.
2. An array storing the points of B sorted by their x-coordinates. With each entry p in this array, we store the following information: Let $B_p := \{b \in B : b_x \leq p_x\}$. We store with p the $\min(5, |B_p|)$ lowest points in B_p. For ease of notation, we denote this array by B.

Consider a south-west query quadrant Q_q, where the point q has positive coordinates and is on or above the diagonal $y = x$. The algorithm does the following.

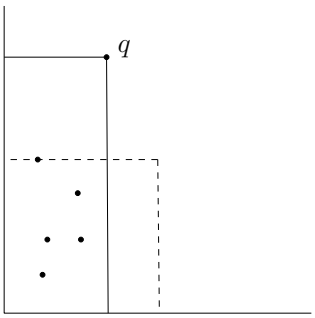

Figure 3: *Among all points to the south-west of q, report the five points that are L_∞-closest to the origin.*

1. Initialize an empty set A'. For each point p that is among the leftmost $\min(5, |A|)$ points in the array A, add p to A' if and only if $p_x \leq q_x$.
2. Do a binary search with the x-coordinate q_x in B, and find the rightmost point p in B for which $p_x \leq q_x$. Initialize an empty set B'. Recall that p stores the $\min(5, |B_p|)$ lowest points in B_p. For each r among these $\min(5, |B_p|)$ points, add r to B' if and only if $r_y \leq q_y$.
3. Compute and report the $\min(5, |A' \cup B'|)$ points in $A' \cup B'$ that are L_∞-closest to the origin.

The correctness of this query algorithm follows from the following facts. First, for each point p in A, the L_∞-distance between p and the origin is equal to the x-coordinate of p. It follows that the set A' that is computed in Step 1, contains the $\min(5, |A \cap Q_q|)$ points in $A \cap Q_q$ that are L_∞-closest to the origin. Second, for each point p in B, the L_∞-distance between p and the origin is equal to the y-coordinate of p. Thus, the set B' that is computed in Step 2, contains the $\min(5, |B \cap Q_q|)$ points in $B \cap Q_q$ that are L_∞-closest to the origin. Third, the union of the sets A' and B' contains the $\min(5, |S \cap Q_q|)$ points in $S \cup Q_q$ that are L_∞-closest to the origin.

Lemma 6. *Let S be a set of n points in the plane, all of whose x- and y-coordinates are positive. The set S can be preprocessed into a data structure of size $O(n)$ such that for any query point q with positive coordinates, the $\min(5, |S|)$ points in $S \cap Q_q$ that are L_∞-closest to the origin, can be reported in $O(\log n)$ time.*

2.4. Computing the closest pair in an anchored 3-sided rectangle

Let ℓ be a fixed vertical line. An *anchored 3-sided rectangle* Q is an axes-parallel rectangle that is unbounded in the positive y-direction and that is intersected by ℓ. Thus, such a rectangle can be written as $Q = [a, b] \times [c, \infty)$, where the x-coordinate of ℓ is between a and b.

We consider queries of the following form: Given an anchored 3-sided rectangle Q, report the closest pair in $S \cap Q$.

Let T be a balanced binary search tree storing the points of S at its leaves, sorted by their y-coordinates. For each internal node u of T, we define the following. Let u_1 and u_2 be the left and right children of u, respectively. We define h_u to be the average of the y-coordinates of (i) the point stored at the rightmost leaf in the subtree of u_1 and (ii) the point stored at the leftmost leaf in the subtree of u_2. Let ℓ'_u be the horizontal line $y = h_u$, and let X_u be the intersection between ℓ and ℓ'_u. Observe that all points of S_{u_1} are below ℓ'_u and all points of S_{u_2} are above ℓ'_u. (Recall the notation established at the beginning of Section 2.) Let $S^l_{u_1}$ be the set of points of S_{u_1} that are to the left of the line ℓ, and let $S^r_{u_1}$ be the set of points of S_{u_1} that are to the right of ℓ. Define $S^l_{u_2}$ and $S^r_{u_2}$ similarly with respect to S_{u_2}. Thus, the lines ℓ and ℓ'_u, whose intersection is X_u, partition the plane into four quadrants. The subsets $S^l_{u_1}$, $S^r_{u_1}$, $S^l_{u_2}$, and $S^r_{u_2}$ form a partition of S_u and each of them is in a unique quadrant; refer to Figure 4.

Each internal node u of T stores the following information:

1. Node u stores a pointer to the data structure of Lemma 4. This structure stores the set $S^l_{u_1} \cup S^l_{u_2}$ and supports queries of the form "report the closest pair in a north-east query quadrant".
2. Node u stores a pointer to the data structure of Lemma 4. This structure stores the set $S^r_{u_1} \cup S^r_{u_2}$ and supports queries of the form "report the closest pair in a north-west query quadrant".
3. Node u stores a pointer to the data structure of Lemma 1. This structure stores the set $S_{u_2} = S^l_{u_2} \cup S^r_{u_2}$ and supports queries of the form "report the closest pair in a vertical query strip".
4. Node u stores pointers to four data structures of Lemma 6, one for each subset $S^l_{u_1}$, $S^r_{u_1}$, $S^l_{u_2}$, and $S^r_{u_2}$. More precisely, for each $H \in \{S^l_{u_1}, S^r_{u_1}, S^l_{u_2}, S^r_{u_2}\}$, node u stores a pointer to a data structure storing the set H and supporting queries of the form "given a query point q that is in the quadrant of the point $X_u = \ell \cap \ell'_u$ containing the subset

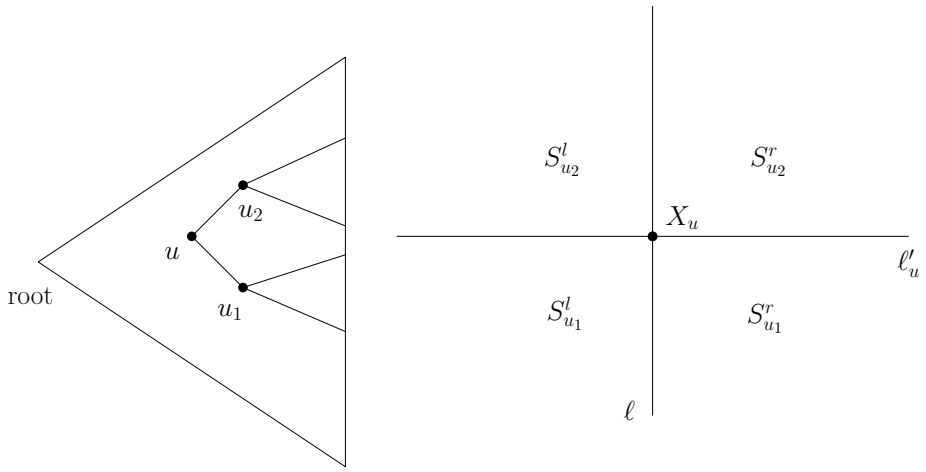

Figure 4: *Illustrating the data structure for closest-pair queries in an anchored 3-sided rectangle.*

H, report the $\min(5, |H \cap Q_q|)$ points in $H \cap Q_q$ that are L_∞-closest to the point X_u, where Q_q is the quadrant of q that contains X_u".

Let $Q = [a, b] \times [c, \infty)$ be an anchored 3-sided query rectangle. Recall that Q is intersected by the vertical line ℓ. The following algorithm ClosestPair(v, Q) takes as input a node v of T and returns the closest pair in $S_v \cap Q$. Thus, by calling ClosestPair(v, Q), where v is the root of T, we obtain the closest pair in $S \cap Q$. Algorithm ClosestPair(v, Q) does the following.

1. If v is a leaf, return ∞ and terminate. Otherwise, start at v and follow the path on the right spine of the subtree of v until the first node u is reached such that the horizontal line ℓ'_u stored at u intersects Q. (If u does not exist, then the algorithm returns ∞ and terminates.) Let u_1 and u_2 be the left and right children of u, respectively. Consider the point X_u, and the subsets $S^l_{u_1}$, $S^r_{u_1}$, $S^l_{u_2}$, and $S^r_{u_2}$ as defined above.
2. Use the data structure storing the set $S^l_{u_1} \cup S^l_{u_2}$ to find the closest pair in the north-east quadrant of the point (a, c).
3. Use the data structure storing the set $S^r_{u_1} \cup S^r_{u_2}$ to find the closest pair in the north-west quadrant of the point (b, c).
4. Use the data structure storing the set $S^l_{u_2} \cup S^r_{u_2}$ to find the closest pair in the vertical query strip bounded by the vertical lines through (a, c)

and (b, c).

5. Use the data structure storing $S^l_{u_1}$ to find the five points in the north-east quadrant of (a, c) that are L_∞-closest to the point X_u. Use the data structure storing $S^r_{u_2}$ to find the five points in the south-west quadrant of (b, ∞) that are L_∞-closest to the point X_u. Compute the closest pair among the resulting 25 pairs of points.
6. Use the data structure storing $S^l_{u_2}$ to find the five points in the south-east quadrant of (a, ∞) that are L_∞-closest to the point X_u. Use the data structure storing $S^r_{u_1}$ to find the five points in the north-west quadrant of (b, c) that are L_∞-closest to the point X_u. Compute the closest pair among the resulting 25 pairs of points.
7. Call algorithm ClosestPair(u_1, Q).
8. Return the closest pair found in these steps.

To prove the correctness of the algorithm, let (p, q) be the closest pair in $S \cap Q$. Consider the node u found in Step 1. First observe that S_u contains all points of $S \cap Q$. The following six cases are possible.

Case 1: p and q are both to the left of ℓ. Since the query region Q is intersected by the vertical line ℓ, querying $S^l_{u_1} \cup S^l_{u_2}$ with Q is equivalent to querying this set with the north-east quadrant of the point (a, c). Therefore, (p, q) is found in Step 2.

Case 2: p and q are both to the right of ℓ. This case is symmetric to the first case; (p, q) is found in Step 3.

Case 3: p and q are both above the line ℓ'_u. Since Q is intersected by ℓ'_u, querying $S^l_{u_2} \cup S^r_{u_2}$ with Q is equivalent to querying this set with the vertical query strip bounded by the vertical lines through (a, c) and (b, c). Therefore, (p, q) is found in Step 4.

Case 4: p is in the south-west quadrant of X_u and q is in the north-east quadrant of X_u. It follows from Lemmas 5 and 6 that (p, q) is found in Step 5.

Case 5: p is in the north-west quadrant of X_u and q is in the south-east quadrant of X_u. This case is symmetric to the previous case; (p, q) is found in Step 6.

Case 6: p and q are both below the line ℓ'_u. In this case, both p and q are contained in the subtree of u_1. Thus, by an inductive argument, the pair (p, q) is found in Step 7.

Lemma 7. *Let ℓ be a vertical line. A set S of n points in the plane can be preprocessed into a data structure of size $O(n \log^3 n)$ such that for any*

anchored 3-sided query rectangle Q, the closest pair in $S \cap Q$ can be reported in $O(\log^2 n)$ time.

2.5. Computing the closest pair in an anchored rectangle

Let ℓ be a fixed horizontal line. An *anchored rectangle* Q is an axes-parallel rectangle that is intersected by ℓ. Such a rectangle can be written as $Q = [a, b] \times [c, d]$, where the y-coordinate of ℓ is between c and d.

We consider queries of the following form: Given an anchored rectangle Q, report the closest pair in $S \cap Q$.

Let T be a balanced binary search tree storing the points of S at its leaves, sorted by their x-coordinates. For each internal node u of T, we define the following. Let u_1 and u_2 be the left and right children of u, respectively. We define v_u to be the average of the x-coordinates of (i) the point stored at the rightmost leaf in the subtree of u_1 and (ii) the point stored at the leftmost leaf in the subtree of u_2. Let ℓ'_u be the vertical line $x = v_u$, and let X_u be the intersection between ℓ and ℓ'_u. All points of S_{u_1} are to the left of ℓ'_u and all points of S_{u_2} are to the right of ℓ'_u. Let $S^a_{u_1}$ be the set of points of S_{u_1} that are above the line ℓ, and let $S^b_{u_1}$ be the set of points of S_{u_1} that are below ℓ. Define $S^a_{u_2}$ and $S^b_{u_2}$ similarly with respect to S_{u_2}. Thus, the lines ℓ and ℓ'_u, whose intersection is X_u, partition the plane into four quadrants. The subsets $S^a_{u_1}$, $S^b_{u_1}$, $S^a_{u_2}$, and $S^b_{u_2}$ form a partition of S_u and each of them is in a unique quadrant.

Each internal node u of T stores the following information:

1. Node u stores a pointer to the data structure of Lemma 7. This structure stores the set $S^a_{u_1} \cup S^a_{u_2}$ and supports queries of the form "report the closest pair in the anchored 3-sided query rectangle $[a, b] \times (-\infty, d]$", for query rectangles that are anchored with respect to ℓ'_u.
2. Node u stores a pointer to the data structure of Lemma 7. This structure stores the set $S^b_{u_1} \cup S^b_{u_2}$ and supports queries of the form "report the closest pair in the anchored 3-sided query rectangle $[a, b] \times [c, \infty)$", for query rectangles that are anchored with respect to ℓ'_u.
3. Node u stores a pointer to the data structure of Lemma 7. This structure stores the set $S_{u_1} = S^a_{u_1} \cup S^b_{u_1}$ and supports queries of the form "report the closest pair in the anchored 3-sided query rectangle $[a, \infty) \times [c, d]$", for query rectangles that are anchored with respect to ℓ.

4. Node u stores a pointer to the data structure of Lemma 7. This structure stores the set $S_{u_2} = S^a_{u_2} \cup S^b_{u_2}$ and supports queries of the form "report the closest pair in the anchored 3-sided query rectangle $(-\infty, b] \times [c, d]$", for query rectangles that are anchored with respect to ℓ.
5. Node u stores pointers to four data structures of Lemma 6, one for each subset $S^a_{u_1}$, $S^b_{u_1}$, $S^a_{u_2}$, and $S^b_{u_2}$. More precisely, for each $H \in \{S^a_{u_1}, S^b_{u_1}, S^a_{u_2}, S^b_{u_2}\}$, node u stores a pointer to a data structure storing the set H and supporting queries of the form "given a query point q that is in the quadrant of the point $X_u = (\ell \cap \ell'_u)$ containing the subset H, report the $\min(5, |H \cap Q_q|)$ points in $H \cap Q_q$ that are L_∞-closest to the point X_u, where Q_q is the quadrant of q that contains X_u.

Let $Q = [a, b] \times [c, d]$ be an anchored query rectangle. The algorithm that finds the closest pair in $S \cap Q$ does the following.

1. Starting at the root of T, follow the path until the first node u is reached such that the vertical line ℓ'_u stored at u intersects Q. Let u_1 and u_2 be the left and right children of u, respectively. Consider the point X_u, and the subsets $S^a_{u_1}$, $S^b_{u_1}$, $S^a_{u_2}$, and $S^b_{u_2}$ as defined above.
2. Use the data structure storing the set $S^a_{u_1} \cup S^a_{u_2}$ to find the closest pair in the anchored 3-sided query rectangle $[a, b] \times (-\infty, d]$.
3. Use the data structure storing the set $S^b_{u_1} \cup S^b_{u_2}$ to find the closest pair in the anchored 3-sided query rectangle $[a, b] \times [c, \infty)$.
4. Use the data structure storing the set $S^a_{u_1} \cup S^b_{u_1}$ to find the closest pair in the anchored 3-sided query rectangle $[a, \infty) \times [c, d]$.
5. Use the data structure storing the set $S^a_{u_2} \cup S^b_{u_2}$ to find the closest pair in the anchored 3-sided query rectangle $(-\infty, b] \times [c, d]$.
6. Use the data structure storing $S^a_{u_1}$ to find the five points in the south-east quadrant of (a, d) that are L_∞-closest to the point X_u. Use the data structure storing $S^b_{u_2}$ to find the five points in the north-west quadrant of (b, c) that are L_∞-closest to the point X_u. Compute the closest pair among the resulting 25 pairs of points.
7. Use the data structure storing $S^a_{u_2}$ to find the five points in the south-west quadrant of (b, d) that are L_∞-closest to the point X_u. Use the data structure storing $S^b_{u_1}$ to find the five points in the north-east quadrant of (a, c) that are L_∞-closest to the point X_u. Compute the closest pair among the resulting 25 pairs of points.

8. Return the closest pair found in these steps.

The correctness proof of this query algorithm follows by a similar analysis as in the correctness proof in Section 2.4.

Lemma 8. *Let ℓ be a horizontal line. A set S of n points in the plane can be preprocessed into a data structure of size $O(n \log^4 n)$ such that for any anchored query rectangle Q, the closest pair in $S \cap Q$ can be reported in $O(\log^2 n)$ time.*

2.6. General closest pair rectangle queries

In this final subsection, we show how to answer general queries of the following form: Given an axes-parallel rectangle Q, report the closest pair in $S \cap Q$.

Let T be a balanced binary search tree storing the points of S at its leaves, sorted by their y-coordinates. For each internal node u of T, we define the horizontal line ℓ'_u as in Section 2.4. Each internal node u of T stores the data structure of Lemma 8. This structure stores the set S_u and supports queries of the form "report the closest pair in an anchored query rectangle", for rectangles that are anchored with respect to the line ℓ'_u.

To answer a query with a given axes-parallel rectangle Q, we do the following. Starting at the root of T, follow the path until the first node u is reached such that the horizontal line ℓ'_u stored at u intersects Q. Then we use the data structure storing S_u to find the closest pair in the query rectangle Q (which is anchored with respect to ℓ'_u). The correctness follows from the fact that $S \cap Q = S_u \cap Q$.

Theorem 9. *A set S of n points in the plane can be preprocessed into a data structure of size $O(n \log^5 n)$ such that for any axes-parallel query rectangle Q, the closest pair in $S \cap Q$ can be reported in $O(\log^2 n)$ time.*

3. Closest pair rectangle queries on randomly distributed points

The data structure of Sharathkumar and Gupta [21] answers closest pair queries for axes-parallel rectangles in $O(\log^3 n)$ time using $O(n \log^3 n)$ space, whereas the data structure of Theorem 9 has a query time of $O(\log^2 n)$ using $O(n \log^5 n)$ space. It is not clear, however, if these data structures can be built in subquadratic time; see Remark 1.

In this section, we consider the case when the points of S are randomly distributed in the unit-square, which is reasonable for many applications. For this case, we will obtain a data structure of expected size $O(n \log^4 n)$ that can be used to answer queries in $O(\log^4 n)$ time. Even though this structure is asymptotically worse than the one in [21], it is quite simple and practical, extends naturally to any fixed dimension $d > 2$, and for $d = 2$, can be built more efficiently than the one in [21].

We consider the two-dimensional problem first. Our approach will be to precompute each point-pair (p, q), with $p, q \in S$, that is the closest pair for at least one axes-parallel query rectangle. We then store each such pair as a weighted point in a four-dimensional range tree. The four dimensions are, successively, the x- and y-coordinates of p and the x- and y-coordinates of q. The weight is the distance $d(p, q)$.

Given an axes-parallel query rectangle Q, we can find the closest pair in $S \cap Q$ by doing a range-minimum query [8] on the tree with the four-dimensional rectangle $Q \times Q$.

Let Λ denote the number of precomputed pairs (p, q). (We consider these pairs to be ordered, where p is to the left of q.) Observe that $\Lambda = O(n^2)$. Then this data structure can be built in time which is equal to the time needed to compute the Λ pairs plus $O(\Lambda \log^3 n)$ time, it uses $O(\Lambda \log^3 n)$ space and it has a query time of $O(\log^4 n)$. Thus, if Λ is "small", then this will be an efficient and practical solution.

More formally, let $\mathcal{Q}$ be the set of all possible axes-parallel query rectangles. Let Λ be the number of pairs (p, q), with $p, q \in S$ and p to the left of q, such that there is a rectangle $Q \in \mathcal{Q}$ for which (p, q) is a closest pair in $S \cap Q$.

Unfortunately, as the following example shows, Λ can be $\Theta(n^2)$ in the worst case. Consider the point-sets S' and S'', each of size $n/2$, on the unit-circle, where the points of S' are in the open angular interval $(\pi/2, \pi)$, while those of S'' are in the open interval $(3\pi/2, 2\pi)$; see Figure 5. Let $S = S' \cup S''$. For any two points p and q in S, where p is to the left of q, let $R(p, q)$ be the (unique) closed rectangle with diagonal $\overline{pq}$. If $p \in S'$ and $q \in S''$, then $R(p, q)$ contains exactly p and q, so (p, q) is the closest pair in $S \cap R(p, q)$ and it contributes a count of 1 to Λ. Thus, $\Lambda = \Omega(n^2)$. Since, $\Lambda = O(n^2)$ as well, the claim follows.

Fortunately, as we shall see below, the situation is much better if the points of S are chosen independently and uniformly at random in the unit-square. Specifically, it turns out that, for this case, the expected value of Λ

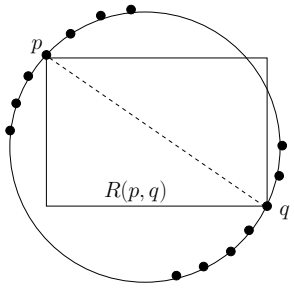

Figure 5: *An example point-set for which* $\Lambda = \Theta(n^2)$.

is $\Theta(n \log n)$.

In what follows, we say that the rectangle $R(p, q)$ is *empty* if it contains no point of $S \setminus \{p, q\}$.

Lemma 10. *Let (p, q) be an ordered point-pair, with $p, q \in S$ and p to the left of q. This pair contributes a count of one to Λ if and only if $R(p, q)$ is empty.*

Proof. If $R(p, q)$ is empty, then (p, q) is the closest pair in $S \cap R(p, q)$, hence it contributes one to Λ.

Assume that (p, q) contributes one to Λ, but $R(p, q)$ is not empty. Let $r \in S \setminus \{p, q\}$ be a point in $R(p, q)$. Since $\overline{pq}$ is a diagonal of $R(p, q)$, we have $d(r, p) < d(p, q)$ (and $d(r, q) < d(p, q)$). Thus, (p, q) is not a closest pair in $S \cap R(p, q)$. Any other axes-parallel rectangle Q that contains p and q also contains $R(p, q)$. Thus Q contains r, so that (p, q) is not a closest pair in $S \cap Q$ either. Therefore, (p, q) contributes zero to Λ—a contradiction. □

In view of Lemma 10, Λ is equal to the number of empty rectangles $R(p, q)$, taken over all ordered point-pairs (p, q), with $p, q \in S$ and p to the left of q. If the points of S are chosen independently and uniformly at random in the unit-square, then the points are "well-distributed", so, intuitively, there should not be too many empty rectangles $R(p, q)$. Indeed, for this case, it follows from Proposition 1 in Felsner [7] that the expected number of such empty rectangles is $O(n \log n)$. The result in [7] is stated without proof; for completeness, we present a proof here.

Lemma 11. *For a set S of n points that are chosen independently and uniformly at random in the unit-square, the expected value of Λ is $\Theta(n \log n)$.*

PROOF. First observe that with probability one, no two points in S have the same x- or y-coordinates. Thus, we may assume that all x-coordinates and all y-coordinates of the points in S are distinct.

Let $p_1, p_2, \ldots, p_n$ denote the points of S sorted by their x-coordinates. For $1 \leq i < j \leq n$, we define the random variable X_{ij} by

$$X_{ij} = \begin{cases} 1 & \text{if } R(p_i, p_j) \text{ is empty,} \\ 0 & \text{otherwise.} \end{cases}$$

Then, we can write Λ (which is a random variable) as

$$\Lambda = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}.$$

By the linearity of expectation, we have that the expected value of Λ is

$$E(\Lambda) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E(X_{ij}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr(X_{ij} = 1).$$

Let $1 \leq i < j \leq n$, and consider the set

$$\bar{S} = \{p_i, p_{i+1}, \ldots, p_{j-1}, p_j\}.$$

For each point s in $\bar{S}$, we define $rk(s)$ to be the number of points in $\bar{S}$ whose y-coordinates are less than or equal to the y-coordinate of s. (Thus, if s has the minimum y-coordinate, then $rk(s) = 1$.) Observe that $X_{ij} = 1$ if and only if $rk(p_i)$ and $rk(p_j)$ differ by exactly one.

Let $t = j - i + 1$. There are $2(t-1)!$ permutations of the points in $\bar{S}$ in which $rk(p_i)$ and $rk(p_j)$ differ by one. Therefore,

$$\Pr(X_{ij} = 1) = \frac{2(t-1)!}{t!} = \frac{2}{t} = \frac{2}{j-i+1}.$$

It follows that

$$\begin{aligned} E(\Lambda) &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\ &= 2\left(\sum_{i=1}^{n-1} (H_{n-i+1} - 1)\right) \\ &= 2\sum_{k=1}^{n} H_k - 2n, \end{aligned}$$

where H_k is the k-th harmonic number. Since $\ln k \leq H_k \leq 1 + \ln k$, it follows that $E(\Lambda) = \Theta(n \log n)$. □

The problem of computing the Λ pairs (p, q) for which $R(p, q)$ is empty is equivalent to computing the direct north-east and north-west domination pairs. (Pair (p, q) is a *direct domination pair* if p dominates q and there is no other point r such that p dominates r and r dominates q. Each such pair (p, q) defines the diagonal of an empty rectangle.) Using the algorithm of Güting *et al.* [11], these Λ pairs can be computed in $O(n \log n + \Lambda)$ time. Given the preceding discussion, we have the following result.

Theorem 12. *Let S be a set of n points that are chosen independently and uniformly at random in the unit-square. In $O(n \log^4 n)$ expected time, the set S can be preprocessed into a data structure of expected size $O(n \log^4 n)$ such that for any axes-parallel query rectangle Q, the closest pair in $S \cap Q$ can be reported in $O(\log^4 n)$ time.*

The previous approach can be generalized to answering closest pair queries for axes-parallel query hyper-rectangles in $\mathbb{R}^d$ also, for any fixed $d \geq 3$. This is based on a result from [15] (see also [5]) that there are, expected, $O(n \log^{d-1} n/(d-1)!)$ direct domination pairs (p, q) in a set of n points that are drawn independently at random from the unit-hypercube in $\mathbb{R}^d$. Thus the expected number Λ of point-pairs in $\mathbb{R}^d$ such that each is a closest pair for at least one query hyper-rectangle is $O(n \log^{d-1} n)$. Our problem reduces now to storing each such pair (p, q) as a weighted point in a $2d$-dimensional range tree, where the $2d$ dimensions are, successively, the x_1-, x_2-, ..., x_d-coordinates of p and the x_1-, x_2-, ..., x_d-coordinates of q, and the weight is the distance $d(p, q)$. The expected space used is $O(E(\Lambda) \log^{2d-1} n) = O(n \log^{3d-2} n)$ and the query time is $O(\log^{2d} n)$.

Theorem 13. *Let S be a set of n points that are chosen independently and uniformly at random in the unit-hypercube in $\mathbb{R}^d$, $d \geq 2$. S can be preprocessed into a data structure of expected size $O(n \log^{3d-2} n)$ such that for any axes-parallel query rectangle Q, the closest pair in $S \cap Q$ can be reported in $O(\log^{2d} n)$ time.*

4. Closest pair disk queries on randomly distributed points

Let S be a set of n points in the plane. In this section, we consider queries of the following form: Given a query disk D, report the closest pair in $S \cap D$.

Our approach will be the same as in Section 3. Let $\mathcal{D}$ be the set of all disks. Let $S_{\mathcal{D}}$ be the set of all ordered point pairs (p, q), with $p, q \in S$ and p to the left of q, such that there is a disk $D \in \mathcal{D}$ for which (p, q) is the closest pair in $S \cap D$. We give each pair (p, q) in $S_{\mathcal{D}}$ a weight, which is equal to $d(p, q)$.

Given a query disk D, finding the closest pair in $S \cap D$ is then equivalent to finding the point-pair (p, q) in $S_{\mathcal{D}}$ of minimum weight for which both p and q are in D. Let $\Lambda = |S_{\mathcal{D}}|$. By using halfspace composition techniques (see Agarwal and Erickson [2]), we can solve this problem with a query time of $O(\Lambda^{2/3+\epsilon})$ using $O(\Lambda^{1+\epsilon})$ space for any constant $\epsilon > 0$.

We will show below that for random points in the unit-square, the expected value of Λ is $O(n \log n)$. It follows that we obtain a data structure whose expected query time is proportional to $E(\Lambda^{2/3+\epsilon})$ and whose expected size is $E(\Lambda^{1+\epsilon})$. Since $\Lambda \leq n^2$, we have $\Lambda^{1+\epsilon} \leq \Lambda n^{2\epsilon}$ and, thus,

$$E(\Lambda^{1+\epsilon}) \leq E(\Lambda) n^{2\epsilon} = O(n^{1+3\epsilon}).$$

Recall Jensen's inequality [17] which states that for any convex function f, $E(f(\Lambda)) \geq f(E(\Lambda))$. The function $f(x) = -x^{2/3+\epsilon}$ (for $x > 0$ and fixed ϵ with $0 < \epsilon < 1/3$) is convex, because the second derivative is positive. It follows that

$$E(\Lambda^{2/3+\epsilon}) \leq (E(\Lambda))^{2/3+\epsilon} = O(n^{2/3+2\epsilon}).$$

Thus, by replacing ϵ by $\epsilon/3$, we obtain a data structure whose expected query time is $O(n^{2/3+\epsilon})$ and whose expected size is $O(n^{1+\epsilon})$.

It remains to show that $E(\Lambda) = O(n \log n)$. Let $\mathcal{R}$ be the set of all axes-parallel rectangles. Let S_r be the set of all ordered point-pairs (p, q), with $p, q \in S$ and p to the left of q, such that there is a rectangle $R \in \mathcal{R}$ for which (p, q) is the closest pair in $S \cap R$.

At first sight, it seems that $S_{\mathcal{D}} \subseteq S_r$. However, as Figure 6 shows, this is not necessarily true. Consider, however, the following modification. Let $\mathcal{T}$ be the set of all axes-parallel right triangles (by "axes-parallel", we mean that the non-hypotenuse sides are parallel to the coordinate axes). Let S_t be the set of all ordered pairs (p, q), with $p, q \in S$ and p to the left of q, such that there is an axes-parallel right triangle $T \in \mathcal{T}$ for which (p, q) is the closest pair in $S \cap T$.

Lemma 14. $S_{\mathcal{D}} \subseteq S_t$.

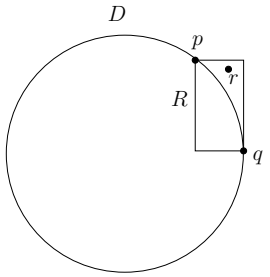

Figure 6: *The pair (p, q) is in the set $S_{\mathcal{D}}$, but not in the set S_r.*

PROOF. Let (p, q) be any point-pair from S that is in $S_{\mathcal{D}}$. We will show that (p, q) is in S_t. By definition, there exists a disk $D \in \mathcal{D}$ such that (p, q) is the closest pair in $S \cap D$. Let $R = R(p, q)$ be the axes-parallel rectangle with diagonal $\overline{pq}$. Let $T^+ = T^+(p, q)$ be the subset of R lying on or above $\overline{pq}$, and let $T^- = T^-(p, q)$ be the subset of R lying on or below $\overline{pq}$. Observe that both T^+ and T^- belong to $\mathcal{T}$.

We establish our result via the following sequence of claims:

- **Claim (i):** At least one of T^+ and T^- is contained in D.

  To prove this, let C be the disk with diameter $\overline{pq}$. Then the third vertices of T^+ and of T^- (i.e., the ones at the right angles) lie on the boundary of C, so both T^+ and T^- are contained in C. If $C \subseteq D$, then we are done. Otherwise, since p and q are contained in D, the boundaries of C and D meet at no more than two points and these are on the boundary of the same semi-disk of C. Thus, the other semi-disk of C is contained in D, and so is the corresponding triangle T^+ or T^-.

- **Claim (ii):** Assume without loss of generality that T^+ is contained in D. Then T^+ is empty, i.e., it does not contain any point of $S \setminus \{p, q\}$.

  To prove this, assume that r is a point of $S\setminus\{p, q\}$ that is in T^+. Since $\overline{pq}$ is the hypotenuse of T^+, we have $d(r, p) < d(p, q)$ (and $d(r, q) < d(p, q)$). Since T^+ is contained in D, so is r. It follows that (p, q) is not a closest pair in $S \cap D$—a contradiction.

- **Claim (iii):** (p, q) is in S_t.

  Since T^+ is empty, (p, q) is the closest pair in $S \cap T^+$. The claim follows now from the definition of S_t. This also completes the proof of

the lemma.

□

By Lemma 14, we have $|S_{\mathcal{D}}| \leq |S_t|$. Below, we will prove that, for points that are randomly distributed in the unit-square, the expected size of S_t is $O(n \log n)$. This implies the same upper bound for the expected size of $S_{\mathcal{D}}$. Our analysis will use the following lemma.

Lemma 15. *Let (p, q) be an ordered point-pair, with $p, q \in S$ and p to the left of q. Then, (p, q) is in S_t if and only if T^+ or T^- is empty.*

PROOF. Assume that T^+ or T^- is empty. Then (p, q) is the closest pair in $S \cap T^+$ or $S \cap T^-$, implying that (p, q) is in S_t.

Assume that (p, q) is in S_t, but T^+ and T^- are both non-empty. Let $r \in S \setminus \{p, q\}$ be in T^+. As in the proof of Claim (ii) in Lemma 14, we have $d(r, p) < d(p, q)$ (and $d(r, q) < d(p, q)$). Thus, (p, q) is not a closest pair in $S \cap T^+$. Similarly, the existence in T^- of a point $r' \in S \setminus \{p, q\}$ establishes that (p, q) is not a closest pair in $S \cap T^-$ either. Any other axes-parallel triangle $T \in \mathcal{T}$ that contains p and q also contains T^+ or T^-.[4] Thus T contains r or r', so (p, q) is not a closest pair in $S \cap T$ either. Therefore, (p, q) is not in S_t—a contradiction. □

Thus, to derive an upper bound on $|S_t|$, we need to determine the number of empty triangles $T^+(p, q)$ or $T^-(p, q)$. In the worst-case, this is $\Theta(n^2)$. The following result shows that the bound is much better for random point sets.

Lemma 16. *For a set S of n points that are chosen independently and uniformly at random in the unit-square, the expected number of point-pairs (p, q), where $p, q \in S$ and p is to the left of q, and such that $T^+(p, q)$ or $T^-(p, q)$ is empty, is $O(n \log n)$.*

[4] The proof involves a tedious case analysis: First consider the case when the supporting line of $\overline{pq}$ has negative slope. Then look at four configurations for T, where its interior faces NW, SE, SW, or NE. In each case, the claim is true. The case when the supporting line of $\overline{pq}$ has positive slope is symmetric. The case of zero (or infinite) slope is trivial.

PROOF. Let $p_1, p_2, \ldots, p_n$ denote the points of S sorted by their x-coordinates. For $1 \leq i < j \leq n$, we define $T_{ij}^+ = T(p_i p_j)^+$ and $T_{ij}^- = T(p_i p_j)^-$. Let X_{ij} be the random variable defined by

$$X_{ij} = \begin{cases} 1 & \text{if } T_{ij}^+ \text{ or } T_{ij}^- \text{ is empty,} \\ 0 & \text{otherwise.} \end{cases}$$

Then

$$|S_t| = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}.$$

By the linearity of expectation, we have

$$E(|S_t|) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E(X_{ij}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr(X_{ij} = 1).$$

Consider two fixed indices i and j with $1 \leq i < j \leq n$, and consider the set

$$\bar{S} = \{p_i, p_{i+1}, \ldots, p_{j-1}, p_j\}.$$

For any integer k, let E_k be the event "the axes-parallel rectangle R with diagonal $p_i p_j$ contains exactly k points of $\bar{S} \setminus \{p_i, p_j\}$". Then

$$\Pr(X_{ij} = 1) = \sum_{k} \Pr(X_{ij} = 1 \mid E_k) \cdot \Pr(E_k).$$

Fix an integer k and assume that E_k holds. Then $X_{ij} = 1$ if and only if (i) all k points in $R \cap (\bar{S} \setminus \{p_i, p_j\})$ are in T_{ij}^+ or (ii) all k points in $R \cap (\bar{S} \setminus \{p_i, p_j\})$ are in T_{ij}^-. Since the points are chosen uniformly at random in the unit-square, any point that is in R has equal probability to be in T_{ij}^+ and T_{ij}^-. It follows that

$$\Pr(X_{ij} = 1 \mid E_k) \leq 2 \left(\frac{1}{2}\right)^k.$$

To analyze the probability that the event E_k holds, we define, for each point q in $\bar{S}$, $rk(q)$ to be the number of points in $\bar{Q}$ whose y-coordinates are less than or equal to the y-coordinate of q. (Thus, if q has the minimum y-coordinate, then $rk(q) = 1$.) The event E_k holds if and only if $|rk(p_i) - rk(p_j)| = k + 1$. Therefore,

$$\Pr(E_k) \leq \frac{2}{j - i}.$$

Thus, we have

$$\begin{aligned}\Pr(X_{ij}=1) &= \sum_k \Pr(X_{ij}=1 \mid E_k)\cdot\Pr(E_k)\\ &\le \sum_{k=0}^{\infty} 2\left(\frac{1}{2}\right)^k \cdot \frac{2}{j-i}\\ &= \frac{8}{j-i}.\end{aligned}$$

We have shown that

$$E(|S_t|) \le \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{8}{j-i} = O(n\log n).$$

□

Thus, for points that are randomly drawn from the unit-square, the expected size of the set $S_{\mathcal{D}}$ is $O(n\log n)$. Combining this with the discussion above, we have the following result.

Theorem 17. *Let S be a set of n points that are chosen independently and uniformly at random in the unit-square. For every constant $\epsilon > 0$, there exists a data structure of expected size $O(n^{1+\epsilon})$ such that for any query disk D, the closest pair in $S \cap D$ can be reported in $O(n^{2/3+\epsilon})$ expected time.*

5. Computing the closest pair "partially inside" a query rectangle

Let S be a set of n points in the plane. In this section, we consider queries of the following form: Given a query region $Q \subseteq \mathbb{R}^2$, report the pair (p, q), where $p \in S \cap Q$ and $q \in S \setminus Q$, whose distance is minimum.

Our solution is based on the following well-known property of the Euclidean Minimum Spanning Tree (EMST) of S. (For completeness, we give a proof here.)

Lemma 18. *Let T be an EMST of S, let $(S', S \setminus S')$ be any partition of S, and let L be the set of shortest line-segments among all line-segments that have one endpoint in S and the other in $S \setminus S'$. Then, at least one line-segment of L is an edge of T.*

PROOF. Assume that no line-segment of L is an edge of T. Let $\overline{pq}$ be any line-segment in L. The graph $T \cup \overline{pq}$ contains a cycle C that has p and q as vertices. Walk around C from p to q, using the edges that avoids $\overline{pq}$. At some point, the walk will traverse an edge $\overline{xy}$ of T, where x and y are on opposite sides of the partition. By assumption, $\overline{xy} \notin L$, so $d(x, y) > d(p, q)$. Observe that the graph $(T \cup \{\overline{pq}\}) \setminus \{\overline{xy}\}$ is a Euclidean spanning tree of S. It follows that there is a Euclidean spanning tree of S that has lower cost than T—a contradiction. □

Lemma 18 implies that, for our query problem, it suffices to restrict our attention to the $n-1$ point-pairs that define edges of the EMST of S (rather than considering all $\binom{n}{2}$ pairs). Specifically, if the boundary ∂Q of Q is a simple closed curve (so that the "inside" and "outside" of Q are well-defined by the Jordan Curve Theorem), then $\overline{pq}$ intersects ∂Q, where (p, q) is the desired closest pair.

Thus, our query problem becomes the following: Among all edges of the EMST of S that are intersected by ∂Q and have exactly one endpoint in Q, find the shortest edge.

We now consider queries where Q is an axes-parallel rectangle. Let e be an edge of the EMST of S and let (e_x, e_y) and (e'_x, e'_y) be its endpoints, where (e_x, e_y) is to the left of (e'_x, e'_y). Let $Q = [a, b] \times [c, d]$ be a query rectangle.

Lemma 19. *If ∂Q intersects e and Q contains exactly one endpoint of e, then at least one of the following holds:*

1. (e_x, e_y) *is in* Q *and* $e'_y > d$,
2. (e_x, e_y) *is in* Q *and* $e'_y < c$,
3. (e_x, e_y) *is in* Q *and* $e'_x > b$,
4. (e'_x, e'_y) *is in* Q *and* $e_y > d$,
5. (e'_x, e'_y) *is in* Q *and* $e_y < c$,
6. (e'_x, e'_y) *is in* Q *and* $e_x < a$.

PROOF. The first three cases correspond to e being intersected by the upper, lower, or right edge of ∂Q, with (e_x, e_y) in Q and (e'_x, e'_y) not in Q. The last three cases correspond to e being intersected by the upper, lower, or left edge of ∂Q, with (e'_x, e'_y) in Q and (e_x, e_y) not in Q. (See Figure 7.) These are the only possibilities. Observe that two of these cases can hold simultaneously, if e passes through a corner of Q. □

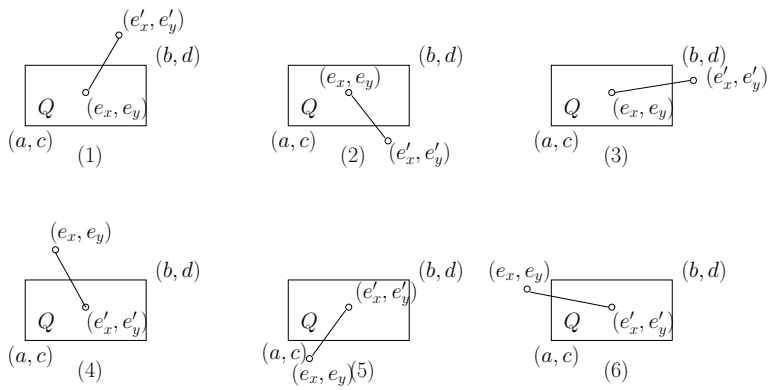

Figure 7: *The different cases in Lemma 19. Top row: Cases 1–3. Bottom row: Cases 4–6.*

Our data structure consists of four structures D_1, D_2, D_3, and D_4, which are defined as follows:

For each edge e of the EMST of S, we create the point (e_x, e_y, e'_y) in $\mathbb{R}^3$ and assign it a weight equal the length of e. Let S_1 be the resulting set of weighted points. The structure D_1 is a 3-level range tree on S_1, where the level-1 tree is built on the real numbers e_x, the level-2 trees are built on the real numbers e_y, and the level-3 trees are built on the real numbers e'_y. Each node of each level-3 tree stores the minimum of the weights of the points in its subtree.

Similarly, let S_2 be the set of weighted points (e_x, e_y, e'_x), S_3 the set of weighted points (e'_x, e'_y, e_y), and S_4 the set of weighted points (e'_x, e'_y, e_x). The structures D_2, D_3, and D_4 are built on S_2, S_3, and S_4, respectively, analogous to how D_1 is built on S_1.

The query algorithm consists of the following six queries, corresponding to the cases in Lemma 19:

1. D_1 is queried first with the 3-dimensional query rectangle $Q \times (d, \infty)$ and then with $Q \times (-\infty, c)$. (This covers cases 1 and 2.)
2. D_2 is queried with $Q \times (b, \infty)$. (This covers case 3.)
3. D_3 is queried first with $Q \times (d, \infty)$ and then with $Q \times (-\infty, c)$. (This covers cases 4 and 5.)
4. D_4 is queried with $Q \times (-\infty, a)$. (This covers case 6.)
5. Each of these six query returns a real number which is the length of the shortest edge of the EMST of S satisfying one of the six conditions

in Lemma 19. The smallest of these is returned as the answer to the original query.

In more detail, the query on D_1 with $Q \times (d, \infty)$ is done as follows. (The discussion for the other cases is similar, hence omitted.) The level-1 tree of D_1 is queried with the interval $[a, b]$ and a set of $O(\log n)$ canonical nodes is identified. The level-2 tree of each such node is queried with the interval $[c, d]$ to identify a total of $O(\log^2 n)$ canonical nodes in the level-2 trees. Finally, the level-3 tree of each of these nodes is queried with the interval (d, ∞), which yields a grand total of $O(\log^3 n)$ canonical nodes in the level-3 subtrees. The union of the points contained in the subtrees of these level-3 canonical nodes is exactly the set of points of S_1 that are in $Q \times (d, \infty)$. The minimum of the weights stored at the $O(\log^3 n)$ level-3 canonical nodes yields the desired answer, i.e., the shortest edge of the EMST of S that has exactly one endpoint in Q and intersects the upper edge of Q.

It is clear that each of the structures D_i, $1 \leq i \leq 4$, uses $O(n \log^2 n)$ space and that the query time is $O(\log^3 n)$. The query time can be improved to $O(\log^2 n)$, via the technique of *layering* (see de Berg *et al.* [6]): Again, considering D_1 for concreteness, we replace each level-3 tree by a sorted array storing the corresponding values e'_y. Moreover, for each suffix and each prefix of the array, we store the smallest weight among the points whose e'_y values appear in the suffix or prefix, respectively. We then apply the layering technique to these arrays. (A similar discussion applies to the other three structures, except that for D_2 we need to store only suffix minima and for D_4 only prefix minima.)

To answer a query on D_1 with $Q \times (d, \infty)$, we proceed as before to identify $O(\log^2 n)$ canonical nodes in level-2 trees. Each of these canonical nodes has an associated sorted array of values e'_y. Using the layering technique, we locate d in *all* of these arrays in a total of $O(\log n)$ additional time. For each array, we identify the suffix consisting of all values greater than d, identify the corresponding minimum weight, and report the smallest of all these weights. The query on D_1 with $Q \times (-\infty, c)$ is similar, except that we consider prefixes consisting of all values less than c. Thus the query time reduces to $O(\log^2 n)$, while the space bound remains $O(n \log^2 n)$.

Theorem 20. *A set S of n points in $\mathbb{R}^2$ can be stored in a data structure of size $O(n \log^2 n)$, such that for any axes-parallel query rectangle Q, the closest pair (p, q), where p is in $S \cap Q$ and q is in $S \setminus Q$, can be reported in $O(\log^2 n)$ time.*

Lemma 18 holds in any dimension $d > 2$, so the approach above can be generalized to axes-parallel query hyper-rectangles in $\mathbb{R}^d$. Specifically, the analog of Lemma 19 has $4d - 2$ conditions, each of which can be expressed as a $(d+1)$-dimensional range restriction and, hence, can be handled using a $(d+1)$-dimensional range tree, with layering at the innermost level. This yields the following generalization of Theorem 20.

Theorem 21. *Let S be a set of n points in $\mathbb{R}^d$, where $d \geq 2$ is a fixed constant. S can be stored in a data structure of size $O(n \log^d n)$, such that for any d-dimensional axes-parallel query hyper-rectangle Q, the closest pair (p, q), where p is in $S \cap Q$ and q is in $S \setminus Q$, can be reported in $O(\log^d n)$ time.*

Remark 2. The same approach can be used to answer queries with regions Q other than axes-parallel rectangles. Let S be a set of n points in the plane and consider a class $\mathcal{Q}$ of query regions. Assume that a tree-based data structure D exists such that for each $Q \in \mathcal{Q}$,

1. a "small" set C of canonical nodes in D can be computed such that $S \cap Q$ is equal to the union of the sets stored at the subtrees of the nodes in C, and
2. a "small" set C' of canonical nodes in D can be computed such that $S \setminus Q$ is equal to the union of the sets stored at the subtrees of the nodes in C'.

Then, we can extend this data structure such that queries of the form "report the closest pair (p, q), where p is in $S \cap Q$ and q is in $S \setminus Q$" can be answered: For each edge e of the EMST, consider its two endpoints p_e and q_e. Then we store the points p_e in the data structure D. For each node u in D, we store a pointer to a level-2 structure, which is again of type D, which stores all points q_e for which the corresponding point p_e is stored in the subtree of u.

This approach can be used for query regions such as halfplanes (resp. disks), using partition trees as the underlying structure, and leads to a solution that uses $O(n)$ space and has a query time of $O(n^{1/2+\epsilon})$ (resp. $O(n^{2/3+\epsilon})$, where $\epsilon > 0$ is a constant. In fact, this approach also generalizes to higher dimensions, with the bounds indicated in Table 1.

6. Computing the diameter in a query rectangle

Let S be a set of n points in the plane. In this section, we consider queries of the form: Given an axes-parallel rectangle Q, report the diameter of the set $S \cap Q$.

We choose a parameter k with $1 \leq k \leq n$. This parameter will give a trade-off between the query time and the space. Our data structure consists of the following.

1. We store the points of S in a range tree; see de Berg *et al.* [6]. The primary tree of this data structure stores the points at its leaves, sorted by their x-coordinates. Every node u in this primary tree stores a pointer to a secondary tree which stores the points of S_u at its leaves, sorted by their y-coordinates.
2. Each node v in each secondary tree in the range tree stores (i) the diameter of the set S_v and (ii) the furthest-point Voronoi diagram of the set S_v. Each furthest-point Voronoi diagram is preprocessed for point location queries; see Kirkpatrick [14].
3. Let B be the set of all nodes v such that v is a node in some secondary tree and $|S_v| \geq k$. We build a table $\mathcal{T}$, which stores, for each pair v, w of distinct vertices in B, the maximum distance between any point in S_v and any point in S_w.

We first analyze the size of the data structure. The total amount of space needed to store the range tree and the furthest-point Voronoi diagrams is $O(n \log^2 n)$. Since the set B has size $O((n/k) \log n)$, the entire data structure has size

$$O\left(n \log^2 n + |B|^2\right) = O\left(\left(n + (n/k)^2\right) \log^2 n\right).$$

Consider a query rectangle Q. The algorithm that finds the diameter of $S \cap Q$ does the following.

1. Compute a set C of $O(\log^2 n)$ canonical nodes v in the secondary structures of the range tree such that $S \cap Q = \cup_{v \in C} S_v$.
2. Each node v of C stores the diameter of the set S_v. Compute the maximum of these diameters.
3. For each pair v, w of distinct nodes in C, do the following. If both S_v and S_w have size at least k, then the table $\mathcal{T}$ stores the largest distance between any point in S_v and any point in S_w. Otherwise,

assume without loss of generality that $|S_v| < k$. Then, for each point p in S_v, we perform a point location query in the furthest-point Voronoi diagram of S_w and, thus, obtain the largest distance between any point in S_v and any point in S_w.

4. Return the largest distance found in these steps.

To prove the correctness of this query algorithm, let (p, q) be the diameter in $S \cap Q$. Let v and w be the nodes in C such that $p \in S_v$ and $q \in S_w$. If $v = w$, then (p, q) is found in Step 2. If $v \neq w$, then the diameter is found in Step 4. It is easy to see that the overall query time is

$$O\left(\log^2 n + (\log^4 n) \cdot k \log n\right).$$

Theorem 22. *Let S be a set of n points in the plane and let k be an integer with $1 \leq k \leq n$. The set S can be preprocessed into a data structure of size $O((n + (n/k)^2) \log^2 n)$ such that for any axes-parallel query rectangle Q, the diameter of $S \cap Q$ can be reported in $O(k \log^5 n)$ time.*

7. Approximating the diameter in a query rectangle

Let S be a set of n points in the plane and let δ be a real number with $0 < \delta < 1$. In this section, we consider queries of the form: Given an axes-parallel rectangle Q, report a $(1 - \delta)$-approximation to the diameter of the set $S \cap Q$. That is, report a pair of points in $S \cap Q$ whose distance is at least $1 - \delta$ times the diameter of $S \cap Q$.

We first consider the case when δ is fixed, i.e., the approximation factor is the same for every query. Let

$$\beta(\delta) = \left\lceil \frac{2 \arcsin(1 - \delta)}{\pi - 2 \arcsin(1 - \delta)} \right\rceil.$$

Observe that $\beta(\delta) = O(1/\sqrt{\delta})$ if δ converges to zero.

The following lemma states that the diameter of S can be approximated by considering only $O(\beta(\delta))$ pairs of points in S. The lemma is adapted from Janardan [13].

Lemma 23. *Choose $2(\beta(\delta)+1)$ equally-spaced vectors around the unit-circle. For each such vector d_i, let p_i be the point of S that is extreme in direction d_i, and let q_i be the point of S that is extreme in direction $-d_i$. Let D be the diameter of S and let $\Delta = \max_i d(p_i, q_i)$. Then $1 - \delta \leq \Delta/D \leq 1$.*

The data structure for our query problem is a range tree; see de Berg *et al.* [6]. With each node v in each secondary tree, we store the $O(\beta(\delta)) = O(1/\sqrt{\delta})$ point-pairs of Lemma 23 for the set S_v. Then, the total size of the data structure is $O((1/\sqrt{\delta})n \log n)$.

Given an axes-parallel query rectangle Q, we compute a set C of $O(\log^2 n)$ canonical nodes v in the secondary structures of the range tree such that $S \cap Q = \cup_{v \in C} S_v$. Consider any of the $O(\beta(\delta))$ direction pairs d_i and $-d_i$. We can compute the extreme points of $S \cap Q$ in directions d_i and $-d_i$ by computing the extreme points among the extreme points stored with the canonical nodes for this direction pair. By Lemma 23, the farthest pair so computed over all direction pairs is an approximation to the diameter of $S \cap Q$.

Theorem 24. *Let S be a set of n points in the plane and let δ be a real number with $0 < \delta < 1$. The set S can be preprocessed into a data structure of size $O((1/\sqrt{\delta})n \log n)$ such that for any axes-parallel query rectangle Q, a $(1-\delta)$-approximation to the diameter of $S \cap Q$ can be reported in $O((1/\sqrt{\delta}) \log^2 n)$ time.*

The data structure in Theorem 24 depends on δ. We now show how to obtain a data structure that does not depend on δ and that can be used to answer queries in which δ comes as an input parameter together with the query rectangle Q.

We again use a range tree for the points of S. Now, every secondary node v stores the convex hull of the point set S_v. The size of this structure is $O(n \log^2 n)$.

Given a query rectangle Q, we find a set C of $O(\log^2 n)$ canonical nodes as above. Observe that each node v of C stores the convex hull of the set S_v. Thus, by merging the $O(\log^2 n)$ convex hulls of the sets S_v, with $v \in C$, we obtain the convex hull of the set $S \cap Q$. Then we use the latter convex hull to compute the $O(1/\delta)$ extremal pairs of $S \cap Q$. Recall that (i) the convex hull of two convex polygons can be computed in logarithmic time and (ii) the extreme point of a convex polygon for a given direction can be computed in logarithmic time; see Preparata and Shamos [18]. It follows that the total time of this query algorithm is $O(\log^3 n + (1/\sqrt{\delta}) \log n)$.

Theorem 25. *Let S be a set of n points in the plane. The set S can be preprocessed into a data structure of size $O(n \log^2 n)$ such that for any axes-parallel query rectangle Q and any real number δ with $0 < \delta < 1$, a $(1 -$*

δ)-approximation to the diameter of $S \cap Q$ can be reported in $O(\log^3 n + (1/\sqrt{\delta}) \log n)$ time.

8. Approximating the width in a query rectangle

Recall that the *width* of a point-set is the width of a narrowest strip that encloses the point-set. Let S be a set of n points in the plane and let δ be a real number with $0 < \delta < 1$. In this section, we consider queries of the following form: Given an axes-parallel query rectangle Q, report a $(1+\delta)$-approximation to the width of the points in $S \cap Q$. That is, report a strip that encloses the points in $S \cap Q$ whose width is at most $1+\delta$ times the width of $S \cap Q$.

We consider only the case when δ is fixed. Thus, the approximation factor is the same for all queries. Define

$$\gamma(\delta) = \left\lceil \frac{\pi}{2 \arccos(1/(1+\delta))} \right\rceil .$$

Observe that $\gamma(\delta) = O(1/\sqrt{\delta})$ if δ converges to zero.

Our approach is based on the following lemma, which is adapted from Janardan [13].

Lemma 26. *Let $S_0 = S$ and for $1 \le i \le \gamma(\delta)$, let S_i be a copy of S that is rotated clockwise around the origin from S_{i-1} by an angle of $\pi/\gamma(\delta)$. For $0 \le i \le \gamma(\delta)$, let L_i be the downward convex chain that is dual to the upper hull of the convex hull of S_i, and let R_i be the upward convex chain that is dual to the lower hull of the convex hull of S_i. (L_i is strictly above R_i.) Let ω_i^L be the minimum distance between any vertex of L_i and any point vertically below it on R_i, and let ω_i^R be the minimum distance between any vertex of R_i and any point vertically above it on L_i. Let $\Omega = \min_i\{\omega_i^L, \omega_i^R\}$ and let W be the width of S. Then $1 \le \Omega/W \le 1+\delta$.*

As discussed in [13], the distances ω_i^L and ω_i^R can be computed in $O(\log^2 n)$ time, via binary search, if the chains L_i and R_i are stored in balanced binary search trees.

The data structure for our query problem is as follows: We store S in a range tree. Each node v in each secondary tree stores $1 + \gamma(\delta)$ instances of the data structure underlying Lemma 26 for the set S_v; specifically, the i-th instance is a pair of balanced binary search trees built on the dual chains

$L_i(v)$ and $R_i(v)$ associated with the i-th rotated copy of S_v. The total size of the data structure is $O((1/\sqrt{\delta})n \log^2 n)$.

Given an axes-parallel query rectangle Q, we compute, in $O(\log^2 n)$ time, a set C of $O(\log^2 n)$ canonical nodes v in the secondary structures of the range tree such that $S \cap Q = \cup_{v \in C} S_v$. Then, for each i, taken in turn, we merge the $L_i(v)$'s for all $v \in C$ into a single chain. This can be done in $O(\log n)$ time per merge by first merging the chains of secondary canonical nodes associated with the same primary canonical nodes, and then merging the chains for the different primary canonical nodes (since all resulting chains will be disjoint). Similarly, for the chains $R_i(v)$. This step takes $O((1/\sqrt{\delta}) \log^3 n)$ time.

From the resulting $O(1/\sqrt{\delta})$ pairs of chains, we compute the minimum vertical distance between each pair and obtain the smallest of these distances as Ω. By Lemma 26, this is an approximation to the width of S.

Theorem 27. *Let S be a set of n points in the plane and let δ be a real number with $0 < \delta < 1$. The set S can be preprocessed into a data structure of size $O((1/\sqrt{\delta})n \log^2 n)$ such that for any axes-parallel query rectangle Q, a $(1+\delta)$-approximation to the width of $S \cap Q$ can be reported in $O((1/\sqrt{\delta}) \log^3 n)$ time.*

9. Concluding remarks

We have considered geometric range-aggregate query problems, where the goal is to generate an informative summary of the objects (e.g., points) that are contained in a query object (rather than simply report or count the objects), by applying a suitable aggregation function on the retrieved objects. The specific aggregation functions considered here are closest pair, farthest pair, and width, which measure the "spread" of the retrieved objects. These problems are challenging since they are not decomposable in the usual sense. Nevertheless, we have been able to design efficient algorithms for several of them, using a range of techniques from computational geometry.

Several open problems are worth investigating. These include devising an efficient algorithm for range-aggregate closest pair queries in higher dimensions, solving the planar range-aggregate width problem exactly, and developing solutions for the range-aggregate diameter and width problems in higher dimensions.

After we wrote a preliminary version of this paper, Abam *et al.* [1] showed how (slight variants of) the data structures underlying Lemmas 1 and 4 can

be built in $O(n \cdot polylog(n))$ time. As a result, they obtain a data structure for planar range-aggregate closest pair queries having the same performance as in Theorem 9 and that can be built in $O(n \log^5 n)$ time.

Acknowledgement

The authors thank an anonymous referee for carefully reading a preliminary version of this paper and for pointing us out to reference [11].

References

[1] M. A. Abam, P. Carmi, M. Farshi, and M. Smid. On the power of the semi-separated pair decomposition. In *Proceedings of the 11th Algorithms and Data Structures Symposium (WADS)*, *Lecture Notes in Computer Science*, Berlin, 2009, Springer-Verlag, to appear.

[2] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages 1–56. American Mathematical Society, Providence, RI, 1999.

[3] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. ACM*, 51:606–635, 2004.

[4] P. Bose, E. Kranakis, P. Morin, and Y. Tang. Approximate range mode and range median queries. In *Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science*, volume 3404 of *Lecture Notes in Computer Science*, pages 377–388, Berlin, 2005. Springer-Verlag.

[5] D. Datta and S. Soundaralakshmi. An efficient algorithm for computing the maximum empty rectangle in three dimensions. *Information Sciences*, 128:43–65, 2000.

[6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 2nd edition, 2000.

[7] S. Felsner. Empty rectangles and graph dimension. arXiv:math/0601767v1, 2006. http://arxiv.org/abs/math/0601767v1.

[8] H. Gabow, J.Bentley, and R. Tarjan. Scaling and related techniques for geometry problems. In *Proceedings of the 16th ACM Symposium on Theory of Computing*, pages 135–142, 1984.

[9] S. Govindarajan, P. K. Agarwal, and L. Arge. CRB-tree: An efficient indexing scheme for range aggregate queries. In *Proceedings of the 9th International Conference on Database Theory*, pages 143–157, 2003.

[10] 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, Berlin, 2005. Springer-Verlag.

[11] R. H. Güting, O. Nurmi, and T. Ottmann. The direct dominance problem. In *Proc. 1st Annu. ACM Sympos. Comput. Geom.*, pages 81–88, 1985.

[12] S. Hong, B. Song, and S. Lee. Efficient execution of range-aggregate queries in data warehouse environments. In *Proceedings of the 20th International Conference on Conceptual Modeling*, volume 2224 of *Lecture Notes in Computer Science*, pages 299–310, Berlin, 2001. Springer-Verlag.

[13] R. Janardan. On maintaining the width and diameter of a planar point-set online. *International Journal of Computational Geometry & Applications*, 3:331–344, 1993.

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

[15] R. Klein. Direct dominance of points. *International Journal of Computer Mathematics*, 19:225–244, 1987.

[16] D. Krizanc, P. Morin, and M. Smid. Range mode and range median queries on lists and trees. *Nordic Journal of Computing*, 12:1–17, 2005.

[17] M. Mitzenmacher and E. Upfal. *Probability and Computing*. Cambridge University Press, Cambridge, UK, 2005.

[18] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, Berlin, 1988.

[19] 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, Berlin, 2003. Springer-Verlag.

[20] R. Sharathkumar and P. Gupta. Range-aggregate proximity detection for design rule checking in VLSI layouts. In *Proceedings of the 18th Canadian Conference on Computational Geometry*, pages 151–154, 2006.

[21] R. Sharathkumar and P. Gupta. Range-aggregate proximity queries. Technical Report IIIT/TR/2007/80, International Institute of Information Technology Hyderabad, `http://www.iiit.net/techreports/2007_80.pdf`, 2007.

[22] S. Shekhar and S. Chawla. *Spatial Databases: A Tour*. Prentice Hall, 2002.

[23] T. G. Szymanski and C. J. van Wyk. Layout analysis and verification. In B. Preas and M. Lorenzetti, editors, *Physical Design Automation of VLSI Systems*, pages 347–407. Benjamin/Cummins, 1988.

[24] Y. Tao and D. Papadias. Range aggregate processing in spatial databases. *IEEE Transactions on Knowledge and Data Engineering*, 16:1555–1570, 2004.

[25] D. Zhang, A. Markowetz, V. Tsotras, D. Gunopulos, and B. Seeger. Efficient computation of temporal aggregates with range predicates. In *Proceedings of the 20th Symposium on Principles of Database Systems*, pages 237–245, 2001.

[26] D. Zhang and V. J. Tsotras. Improving min/max aggregation over spatial objects. *VLDB Journal*, 14:170–181, 2005.

[27] D. Zhang, V. J. Tsotras, and D. Gunopulos. Efficient aggregation over objects with extent. In *Proceedings of the 21st Symposium on Principles of Database Systems*, pages 121–132, 2002.