# Truly Subquadratic Time Algorithms for Diameter and Related Problems in Graphs of Bounded VC-dimension

Timothy M. Chan* Hsien-Chih Chang† Jie Gao‡ Sándor Kisfaludi-Bak§
Hung Le¶ Da Wei Zheng‖

January 4, 2026

**Abstract**

We give the first truly subquadratic time algorithm, with $\widetilde{O}(n^{2-1/18})$ running time, for computing the diameter of an $n$-vertex unit-disk graph, resolving a central open problem in the literature. Our result is obtained as an instance of a general framework, applicable to different graph families and distance problems. Surprisingly, our framework completely bypasses sublinear separators (or $r$-divisions) which were used in all previous algorithms. Instead, we use *low-diameter decompositions* in their most elementary form. We also exploit *bounded VC-dimension* of set systems associated with the input graph, as well as new ideas on *geometric data structures*. Among the numerous applications of the general framework, we obtain:

1. An $\widetilde{O}(mn^{1-1/(2d)})$ time algorithm for computing the diameter of $m$-edge sparse unweighted graphs with constant VC-dimension $d$. The previously known algorithms by Ducoffe, Habib, and Viennot *[SODA 2019]* and Duraj, Konieczny, and Potępa *[ESA 2024]* are truly subquadratic only when the diameter is a small polynomial. Our result thus generalizes truly subquadratic time algorithms known for planar and minor-free graphs (in fact, it slightly improves the previous time bound for minor-free graphs).
2. An $\widetilde{O}(n^{2-1/12})$ time algorithm for computing the diameter of intersection graphs of axis-aligned squares with arbitrary size. The best-known algorithm by Duraj, Konieczny, and Potępa *[ESA 2024]* only works for unit squares and is only truly subquadratic in the low-diameter regime.
3. The first algorithms with truly subquadratic complexity for other distance-related problems, including all-vertex eccentricities, Wiener index, and exact distance oracles. In particular, we obtain the first exact distance oracle with truly subquadratic space and $\widetilde{O}(1)$ query time for any sparse graph with bounded VC-dimension, again generalizing previous results for planar and minor-free graphs.

*Siebel School of Computing and Data Science, University of Illinois at Urbana-Champaign. Email: tmc@illinois.edu. Supported by NSF grant CCF-2224271.

†Department of Computer Science, Dartmouth College. Email: hsien-chih.chang@dartmouth.edu.

‡Department of Computer Science, Rutgers University. Email: jg1555@rutgers.edu. Gao would like to acknowledge NSF support through CNS-2515159, IIS-2229876, DMS-2220271, DMS-2311064, CCF-2208663, CCF-2118953.

§Department of Computer Science, Aalto University, Finland. Email: sandor.kisfaludi-bak@aalto.fi. Supported by the Research Council of Finland, Grant 363444.

¶Manning CICS, UMass Amherst. Email: hungle@cs.umass.edu. Supported by NSF grants CCF-2517033 and CCF-2121952, NSF CAREER Award CCF-2237288, and a Google Faculty Research Award.

‖Institute of Science and Technology Austria, Klosterneuburg, Austria. Work in this paper was done while at the Siebel School of Computing and Data Science, University of Illinois at Urbana-Champaign.

# Contents


**1 Introduction** **1**
- 1.1 Main Results on Diameter . . . 2
- 1.2 Technical Overview . . . 2
- 1.3 Other Distance-related Problems . . . 7

**2 Preliminaries** **9**
- 2.1 Graphs and Low-diameter Decomposition . . . 9
- 2.2 VC-dimension . . . 10
- 2.3 Stabbing Path and Interval Representation . . . 12
- 2.4 Geometric Data Structures . . . 12
- 2.5 Handling of Small Pieces . . . 13

**3 Framework for Diameter and Eccentricities** **14**

**4 Diameter/Eccentricities in Sparse Graphs of Bounded VC-dimension** **16**

**5 Diameter/Eccentricities in Square Graphs** **17**

**6 Diameter/Eccentricities in Unit-square Graphs** **20**

**7 Diameter/Eccentricities in Unit-disk Graphs** **20**
- 7.1 Restriction to Fixed Types . . . 21
- 7.2 Implementation of the Neighborhood Growing Step . . . 22
- 7.3 Analysis for Eccentricities . . . 26
- 7.4 Analysis for Diameter . . . 27

**8 Framework for Distance Oracles (and Wiener Index)** **28**

**9 Distance Oracles for Sparse Graphs of Bounded VC-dimension** **30**

**10 Distance Oracles for Square Graphs** **31**

**11 Distance Oracles for Unit-square Graphs** **32**

**12 Distance Oracles for Unit-disk Graphs** **33**

**13 Conclusion and Open Questions** **34**

**A Low-diameter Decompositions** **39**
- A.1 Sparse Graphs . . . 40
- A.2 Geometric Intersection Graphs . . . 40

**B VC-dimension Lemma** **42**
- B.1 Type-1 $M$-Walks . . . 43
- B.2 Type-2 $M$-Walks . . . 45

**C Geometric Data Structures** **45**
C.1 Reductions Between Data Structure Problems . . . . . 45
C.2 Data Structure for Square Graphs . . . . . 48
C.3 Data Structure for Unit Disks . . . . . 50
C.4 Data Structure for Unit Squares . . . . . 52

**D Switching Interval Representation between Different Stabbing Paths** **56**

**E Handling Small Pieces** **58**
E.1 Patterns . . . . . 58
E.2 Diameter and Eccentricities using Patterns . . . . . 58
E.3 Boundary Weighted BFS in Geometric Intersection Graphs . . . . . 59
E.4 Exact Distance Oracles . . . . . 60

# 1 Introduction

A simple algorithm for computing the diameter of an unweighted $n$-vertex graph is to run a BFS from every vertex of the graph. For sparse graphs or intersection graphs of various classes of geometric objects (such as unit disks), BFS can be implemented in $\widetilde{O}(n)$ time, leading to an algorithm to compute the graph diameter in $\widetilde{O}(n^2)$ time[1]. Can we beat this naïve quadratic-time algorithm? More precisely, can we compute the diameter in *truly subquadratic time* $O(n^{2-\varepsilon})$ for some fixed constant $\varepsilon > 0$ for these graphs? This simple question has motivated the development of a broad range of techniques that have driven algorithmic research for decades.

For general sparse graphs, even distinguishing the diameter between 2 and 3 in truly subquadratic time is impossible, assuming the Strong Exponential Time Hypothesis (SETH) [RW13]. (For dense undirected graphs, one can exploit the matrix multiplication subroutine to compute the diameter in $O(n^\omega)$ time [Sei95], where $\omega < 2.371339$ is the matrix multiplication exponent [ADW+25].) Given the negative result, it is natural to consider more structured classes of sparse graphs, such as planar and minor-free graphs. For planar graphs, Cabello [Cab18] designed the first truly subquadratic algorithm for the diameter problem by introducing a new technique based on abstract Voronoi diagrams. This technique heavily exploits planarity and hence fails for minor-free graphs. Then Ducoffe, Habib, and Viennot [DHV22] devised a new technique based on VC-dimension to compute the diameter of minor-free graphs in truly subquadratic time. Both the Voronoi diagram and VC-dimension techniques are major milestones in algorithm design for planar and minor-free graphs, opening the door for solving other distance-related problems in truly subquadratic complexity (time or space), such as designing compact (exact) distance oracles and computing eccentricities or Wiener index in planar and minor-free graphs [Cab18, GKM+21, LP19, DHV22, LW24, KZ25].

For geometric intersection graphs of objects in the plane, designing a truly subquadratic time algorithm for the diameter problem has been much more challenging. A *geometric intersection graph* is a graph whose vertices are associated with objects in the plane, and edges correspond to object pairs that intersect.[2] Unit-disk graphs—the intersection graphs of unit disks—are among the most basic and well-studied graphs in the geometric setting; alternatively, this is equivalent to constructing an unweighted graph based on a set of points in a metric space by connecting pairs of points whose distance is below some fixed threshold.

While truly subquadratic algorithms have been ruled out for intersection graphs of unit segments, unit equilateral triangles, or unit balls (in 3D) under standard fine-grained complexity assumptions [BKK+22], the lower bound techniques for these objects fail for unit disks. Therefore, computing diameter for unit-disk graphs in truly subquadratic time has become a central open problem raised by many authors [CS16, BKK+22, DKP24, CGL24]. Such an algorithm points to a larger landscape where truly subquadratic results for basic geometric intersection graphs are possible. We note that even distinguishing the diameter between 2 and 3 in truly subquadratic time for unit-disk graphs remains open.

**Question 1.1.** *Can one compute the diameter of unit-disk graphs in truly-subquadratic time?*

Unlike planar graphs which are sparse, unit-disk graphs (and intersection graphs in general) can be dense: they can contain cliques of arbitrary size. Even computing the BFS tree in $\widetilde{O}(n)$ time becomes non-trivial [CJ15]. Recently, Chang, Gao, and Le [CGL24] ported the VC-dimension technique for computing diameter of minor-free graphs to unit-disk graphs; similar to planar graphs, on a unit-disk graph, the radius-$r$ balls for all integer values $r$ also have bounded VC-dimension. A one-sentence summary of their technique is that they treated a (possibly large) clique as a single vertex, and designed a clique-based

[1] Throughout this paper, $\widetilde{O}(\cdot)$ notation hides polylogarithmic factors, and $O^*(\cdot)$ hides $n^{o(1)}$ factors.

[2] We represent an intersection graph by the objects themselves, so the input size is $O(n)$ even if the graph could be dense.

separator hierarchy [dKMT23]. As a result, they obtained a subquadratic ($\widetilde{O}(n^{2-1/18})$-time) algorithm that could only compute an approximation of the diameter with an additive error at most 1 in unit-disk graphs. While the additive error is very small, their algorithm falls short of distinguishing between diameters 2 and 3. This suggests that computing the diameter *exactly* for unit-disk graphs requires a very different approach. (There are many examples in the general graph literature where allowing a small constant additive approximation can make the problem significantly easier to solve; for example, see [ACIM99].) For exact algorithms, Duraj, Konieczny, and Potępa [DKP24] adapted the technique by Ducoffe, Habib, and Viennot [DHV22], which is also based on VC-dimension and a *stabbing path data structure*, to the intersection graph of *unit squares*.[3] However, their technique only works when the true diameter is small $D = O(n^{1/4-\varepsilon})$ [DKP24] and more importantly, their stabbing path data structure does not work for unit disks, or even (non-unit) square graphs, as they heavily exploit the nice geometry of unit squares. (In fact, they explicitly asked, even when the diameter is a constant, if the diameter of a unit-disk graph can be computed in truly subquadratic time.)

## 1.1 Main Results on Diameter

In this paper, we give the first truly subquadratic algorithm for computing the diameter in unit-disk graphs, resolving Question 1.1 affirmatively. Moreover, our framework has many other applications and yields the first truly subquadratic algorithms for the intersection graph of axis-aligned (arbitrarily sized) squares, as well as arbitrary sparse graphs with bounded VC-dimension.

**Theorem 1.2.** *Let $G$ be a graph on $n$ vertices. We can compute the diameter of $G$ by Las Vegas randomized algorithms in:*

- $O^*(n^{2-1/18})$ *time if $G$ is the intersection graph of unit disks, and*
- $\widetilde{O}(n^{2-1/12})$ *time if $G$ is the intersection graph of axis-aligned squares. For unit-square graphs, the running time is $O^*(n^{2-1/8})$.*
- $\widetilde{O}(mn^{1-1/(2d)})$ *time if $G$ has $m$ edges and VC-dimension $d$. For the special case of $K_h$-minor-free graphs for a fixed $h$, the running time becomes $\widetilde{O}(n^{2-1/(2h-2)})$.*

See Table 1 for the summary of our results on the diameter problem in comparison with previous work. (Incidentally, our result even slightly improves previous time bounds in the special case of $K_h$-minor-free graphs. The fact that the exponent of our algorithm for unit disks is the same as in Chang, Gao, and Le's $+1$-approximation algorithm [CGL24] is a complete coincidence—the algorithms are very different.)

## 1.2 Technical Overview

All previous subquadratic diameter algorithms for planar and minor-free graphs for arbitrary diameters [Cab18, GKM+21, LW24, DHV22, CGL24] use sublinear separators (or $r$-divisions), which are not available for geometric intersection graphs that could be dense. A key highlight of our framework is that we completely bypass sublinear separators! Instead, we use *low-diameter decompositions (LDD)*. LDDs have been used in recent breakthrough results, such as negative-weight shortest paths [BNW22] and $(2-\varepsilon)$-approximation for vertex cover on string graphs [LPS+24] (see the references in [BNW22] for more background). We stress that we only need the most elementary, non-probabilistic form of LDDs (dating back to [Awe85]), which are constructible simply by a number of "truncated" BFSes, and do not require expanders or flows. In some ways, they are even simpler than planar-graph separators or $r$-divisions.

[3] All squares are axis-aligned in this paper.

| graph class | best previous | | new |
|---|---|---|---|
| planar | $\widetilde{O}(n^{5/3})$ | [Cab18, GKM+21] | |
| $K_h$-minor-free | $\widetilde{O}(n^{2-1/(3h-1)})$ | [DHV22, LW24] | $\widetilde{O}(n^{2-1/(2h-2)})$ |
| VC-dim.-bounded | $\widetilde{O}(\min\{Dmn^{1-1/d}, mn\})$ | [DHV22, DKP24] | $\widetilde{O}(mn^{1-1/(2d)})$ |
| unit-square | $\widetilde{O}(\min\{Dn^{7/4}, n^2\})$ | [DKP24] | $O^*(n^{2-1/8})$ |
| square | $\widetilde{O}(n^2)$ | [CS19] | $\widetilde{O}(n^{2-1/12})$ |
| unit-disk | $O(n^2\sqrt{\frac{\log\log n}{\log n}})$ | [CS16] | $O^*(n^{2-1/18})$ |

**Table 1.** Time bounds of exact diameter algorithms for different classes of unweighted graphs. Here, $n$ is the number of vertices, $m$ is the number of edges, $D$ is the diameter, and $d$ is the (generalized distance) VC-dimension. Squares are axis-aligned.

In addition to LDD, our framework incorporates many new ideas about the usage of *bounded VC-dimension* as well as the design of *geometric data structures*. We will describe all three components of our framework in a little more detail below.

**Component 1: Low-diameter decomposition.** For a given parameter $\Delta > 0$, a *low-diameter decomposition (LDD)* decomposes the input graph into *pieces* of diameter at most $\Delta$ such that the total number of boundary vertices of all the pieces is $\widetilde{O}(n/\Delta)$. (It is helpful to imagine choosing $\Delta = n^\delta$ for some small constant $\delta$, and hence the number of boundary vertices is truly sublinear.) The ability to control the total number of boundary vertices is reminiscent of $r$-division [Fre87] used for diameter computation in planar [Cab18, GKM+21] and minor-free graphs [LW22], but an important difference is that a piece in an LDD could have up to $\Omega(n)$ vertices, while in an $r$-division, every piece has truly sublinear size (for a typical choice of $r$). LDDs can be computed in $\widetilde{O}(m)$ time for general graphs and $\widetilde{O}(n)$ time for many classes of intersection graphs, as we will show (in Appendix A).

**Component 2: Bounded VC-dimension and stabbing paths.** Since any sparse graph has a good low-diameter decomposition, an LDD itself is not sufficient for constructing truly subquadratic algorithms due to the aforementioned conditional lower bound based on SETH [RW13]. A recent line of work on the diameter problem has hinted at bounded VC-dimension as an overarching property: planar graphs (more generally, minor-free graphs) [CEV07, BT15, LP19, DHV22, LW24] and intersection graphs of pseudo-disks [ACM+21, DKP24, CGL24] (in particular, disks and squares) have bounded VC-dimension. Thus, we also assume that the input graph has a bounded VC-dimension.

Given a set system $(U, \mathcal{F})$ with a ground set $U$ and a family $\mathcal{F}$ of subsets of $U$, its *VC-dimension* is the cardinality of the largest $S \subseteq U$ such that $S$ is *shattered* by $\mathcal{F}$—for every $S' \subseteq S$, there is some $X \in \mathcal{F}$ such that $X \cap S = S'$. Given a graph $G$, there are several different ways to form a set system of bounded VC-dimension; see Section 2. The simplest one is the set system of neighborhood balls $(V_G, \{N^r[v]\}_{r\geq 0})$: we say that a graph $G$ has *VC-dimension*[4] at most $d$ if its system of neighborhood balls has VC-dimension at most $d$. ($N^r[v]$ is the set of all vertices that are at a distance at most $r$ from $v$, including $v$ itself.) It was known that planar graphs have VC-dimension at most 4; $K_h$-minor-free graphs have VC-dimension at most $h-1$; and intersection graphs of pseudo-disks have VC-dimension at most 4 [CGL24].

There are two main ways that VC-dimension was used in the diameter computation: (1) *stabbing path*: constructing a path that stabs each neighborhood ball $N^r[v]$ a sublinear number of times (in the worst case or on average), and (2) *distance compression:* showing that there are few different distance

[4] A more precise terminology is *distance VC-dimension* at most $d$; see Section 2 for clarification.

vectors to a fixed set of important vertices (i.e., the boundary of a piece in an $r$-division). The first approach has been very successful in the *low-diameter regime*: computing the diameter in time $\widetilde{O}(Dn^{2-\varepsilon_d})$ where $\varepsilon_d$ is a constant depending on the VC-dimension $d$ [DHV22, DKP24]. The second approach works for the arbitrary-diameter regime, but either requires sublinear separators [LW24] or allows distance approximation [CGL24]. We overcome the limitation and inherent obstacles from both approaches and devise a method in the presence of low-diameter decomposition to compute stabbing paths even when the graph diameter is large. (In certain applications, we also manage to perform distance compression exactly without the presence of separators.)

The basic idea of the stabbing path approach is to order the vertices from 1 to $n$, in such a way that each neighborhood ball $N^r[v]$ of radius $r$ can be represented as a union of $\widetilde{O}(n^{1-1/d})$ many intervals on the stabbing path.[5] The existence of a spanning path with $O(n^{1-1/d})$ stabbing (or "crossing") number was first shown in a seminal paper by Chazelle and Welzl [CW89], and had found numerous applications in computational geometry, for example, in geometric range searching. Constructing a good stabbing path may seem to require knowledge of the entire set system of balls $N^r[v]$ in the first place (which we do not have, since our problem is to compute all $N^r[v]$!). Fortunately, it turns out that by known random sampling techniques[6], we only need to evaluate a small subset of balls to compute a good stabbing path; for example, in the unit disk or square case, the construction time is $\widetilde{O}(n^{1+1/d})$ (more generally, the construction time is $\widetilde{O}(n\rho)$ for stabbing number $\widetilde{O}(n/\rho + \rho^{d-1})$ for a trade-off parameter $\rho$).

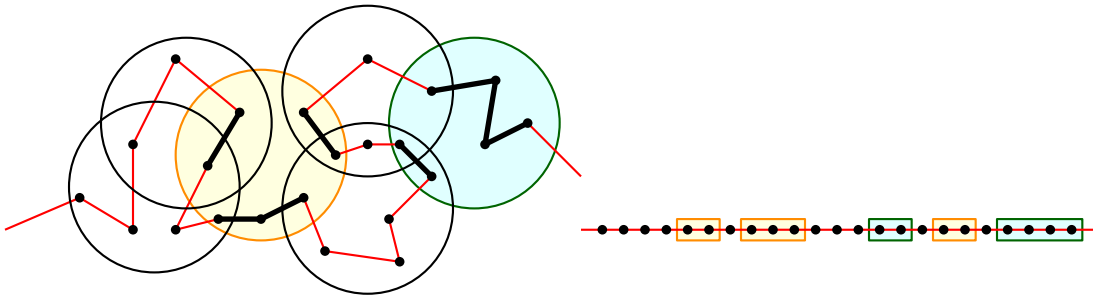

**Figure 1.** Stabbing path and interval representation of disks. The yellow disk is represented by three yellow intervals, and the green disk is represented by two intervals. The intervals representing different disks could overlap.

Let's say $v$ is one vertex in a diametral pair, whose shortest path distance realizes the diameter $D$. Given the interval representation, we can check in time linear to the number of intervals ($\widetilde{O}(n^{1-1/d})$) whether the union of all the intervals (and hence $N^r[v]$) covers $[1:n]$. If the answer is yes, then $r$ is at least the diameter $D$. By iterating through every vertex $v$ as a potential endpoint of a diametral path, we can check if $r$ is greater than the true diameter $D$ in $\widetilde{O}(n^{2-1/d})$ time. To compute the interval representations of $N^r[v]$ for all vertices in $V$, the pioneering work of Ducoffe, Habib, and Viennot [DHV22] introduced a *ball growing* process: For each $r$, one computes the interval representations of $\{N^r[v] : v \in V\}$ from the interval representations of $\{N^{r-1}[v] : v \in V\}$ via the identity $N^r[v] = \bigcup_{u \in N[v]} \hat{N}^{r-1}[u]$. (For the base case $r = 0$, $N^0[v] = \{v\}$.) This approach leads to running time $\widetilde{O}(Dmn^{1-1/d})$ for sparse graphs [DKP24]. For geometric intersection graphs, we cannot afford to access the neighbors of every vertex as it would result in $\Omega(n^2)$ time, and hence, different ideas are needed to avoid explicitly accessing neighbors. For the intersection graphs of unit squares, Duraj, Konieczny, and Potępa [DKP24] devised a certain "neighbor-set data structure" to achieve total running time $\widetilde{O}(Dn^{2-1/4})$. The factor of $D$ in the running time seems inherent to this approach, and $D$ could be as big as $\Omega(n)$. Furthermore, their data structure does not work for arbitrary squares or unit disks.

[5] For a simpler exposition, we assume the worst-case bound on $\widetilde{O}(n^{1-1/d})$ on the number of intervals representing $N^r[v]$. In the detailed implementation, we work with an *amortized bound* which is faster to compute.

[6] This is where all our algorithms use Las Vegas randomization.

To handle possibly large $D$, our new approach is to *combine with low-diameter decomposition*. First, by computing BFS trees from the boundary vertices of the LDD with parameter $\Delta$, we could compute an estimated value $\tau \in [D-\Delta : D]$. Note that there are only a truly sublinear number of boundary vertices, and hence, the total running time of this step remains truly subquadratic. (In the case of geometric intersection graphs, we can implement BFS in $\widetilde{O}(n)$ time using known techniques such as bichromatic intersections [CS16, Klo23, CGL24]. See Appendix A.2 for details.) If we have $\{N^{\tau}[v] : v \in V\}$, then we only need to grow balls for another $\Delta$ iterations (here $\Delta \ll D$). However, we do not have access to $\{N^{\tau}[v] : v \in V\}$ explicitly. Our key idea here is to define a *modified* neighborhood ball $\hat{N}^{r}[v]$ in a way that we can initialize $\hat{N}^{\tau}[v] = \varnothing$ to kick-start the ball growing process, and at the same time the information computed is sufficient to answer the question about diameter $D$. Therefore, the precise definition of $\hat{N}^{r}[v]$ is somewhat tricky, tying directly to the pieces in the LDD; see Section 3 for the details. For sparse graphs (with bounded VC-dimension), we can afford to access the neighbors of every vertex explicitly. Hence, we could simply grow the modified balls $\{\hat{N}^{r}[v]\}$ in $O(\Delta)$ rounds in $\widetilde{O}(\Delta \cdot mn^{1-1/d})$ time. By choosing $\Delta = n^{1/2d}$ (to balance with the $\widetilde{O}(mn/\Delta)$ running time of BFS computation), this leads to a relatively simple diameter algorithm running in $\widetilde{O}(mn^{1-1/(2d)})$ time, which is truly subquadratic for the arbitrary-diameter regime.

**Component 3: Geometric data structures.** For geometric intersection graphs that are not sparse, we cannot afford to access the neighbors directly even with the modified neighborhoods, so more ideas are needed. The data structure subproblem for the ball growing step we need to solve is the following: assume the interval representations of $\hat{N}^{r-1}[v]$ for every vertex $v$ are precomputed and stored; given a query object $s$, compute the union of intervals in $\hat{N}^{r-1}[v]$ for all objects $v$ that intersect $s$. We can reduce this problem to the following:

**Problem 1.3 (Interval Cover).** *Given a set of $N$ objects $\mathcal{O}$ and each object $o \in \mathcal{O}$ is associated with an interval $I_o \subseteq [1 : n]$. Design a data structure to answer the following query:*

- COVERS?$(q, I)$*: Given a query object $q$ and a query interval $I \subseteq [1 : n]$, decide whether the union of intervals associated with the objects intersecting*[7] *$q$ in $\mathcal{O}$ covers the whole $I$.*

Problem 1.3 can be viewed as a generalization of *range searching* [AE99]: given a query object $q$, find the objects intersecting $q$. In the computational geometry literature, *colored* variants of range searching have been studied [GJS95, KRSV08, GJRS18, CHN20]. The above problem is an even more challenging variant, where each object is equipped with not a color but an interval. This interesting generalization has not been considered before, to the best of our knowledge. (There have been some prior works on *time-windowed* geometric data structures [BDG+14, BCE15, CHP19], but typically queries are associated with a time interval but not the input objects; even more crucially, the queries in those works are mainly about whether a property is true for *some* time value in a query interval $I$, rather than for *all* time values in $I$.) One reason the problem is more challenging than standard range searching is that it is not *decomposable* (if the input set is divided into two subsets, knowing the answers of a query for the subsets does not necessarily help with the overall answer).

We note that en route to their unit square result, Duraj, Konieczny, and Potępa [DKP24] also formulated a non-standard geometric "neighbor-set data structure" problem, but their formulation appears more complicated, as they (and Ducoffe, Habib, and Viennot [DHV22] earlier) worked with symmetric differences of neighborhood sets. Our approach using interval representations is in some sense "dual" to these previous approaches, and is more natural, leading to a geometric data structure problem that is simpler to state.

[7] Here we mean the objects intersect, not their associated intervals.

For unit squares, we give a solution to Problem 1.3 with $N^{1+o(1)}$ preprocessing time and $N^{o(1)}$ query time. Our data structure is deterministic, in contrast to Duraj et al.'s, which uses hashing techniques and inherently requires Monte Carlo randomization. For arbitrary squares, Duraj et al.'s data structure approach does not work at all. Although we are not able to obtain $N^{o(1)}$ query time for Problem 1.3 either, we propose a simple method which divides the range $[1 : n]$ into blocks of size $b$, and builds a data structure for *rainbow colored intersection searching* (a version of colored range searching) for each block. (See Appendix C.1 for details.) This yields $\widetilde{O}(N \cdot b)$ preprocessing and $\widetilde{O}(L/b)$ query time, where $L$ is the length of the query interval. This trade-off turns out to be sufficient to obtain a truly subquadratic algorithm in the end, for an appropriate choice of the parameter $b$.

For unit disks, Problem 1.3 is related to the well-known *Hopcroft problem*[8], and hence a query time $o(N^{1/3})$ appears unlikely [Eri96]; however, to obtain a truly subquadratic time for diameter, we need $O(N^{\delta})$ query time for tiny $\delta > 0$ (since the total number of input and query intervals is $\Omega(n^{2-1/4})$ or worse in our application). We circumvent this issue entirely by partitioning the given set of unit disks into constantly many *modulo classes* (i.e., we partition the plane into cells of constant side-length (say 1/2), and take modulo classes of the index pairs of the cells). This way, if we take one cell $\square$, the collection of disks from the same modulo class intersecting $\square$ forms a *pseudoline arrangement*. When input disks are restricted to one modulo class, we are able to solve Problem 1.3 with $N^{1+o(1)}$ preprocessing time and $N^{o(1)}$ query time.[9] These data structure results may be of independent interest to computational geometers. They do not follow directly from existing techniques. Instead, we propose a clever recursion, repeatedly and alternately taking *lower envelopes and upper envelopes* of pseudo-segments [AS00, Pet15]. (Experts in geometric data structure may find this part interesting, and are encouraged to read Section C.3 for the details.)

**Additional complications for unit disks.** The fact that we have efficient data structures for unit disks only when restricted to a fixed modulo class creates a number of extra technical challenges:

- Because we can only take union over balls from a fixed modulo class, the intermediate sets are no longer neighborhood balls, i.e., we need to work with a new set system. Fortunately, we can still prove that the (dual) VC-dimension of the new set system is at most 4, but *only when the balls have the same radius* $r$.
- This condition in turn forces us to change the stabbing path—and all of its associated interval representations—every time we increment $r$. Fortunately, we show that the interval representations can be updated efficiently using random sampling techniques (with slightly worse amortized stabbing number $\widetilde{O}(n/\rho + \rho^d)$).
- At intermediate steps, we may now need to work with balls from two or three set systems across different types. Fortunately, the combined set systems still have bounded VC-dimension (at most 8).
- The extra overhead in switching stabbing paths is too costly since each stabbing path computation costs $O(n\rho)$ time, and we have to compute for $\widetilde{O}(n/\Delta)$ pieces and $O(\Delta)$ rounds. To achieve overall subquadratic time, we only work with pieces larger than a certain threshold $A$; for small pieces, we need to switch to a different method (based on distance compression), which achieves running time $\widetilde{O}(n \cdot |\partial P| + |P| \cdot (|P| + (|\partial P|\Delta)^d))$ for each piece $P$ of size at most $A$ with boundary $\partial P$.

All these details are explained in Section 7, but to illustrate the intricacies of the overall algorithm to the curious readers, the time bound for diameter for unit-disk graphs has the following form, where the

[8] The Hopcroft problem tests, for a given system of points and lines in the Euclidean plane, whether any point lies on any line. The total number of points and lines is assumed to be $n$.

[9] We do not break the Hopcroft problem's lower bound as we only solve the data structure problem for one modulo class.

| graph class | best previous | | new |
|---|---|---|---|
| planar | $O^*(n^{3/2})$, $O^*(n)$ | [CGL+23] | |
| $K_h$-minor-free | $\widetilde{O}(n^{2-1/(3h-1)})$ | [LW24] | |
| VC-dim.-bounded | $O(mn)$, $O(n^2)$ | folklore | $\widetilde{O}(mn^{1-1/(4d+1)})$ |
| unit-square | $\widetilde{O}(n^2)$ | [CS19] | $O^*(n^{2-1/16})$ |
| square | $\widetilde{O}(n^2)$ | [CS19] | $\widetilde{O}(n^{2-1/20})$ |
| unit-disk | $O(n^2\sqrt{\frac{\log\log n}{\log n}})$ | [CS16] | $O^*(n^{2-1/20})$ |

**Table 2.** Construction time and space bounds of exact distance oracles for different classes of unweighted undirected graphs, with $\widetilde{O}(1)$ query time. We write out both construction time and space bounds only when they are different.

sums are over all pieces $P$ of the LDD (which satisfies $\sum_P |P| = O(n)$ and $\sum_P |\partial P| = \widetilde{O}(n/\Delta)$):

$$O^*\left(\Delta \cdot n\rho + \sum_{P:|P|>A}\left(|\partial P|\cdot n + \Delta\cdot(n+|P|\cdot(n/\rho+\rho^8))\right) + \sum_{P:|P|\le A}\left(n\cdot|\partial P| + |P|\cdot(|P|+(|\partial P|\Delta)^4)\right)\right).$$

Balancing cost by setting parameters $\Delta = n^{1/18}$ and $\rho = A = \Delta^2$ then yields $O^*(n^{2-1/18})$. (Other variants of the algorithm for different graph classes and other related problems will have different expressions and different settings of parameters.)

## 1.3 Other Distance-related Problems

Our framework for computing diameter naturally opens up the possibility of solving other distance-related problems. Here, we focus on three well-studied problems: all-vertex eccentricities, exact distance oracles, and Wiener index.

**Eccentricities.** To highlight the new challenges beyond diameter computation, let us begin with eccentricities. The *eccentricity* of a vertex $v$, denoted by $\mathrm{ecc}(v)$, is the maximum distance from $v$ to any other vertex in $G$. Our goal is to compute $\mathrm{ecc}(v)$ for every $v \in V_G$ in truly subquadratic time. Observe that the diameter is the maximum eccentricity and hence, computing all eccentricities is often more difficult.

For computing diameter, we kick-start the ball growing process with radius $\tau \in [D-\Delta : D]$ and therefore we only need to grow in $O(\Delta)$ interactions. The key challenge in computing eccentricities is that $\mathrm{ecc}(v)$ of some vertex $v$ could be as small as $D/2$, and hence any ball growing process has to cover radii in the entire range $[D/2 : D]$, which can be as large is $\Omega(n)$. Interestingly, our framework for the diameter problem also points us to a way to resolve this issue. Specifically, we grow the modified neighborhood ball $\hat{N}^r[v]$ only for vertices in the same piece $P$ of the low-diameter decomposition. The observation is that for any two vertices $u$ and $v$ in $P$, $|\mathrm{ecc}(u) - \mathrm{ecc}(v)| = O(\Delta)$. Hence, to restrict to computing eccentricities of vertices in $P$, it suffices to grow modified neighborhood balls in $O(\Delta)$ steps. (For different pieces, the ranges of radii could be vastly different.) As the range of radii is piece-specific, the stabbing path data structure also has to be piece-specific instead of being "global" as in the case of computing graph diameter. Our results are summarized in the following theorem.

**Theorem 1.4.** *Let $G$ be a graph on $n$ vertices. We can compute all-vertex eccentricities of $G$ by Las Vegas randomized algorithms in:*

- $O^*(n^{2-1/20})$ *time if $G$ is the intersection graph of unit disks, and*

- $\widetilde{O}(n^{2-1/12})$ *time if $G$ is the intersection graph of axis-aligned squares. For unit-square graphs, the running time is $O^*(n^{2-1/8})$.*
- $\widetilde{O}(mn^{1-1/(2d)})$ *time if $G$ has $m$ edges and (generalized distance) VC-dimension $d$.*

**Exact distance oracle.** An *exact distance oracle* is a data structure that, when given a pair of vertices, returns the shortest path distance of the vertices quickly. Our goal is to construct an oracle with a truly subquadratic *space*. For geometric intersection graphs, known oracles with a truly subquadratic space can answer a distance query approximately within an additive error of 1 [AdT24, CGL24]; for sparse graphs, the query time is close to linear ($\Omega(n^{1-\varepsilon_d})$) for some small constant $\varepsilon_d = 1/2^{O(d)}$ depending on the VC-dimension $d$ [DHV22].

For square and unit-disk graphs and sparse graphs with bounded VC-dimension, we provide an exact oracle with a truly subquadratic space and polylogarithmic query time. Furthermore, our oracle can be constructed in truly subquadratic time; therefore, our result can be interpreted as solving the all-pairs shortest-path problem in truly subquadratic time. (Of course any such algorithm has to output an implicit representation of the shortest distances since the explicit output size is $\Omega(n^2)$.)

Constructing an exact distance oracle is more difficult than computing all-vertex eccentricities: the queried distance range is $[0 : n]$. In computing diameter and eccentricities, the modified ball $\hat{N}^r[s]$ is a subset of the true neighborhood ball $N^r[s]$ and we compute $\hat{N}^r[s]$ for all $r \in [\mathrm{ecc}(s) - O(\Delta) : \mathrm{ecc}(s) + \Delta]$. However, if we query distance between $s$ and $t$ where $d_G(s,t) \ll \mathrm{ecc}(s) - O(\Delta)$, then knowing the true neighborhood ball $N^r[s]$ for $r \geq \mathrm{ecc}(s) - O(\Delta)$ (let alone its subset) does not tell us anything about $d_G(s,t)$. Our idea is to assign weights to vertices of $G$ appropriately and incorporate vertex weight in the definition of $\hat{N}^r[s]$, so that every vertex $t$ belongs to $\hat{N}^r[s]$ for some value of $r \in [-\Delta : \Delta]$; the radius could be negative, which is somewhat counterintuitive. As the range of the (weighted) radii is now $\Theta(\Delta)$, the ideas we develop for computing the diameter and eccentricities now can be applied here. As distances with vertex weights are closely connected to the notion of *generalized* VC-dimension (formally defined in Section 2), we assume the input graph has a bounded generalized VC-dimension in the case of sparse graphs. All other graphs, such as geometric intersection graphs and minor-free graphs, have their generalized VC-dimension equal to the regular VC-dimension (of the neighborhood ball system).

All of these ideas lead to our exact distance oracles for various types of graphs. See Table 2 for a comparison of existing results and ours.

**Theorem 1.5.** *Let $G$ be a graph on $n$ vertices. We can compute an exact distance oracles for $G$ (by randomized Las Vegas algorithms) with the following guarantees:*

- $O^*(n^{2-1/20})$ *construction time and size and $\widetilde{O}(1)$ query time if $G$ is a unit-disk graph.*
- $\widetilde{O}(n^{2-1/20})$ *construction time and size and $\widetilde{O}(1)$ query time if $G$ is a square graph. For unit-square graphs, the construction time and size can be improved to $\widetilde{O}(n^{2-1/16})$.*
- $\widetilde{O}(mn^{1-1/(4d+1)})$ *construction time, $\widetilde{O}(n^{2-1/(4d+1)})$ size, and $\widetilde{O}(1)$ query time if $G$ has $m$ edges and (generalized distance) VC-dimension $d$.*

Interestingly, if we ignore construction time, the above theorem implies the existence of subquadratic-size, $\widetilde{O}(1)$-time distance oracles for all (not necessarily sparse) graphs with bounded (generalized distance) VC-dimension, in particular, all pseudo-disk graphs.

**Wiener index.** The Wiener index of a graph $G$ is the sum of the distances between all pairs of vertices. Computing the Wiener index has been studied [CK97, CK09, WN09]; truly subquadratic algorithms are only known for planar and minor-free graphs [Cab18, GKM+21, LW24]. Here we provide the first such

algorithms for graphs with bounded generalized VC-dimension and several geometric intersection graphs. Indeed, the algorithms for Wiener index are simple corollaries of our algorithms for exact distance oracles in Theorem 1.5 and therefore have the same running time guarantees.

**Theorem 1.6.** *Let $G$ be a graph on $n$ vertices. We can compute the Weiner index of $G$ (by randomized Las Vegas algorithms) in:*

- $O^*(n^{2-1/20})$ *time if $G$ is the intersection graph of unit disks.*
- $\widetilde{O}(n^{2-1/20})$ *time if $G$ is the intersection graph of axis-aligned squares. For unit-square graphs, the running time is $\widetilde{O}(n^{2-1/16})$.*
- $\widetilde{O}(mn^{1-1/(4d+1)})$ *time if $G$ has $m$ edges and (generalized distance) VC-dimension $d$.*

# 2 Preliminaries

## 2.1 Graphs and Low-diameter Decomposition

**Graph notation.** Let $G = (V_G, E_G)$ be an unweighted undirected graph with $n$ vertices and $m$ edges. For two vertices $u, v \in V$, let $d_G(u, v)$ denote the distance between $u$ and $v$ in $G$. Often we will omit the subscript and simply write $d(u, v)$ when the graph $G$ is clear. The *neighborhood* of a vertex $v \in V_G$ is the set of vertices that are distance at most 1 to $v$, denoted by $N[v] := \{u \in V_G : d(u, v) \leq 1\}$. The *$k$-neighborhood ball* of a vertex $v \in V_G$ is the set of vertices with distance at most $k$ from $v$, denoted by $N^k[v] := \{u \in V : d(u, v) \leq k\}$. (Notice that $N[v] = N^1[v]$ and $N^k[v] = N[N^{k-1}[v]]$.) Define the set of $k$-neighborhood balls as $\mathcal{N}_G^k := \{N^k[v] : v \in V\}$, and the set of all neighborhoods balls as $\mathcal{B}_G := \bigcup_k \mathcal{N}_G^k$.

**Geometric intersection graphs.** Consider a set $S$ of $n$ geometric objects in the plane. We define the *geometric intersection graph* $G$ of $S$ as the graph obtained by creating a vertex for every geometric object, and connecting two geometric objects if they intersect. When $S$ consists of unit disks, i.e., disks of radius 1, we refer to the geometric intersection graph $G$ as a *unit-disk graph*. If $S$ consists of axis-aligned unit squares, we refer to the geometric intersection graph $G$ as a *unit-square graph*. We will also consider when $S$ consists of axis-aligned squares (of arbitrary size). We refer to such graphs as *square graphs*. In Appendix A.2, we describe a near-linear time algorithm for computing a BFS tree for square graphs, as stated below; the algorithm for unit-disk graphs is known [Klo23].

**Lemma 2.1.** *Let $G$ be the geometric intersection graph of squares or unit disks with $n$ vertices. We can compute a BFS tree from any given vertex of $G$ in $\widetilde{O}(n)$ time.*

**Low-diameter decomposition.** Let $G$ be a graph with $n$ vertices and $\Delta > 0$ be a diameter parameter. A *low-diameter decomposition* (LDD) of $G$ with parameter $\Delta$ is a decomposition of the vertex set $V$ into disjoint sets $V = V_1 \cup \ldots \cup V_k$ and corresponding induced subgraphs $P_i := G[V_i]$ called *pieces*, such that:

- **Low diameter:** Piece $P_i$ is a single connected component of (strong) diameter[10] at most $\Delta$.
- **Small boundary:** Denote the boundary vertices of $P_i$ as $\partial P_i$, that is, the subset of vertices of $P_i$ that has an edge to a vertex in $V_G \setminus V_i$. The decomposition satisfies $\sum_{i=1}^{k} |\partial P_i| = \widetilde{O}(n/\Delta)$.
- **No small pieces:** Each piece has size at least $\widetilde{\Omega}(\Delta)$.

We show in Appendix A that such a decomposition always exists. Furthermore, in Appendix A.1, we show an efficient algorithm for computing this decomposition.

[10] By strong diameter we mean that the shortest path between any two vertices in $P_i$ *within* the subgraph $P_i$ is at most $\Delta$.

**Theorem 2.2.** *Let $G$ be a graph with $n$ vertices and $m$ edges. For any parameter $24\log n < \Delta \le n$, we can compute a low-diameter decomposition for $G$ in $O(m+n)$ time.*

For unit-disk graphs and square graphs, we prove in Appendix A.2 that the low-diameter decomposition is efficiently computable in near-linear time.

**Theorem 2.3.** *Let $G$ be an intersection graph of $n$ unit disks or an intersection graph of $n$ axis-aligned squares. For any parameter $24\log n < \Delta \le n$, we can compute a low-diameter decomposition for $G$ in $\widetilde{O}(n)$ time.*

## 2.2 VC-dimension

A *set system* is a pair $(X, \mathcal{S})$, consisting of a ground set $X$ and a collection of ranges that are subsets of $X$; in notation, $\mathcal{S} \subseteq 2^X$. A subset $Y \subseteq X$ is said to be *shattered* by $\mathcal{S}$ if the collection $\{Y \cap S : S \in \mathcal{S}\} = 2^Y$, that is, all possible subsets of $Y$ can be obtained by $\mathcal{S}$. The *shatter function*, denoted by $\pi_{(X,\mathcal{S})}(k)$ is the largest number of sets that is created by the set system when restricted to $Y \subseteq X$ of size $k$. Formally it is:

$$\pi_{(X,\mathcal{S})}(k) = \max_{\substack{Y \subseteq X \\ |Y|=k}} |\{Y \cap S : S \in \mathcal{S}\}|.$$

The *shatter dimension* of a set system is the smallest value $d$ such that $\pi_{(X,\mathcal{S})}(k) = O(k^d)$ for all $k$. The *VC-dimension* of a set system $(X, \mathcal{S})$ is the size of the largest subset of $Y \subseteq X$ that can be shattered by $\mathcal{S}$. The *dual set system* of $(X, \mathcal{S})$ is the set system $(\mathcal{S}^*, X^*)$, where the ground set $\mathcal{S}^* = \{w_S : S \in \mathcal{S}\}$ consists of elements indexed by $\mathcal{S}$, and each $s \in S$ induces a range $s^* = \{w_S \in \mathcal{S}^* : S \ni s\}$ in $\mathcal{S}^*$. The *dual VC-dimension* of a $(X, \mathcal{S})$ is the VC-dimension of the dual set system, and analogously the *dual shatter dimension* is the shatter dimension of the dual set system. We state some well-known results [Har11].

**Lemma 2.4.** *Let $(X, \mathcal{S})$ be a set system of VC-dimension $d$. The following is true:*

1. *The dual set system $(\mathcal{S}^*, X^*)$ has VC-dimension at most $2^{d+1} - 1$.*
2. *For $Y \subseteq X$, the set system $(Y, \mathcal{S})$ has VC-dimension at most $d$.*
3. *(Sauer-Shelah Lemma [She72, Sau72].) If $|X| \le n$ then $|\mathcal{S}| \le O(n^d)$, so the shatter dimension of $(X, \mathcal{S})$ is at most $d$.*

**VC-dimension in graphs.** The *$k$-distance VC-dimension* of a graph $G = (V_G, E_G)$ is the VC-dimension of the set system of $k$-neighborhood balls $(V_G, \mathcal{N}_G^k)$. (Sometimes in the literature, e.g., [DHV22], the VC-dimension of $G$ is defined to be the 1-distance VC-dimension.) The *distance VC-dimension* of $G$ is the VC-dimension of the set system of balls $(V_G, \mathcal{B}_G)$. Observe that the $k$-neighborhood set system $(V_G, \mathcal{N}_G^k)$ is equivalent to its dual, so the dual VC-dimension is the same as the primal. This is not the case for the set system of arbitrary balls since the ground set and the set of ranges have different sizes.

Karczmarz and Zheng [KZ25] introduced[11] a natural generalization: a set system $(U, \mathcal{GB}_G)$ whose ground set is $U = V_G \times \mathbb{Z} = \{(u, r) : u \in V_G, r \in \mathbb{Z}\}$, and the ranges $\mathcal{GB}_G$ consists of *generalized neighborhood balls* for $v \in V_G$ and $k \in \mathbb{Z}$ of the form:

$$\tilde{N}^k[v] := \left\{(u, r) \in V_G \times \mathbb{Z} : d(u, v) \le r + k\right\}.$$

Note that values of $r$ and $k$ are allowed to be negative. We call the VC-dimension of $(U, \mathcal{GB}_G)$ the *generalized distance VC-dimension* of a graph. It can be observed that this set system is equivalent to its dual. Furthermore, we can observe the following relationship between these VC-dimensions.

[11] [KZ25] consider what they call a *multiball* set system where the ground set is $V_G \times M$ for a set of real weights $M \subseteq \mathbb{R}$.

**Observation 2.5.** *$k$-distance VC-dimension of $G \leq$ distance VC-dimension of $G \leq$ generalized distance VC-dimension of $G$.*

Throughout this paper, when we refer to graphs of bounded VC-dimension, we will be referring to families of graphs whose generalized distance VC-dimension of the graph is bounded by an absolute constant. Many of our results can also be adapted with more work to graphs that have bounded $k$-distance VC-dimension for all $k$, or graphs with bounded distance VC-dimension. We will focus on generalized distance VC-dimension as it holds for minor-free graphs and the geometric intersection graphs we care about, and also leads to the simplest exposition of our ideas.

**Connection to distance encoding VC-dimension.** Distance encodings were used by Li and Parter [LP19] to compute the diameter in a planar graph. This was later modified to a more general setting by Le and Wulff-Nilsen [LW24], whose definition we present below (restricted to unweighted graphs).

**Definition 2.6.** Let $G = (V_G, E_G)$ be an undirected unweighted graph. Let $M \subseteq \mathbb{Z}$ be a set of integers. Let $S \subseteq V_G$ be an ordered set of $\ell$ vertices $S = \{s_0, s_1, \ldots, s_{\ell-1}\}$. For every vertex $v \in V_G$ define the set:

$$X_{S,M}(v) \coloneqq \left\{(s_i, \delta) : s_i \in S, \delta \in M, d(v, s_i) - d(v, s_0) \leq \delta\right\}.$$

Let $X_{S,M} \coloneqq \left\{X_{S,M}(v) : v \in V\right\}$ be the set of subsets of the ground set $S \times M$. The *distance encoding VC-dimension* of $G$ is the maximum VC-dimension of set systems of the form $(S \times M, X_{S,M})$ for all possible $S$ and $M$.

Observe that the set $X_{S,M}(v)$ is isomorphic to $\tilde{N}^{d(v,s_0)}[v] \cap (S \times M)$. Restricting the ground set of the set system $(U, \mathcal{GB}_G)$ to $(S \times M, \mathcal{GB}_G)$ does not increase the VC-dimension by Lemma 2.4, so we conclude the following observation.

**Observation 2.7.** *Distance encoding VC-dimension of $G \leq$ generalized distance VC-dimension of $G$.*

**Graphs of bounded generalized distance VC-dimension.** It was shown by Chepoi, Estellon, and Vaxes [CEV07] that planar graphs have distance VC-dimension at most 4 by explicitly constructing a $K_5$ minor (by contradiction). This argument was extended by Bousquet and Thomassé [BT15] to show that $K_h$-minor-free graphs have distance VC-dimension at most $h-1$. Le and Wulff-Nilsen [LW24] used a variation of this argument to show that $K_h$-minor-free graphs have distance encoding VC-dimension at most $h-1$. This argument was adapted by Karczmarz and Zheng [KZ25] to show that $K_h$-minor-free graphs have generalized distance VC-dimension at most $h-1$ as well.

**Theorem 2.8 ([KZ25]).** *Any $K_h$-minor-free graph has generalized distance VC-dimension at most $h-1$.*

For unit-disk graphs, it was shown by Abu-Affash *et al.* [ACM+21] that the distance VC-dimension is 4. Later, by Chang, Gao, and Le [CGL24], the intersection graph of pseudo-disks has distance VC-dimension and distance encoding VC-dimension of 4 as well. The bound on distance VC-dimension was also independently shown by Duraj, Konieczny, and Potępa [DKP24] for intersection graphs of fixed translates of geometric objects in the plane. The proof in [CGL24] can be easily adapted to also bound the generalized distance VC-dimension.

**Theorem 2.9.** *Any geometric intersection graph of pseudo-disks in the plane has generalized distance VC-dimension at most* 4.

## 2.3 Stabbing Path and Interval Representation

Let $(X, \mathcal{S})$ be a set system with $|X| \le n$ and $|\mathcal{S}| \le m$. Let $\lambda$ be an ordering of the elements of $X$. Given a set $S \in \mathcal{S}$, define the *$\lambda$-interval representation* $\mathsf{Rep}_\lambda(S)$ ($\lambda$-representation for short) as the collection of maximal contiguous subsequences of $\lambda$—called *intervals*—whose union is $S$. The size of the representation $|\mathsf{Rep}_\lambda(S)|$ refers to the number of such intervals. For a parameter $1 \le \rho \le m$, a *$\rho$-stabbing path $\lambda$* of a set system $(X, \mathcal{S})$ of dual VC-dimension $d$ is an ordering of $X$ such that $\sum_{S\in\mathcal{S}} |\mathsf{Rep}_\lambda(S)| = \widetilde{O}(mn/\rho + m\rho^{d-1})$. Observe that if $n^{1/d} \le m$, this quantity is minimized when $\rho = n^{1/d}$ so $\sum_{S\in\mathcal{S}} |\mathsf{Rep}_\lambda(S)| = \widetilde{O}(mn^{1-1/d})$. We will sometimes refer to an $n^{1/d}$-stabbing path $\lambda$ simply as a *stabbing path*. We assume the existence of an *element reporting oracle* that, given $S \in \mathcal{S}$, can enumerate all elements of $S$ in $T_0(n)$ time, where $T_0(n) \ge n$.

In Appendix D, we show the following lemma to construct $\rho$-stabbing paths with high probability[12].

**Lemma 2.10.** *Let $(X, \mathcal{S})$ be a set system with $|X| \le n$ and $|\mathcal{S}| \le m$ with dual shatter dimension at most $d$. For any parameter $1 \le \rho \le m$, we can construct a stabbing path of $(X, \mathcal{S})$, that is, an ordering $\lambda$ of $X$ such that $\sum_{S\in\mathcal{S}} |\mathsf{Rep}_\lambda(S)| = \widetilde{O}(mn/\rho + m\rho^{d-1})$ in $\widetilde{O}(T_0(n)\cdot\rho)$ time with high probability.*

## 2.4 Geometric Data Structures

We consider three different geometric data structures in decreasing difficulty that we will use in our algorithms, and the reduction from difficult problems to easier problems. We provide the details of the reductions in Appendix C.1.

**Interval searching.** The interval searching problem directly captures the ball growing process for various objects in the frameworks we study in Section 3 and Section 8.

**Problem 2.11 (Interval Searching).** *Let $\mathcal{O}_{IS}$ be a given set of objects, where each object $o \in \mathcal{O}_{IS}$ is associated with a set of intervals (of integer points) of $[1:n]$, denoted by $\mathcal{I}_o$. Design a data structure that answers the following query:*

- IntervalSearch($q$)*: Given an object $q$, return (the interval representation of) all the integer points in $[1:n]$ associated with objects in $\mathcal{O}_{IS}$ that intersect $q$.*

For each query object $q$, let $\mathcal{I}_{out}(q)$ be the set of intervals representing all the integer points of the *output*. Ideally, we want to construct a data structure for the interval searching problem that has near-linear preprocessing time and poly-logarithmic query time. However, this is difficult even when the objects are unit-disk graphs.

In our context, we will be querying interval searching for each object in $\mathcal{O}_{IS}$, and therefore, we will be solving the offline version of Problem 2.11. Let $N_{IS} := \sum_{o\in\mathcal{O}_{IS}} (|\mathcal{I}(o)| + |\mathcal{I}_{out}(o)|)$ be the total number of input and output intervals. Let $L_{IS} := \sum_{o\in\mathcal{O}_{IS}} \sum_{I\in\mathcal{I}(o)\cup\mathcal{I}_{out}(o)} |I|$ be the total length of the input and output intervals. We want to construct a data structure $\mathcal{D}_{IS}$ for solving Problem 2.11 that has a small total run time as a function of $N_{IS}$ and $L_{IS}$. Here, the *total run time* of the $\mathcal{D}_{IS}$ includes: (1) the preprocessing time and (2) the total time to answer all the queries.[13]

[12] In this paper we say an event $E$ happens *with high probability* if $\Pr[E] \ge 1 - n^{-c}$ for some big enough constant $c$.

[13] Alternatively, the offline version of Problem 2.11 is equivalent to computing a Boolean matrix product $C = AB$, where $A$ is the adjacency matrix of an intersection graph, and $B$ and $C$ are Boolean matrices whose 1 entries can be covered by a small number of row intervals. We will not adopt this viewpoint here.

**Interval cover.** This is the data structure Problem 1.3. Recall that $N$ is the number of input objects. Let $N_{IC}$ be the total number of input objects *and* query objects, and $L_{IC}$ be the total length of the input and query intervals. Similar to the interval searching problem, we want to construct a data structure $\mathcal{D}_{IC}$ for solving Problem 1.3 with small total running time, as a function of $N_{IC}$ and $L_{IC}$. We will show (in Appendix C.1) that if we can solve the interval cover problem efficiently, then we can solve the interval searching efficiently.

**Lemma 2.12.** *If one can construct a data structure $\mathcal{D}_{IC}$ for solving Problem 1.3 with total run time $T(N_{IC}, n, L_{IC})$ (for some polynomial function $T$), then we can construct a data structure $\mathcal{D}_{IS}$ for solving Problem 2.11 in total run time $\widetilde{O}(T(N_{IS}, n, L_{IS}))$. Furthermore, if $\mathcal{D}_{IC}$ has preprocessing time $P(N)$ and query time $Q(N)$, then $\mathcal{D}_{IS}$ has preprocessing time $\widetilde{O}(P(\tilde{N}_{IS}))$ and query time $\widetilde{O}(Q(\tilde{N}_{IS}) \cdot |\mathcal{I}_{out}(q)|)$ where $\tilde{N}_{IS} := \sum_{o \in \mathcal{O}_{IS}} |\mathcal{I}(o)|$ is the total number of input intervals and $\mathcal{I}_{out}(q)$ is the set of output intervals from the interval search query of $q$ to $\mathcal{D}_{IS}$.*

**Rainbow colored intersection search.** On the surface, the next problem we present seems to be a strict special case of Problem 1.3 by requiring the interval to be a singleton. However, we will show that a solution to this problem gives us solutions to the two other problems.

**Problem 2.13 (Rainbow Colored Intersection Searching).** *Given a set of objects $\mathcal{O}_{RC}$, each object $o \in \mathcal{O}_{RC}$ is associated with a color. Design a data structure to answer the following query:*

- RainbowCover?$(q)$*: Given a query object $q$, decide whether all the colors appear in the set of objects intersecting $q$.*

In Appendix C.1, we show how to use a data structure $\mathcal{D}_{RC}$ for solving Problem 2.13 to design a data structure $\mathcal{D}_{IC}$ for solving Problem 1.3.

**Lemma 2.14.** *If we can construct in $\widetilde{O}(|\mathcal{O}_{RC}|)$ time a data structure $\mathcal{D}_{RC}$ with $\tilde{O}(1)$ query time for solving Problem 2.13, then for any parameter $b \in [1, n]$, we can construct a data structure $\mathcal{D}_{IC}$ for solving Problem 1.3 that has total run time $\widetilde{O}(N_{IC} \cdot b + L_{IC}/b)$.*

This together with Lemma 2.12 implies a solution to the interval searching problem, in particular, a data structure $\mathcal{D}_{IS}$ for solving Problem 2.11 that has total run time $\widetilde{O}(N_{IS} \cdot b + L_{IS}/b)$.

## 2.5 Handling of Small Pieces

While the low-diameter decomposition guarantees that all pieces have size at least $\widetilde{\Omega}(\Delta)$, sometimes this guarantee is not enough, and we will switch to a different algorithm.

**Diameter and eccentricities.** For computing diameter and eccentricities, we use the notion of *patterns* [LP19], and present the following lemma implicit in the work of Le and Wulff-Nilsen [LW24].

**Lemma 2.15.** *Let $G$ be a graph on $n$ vertices with distance encoding VC-dimension $d$. Let $P$ be a piece in $G$ with boundary $\partial P$ and diameter $\Delta$. If distances from $\partial P$ to all vertices of $G$ are known, the eccentricity of all vertices in $P$ can be computed in $O\left(n \cdot |\partial P| + (|P| + |\partial P|^d \Delta^d) \cdot T(P)\right)$ where $T(P)$ is the time it takes to run boundary weighted BFS on $P$ with weights at most $\Delta$.*

We add one small optimization to the result of Le and Wulff-Nilsen [LW24] using the notion of boundary weighted BFS, a BFS where boundary vertex distances are initialized. This boundary weighted BFS can be performed in time linear in the number of edges of the piece for sparse graphs, and in time near-linear in the number of vertices of $P$ for geometric intersection graphs. See Appendix E.2 for further details and a complete proof of Lemma 2.15.

**Distance oracles.** Similarly, for distance oracles, we will use the following lemma, also implicit in the work of Le and Wulff-Nilsen [LW24].

**Lemma 2.16 (Section 4.3.1 of [LW24]).** *Let $G = (V_G, E_G)$ be a graph with bounded distance VC-dimension $d$, and $P$ be an induced subgraph of $G$ with boundary $\partial P$ and diameter $\Delta$. There exists a distance oracle that answers distances from any vertex $s \in P$ and any vertex $t \in V_G$ with $O(n \cdot |\partial P| + |P|^d)$ space and $O(\log |\partial P|)$ query time.*

*Furthermore, if $G$ also has bounded generalized distance VC-dimension $d$ and distances from $\partial P$ to all vertices of $G$, the distance oracle can be computed in $O\left(n \cdot |\partial P| + (|\partial P|^d \Delta^d + |P|) \cdot T(P)\right)$ time, where $T(P)$ is the time it takes to run vertex weighted BFS on $P$ with weights at most $\Delta$.*

For completeness, we provide the proof of Lemma 2.16 in Appendix E.4.

# 3 Framework for Diameter and Eccentricities

In this section, we outline the algorithmic framework for computing the all-vertex eccentricity of different graph families in truly subquadratic time. (Recall that the *eccentricity* of a vertex $u$ is defined to be $\mathrm{ecc}(u) := \max_{v \in V_G} d(u, v)$.) Note that as diameter is the maximum eccentricity of any vertex in the graph, we can also compute diameter in truly subquadratic time. Our framework can be tweaked for other problems, such as constructing distance oracles and Weiner index; we defer to Section 8.

Now we formally set up the framework, which consists of a few high-level instructions, with the goal to compute for every vertex $u$, the $r$-neighborhood balls $N^r[u]$ iteratively for growing values of $r$. This is enough to answer the diameter problem exactly because a graph $G$ has diameter at most $D$ if and only if every radius-$D$ neighborhood ball contains all vertices in $G$.

Let $G$ be the input graph, given either explicitly using adjacency lists or implicitly as the intersection graph of objects. Our framework has three steps.

**Step 1: Low-diameter decomposition (LDD).** Compute a low-diameter decomposition $\mathcal{L}$ of $G$ with a diameter parameter $\Delta > 0$. $\mathcal{L}$ has $\widetilde{O}(n/\Delta)$ pieces, each of strong diameter at most $\Delta$. Furthermore, $\sum_{P \in \mathcal{L}} |\partial P| = \widetilde{O}(n/\Delta)$. The vertices in $\bigcup_P \partial P$ are called the *boundary vertices*.

**Step 2: Shortest-path computations.** For each boundary vertex $v$ in $\bigcup_P \partial P$, compute a breadth-first search tree in $G$ rooted at $v$. We obtain $\mathrm{ecc}(v)$ as a byproduct. Define

$$\partial\mathrm{ecc} := \max_{\text{boundary vertex } v} \mathrm{ecc}(v).$$

**Step 3: Growing neighborhood balls.** Consider one piece $P$ in the low-diameter decomposition $\mathcal{L}$. Our goal is to compute *some modified version of* $N^r[s]$ for every vertex $s$ in $P$ and *necessary values of* $r$. Fix an arbitrary vertex $s_P$ in $\partial P$, and define $\mathrm{ecc}_P$ as the corresponding eccentricity $\mathrm{ecc}(s_P)$. (Notice that $\mathrm{ecc}(s_P)$ is known after the shortest-path computation in Step 2 because $s_P$ is a boundary vertex of $P$.) We set the *modified neighborhood ball* for each vertex $s$ in $P$ to be

$$\hat{N}^r[s] := N^r[s] \cap R_P, \text{ with } R_P := \left\{t \in V_G : \mathrm{dist}(s_P, t) \geq \mathrm{ecc}_P - 2\Delta\right\},$$

where $\Delta$ was defined to be the strong diameter bound of the pieces in the LDD and $R_P$ is called the *relevant region* for the eccentricity computation for $P$. We will compute $\hat{N}^r[s]$ for every $s$ in $P \setminus \partial P$ iteratively using the inductive formula

$$\hat{N}^r[s] = \bigcup_{v \in N[s]} \hat{N}^{r-1}[v]. \tag{1}$$

We emphasize that while the notation seems to suggest otherwise, the definition of modified neighborhood balls $\hat{N}^r[s]$ *depends on the piece* $P$. Define the set of *relevant balls* to be

$$\mathcal{S}_P := \left\{\hat{N}^r[v] : v \in P, r \in [\mathrm{ecc}_P - 3\Delta, \mathrm{ecc}_P + \Delta]\right\}.$$

Let $\mathcal{S} := \bigcup_{P\in\mathcal{L}} \mathcal{S}_P$. The *ball growing process* consists the following substeps:

3.1. For every $s \in \partial P$ and every $r \in [\mathrm{ecc}_P - 3\Delta, \mathrm{ecc}_P + \Delta]$, compute modified balls $\hat{N}^r[s]$ using Step 2.

3.2. As a base case, we initialize $\hat{N}^r[s] = \emptyset$ for every $s \in P \setminus \partial P$ when $r = \mathrm{ecc}_P - 3\Delta - 1$.

3.3. For other values of $r \in [\mathrm{ecc}_P - 3\Delta, \mathrm{ecc}_P + \Delta]$, compute $\hat{N}^r[s]$ using the inductive formula (1).

Then $\mathrm{ecc}(s)$ is the smallest value $r$ such that $\hat{N}^r[s]$ is the whole relevant region $R_p$. Therefore, we can compute $\mathrm{ecc}(s)$ from $\left\{\hat{N}^r[s] : r \in [\mathrm{ecc}_P - 3\Delta - 1, \mathrm{ecc}_P + \Delta]\right\}$.

Note that we will assume that the diameter of the graph is at least $4\Delta$, otherwise the entire graph $G$ is a low-diameter decomposition of parameter $4\Delta$, and we can simply apply Step 3 with the relevant neighborhood balls being all balls, the relevant region $R_P$ being $V_G$, and the modified neighborhood balls being normal neighborhood balls.

**Correctness.** To show that our algorithmic framework is correct, we show that we correctly computed all (modified) neighborhood balls, assuming the ball growing process is correct. First note that for $s \in \partial P$, we have correctly computed the modified neighborhood balls in Step 3.1. If $s \in P \setminus \partial P$, given the pair $(s, t)$ realizing $\mathrm{ecc}(s)$, we can guarantee that the vertex $t$ must lie in the relevant region $R_P$: Denote $t_P$ to be the vertex that has distance $\mathrm{ecc}_P$ to $s_P$, then because $\mathrm{dist}(s, s_P) \le \Delta$, we have

$$\mathrm{dist}(s_P, t) \ge \mathrm{dist}(s, t) - \Delta \ge \mathrm{dist}(s, t_P) - \Delta \ge \mathrm{dist}(s_P, t_P) - 2\Delta,$$

and thus $t$ can be found in $\hat{N}^r[s]$ if $\hat{N}^r[s]$ is a relevant ball in $\mathcal{S}_P$. Furthermore, again by triangle inequality, $\mathrm{ecc}(s)$ is at least $\mathrm{ecc}_P - \Delta$ and at most $\mathrm{ecc}_P + \Delta$. Thus it is sufficient to initialize $r$ to be $\mathrm{ecc}_P - 3\Delta - 1$ (in which case $N^r[s] \cap R_P = \emptyset$), so the initialization in Step 3.2 is correct. Thus, assuming the ball expansion step is correct, all modified neighborhood balls are computed correctly in Step 3.3.

**VC-dimension of neighborhood balls and stabbing paths.** We cannot afford to store the (modified) neighborhood balls $\hat{N}^r[v] \in \mathcal{S}_P$ explicitly. Instead, we will rely on a compact representation of a set system with bounded VC-dimension to store the neighborhood balls *implicitly* in a data structure. Given the (modified) neighborhood ball system $(V_G, \mathcal{S}_P)$, we are responsible for bounding the (dual) VC-dimension of $(V_G, \mathcal{S}_P)$ to be a constant $d$. Then we compute stabbing path $\lambda$ for $(V_G, \mathcal{S}_P)$, such that the interval representation $\mathsf{Rep}_\lambda(\hat{N}^r[v])$ of set $\hat{N}^r[v]$ has sublinear size. (See Section 2.3 for definition.)

**Implementing the ball growing process.** To implement the ball growing process, we will use a stabbing path $\lambda$ for the modified neighborhood balls to ensure we can compactly store all such balls. The exact details on how we implement the process will depend on the type of graph we are dealing with.

In a sparse graph $G$, we will show how to implement the ball expansion data structure in $G$ directly by explicitly considering the neighbors $N[v]$ of each vertex $v$ in the graph $G$.

In a geometric intersection graph $G$, we instead implement the ball growing process for a piece $P \in \mathcal{L}$ with a data structure for the interval searching problem defined in Problem 2.11. Each vertex $v$ in $P$ is associated with a geometric object $o_v$. Let $\mathcal{O}_P$ denote the set of these geometric objects. Suppose we have computed compact interval representations $\mathsf{Rep}_\lambda(\hat{N}^{r-1}[v])$ for every vertex $v$ in $P \setminus \partial P$, so we can associate these intervals to $o_v$. Using a data structure $\mathcal{D}_{IC}$ for Problem 2.11, the union of intervals of

objects in $\mathcal{O}_P$ that intersect with $o_v$ is exactly $\hat{N}^r[v]$ by Equation (1). Thus, we can implement the ball growing process in a geometric intersection graph if we have an offline data structure for Problem 2.11. The efficiency of the algorithm will depend on the number of intervals in the representation with respect to the stabbing path $\lambda$.

**Organization.** In the next four sections, we will apply our framework to devise algorithms for diameter and eccentricities for different graph classes: sparse graphs of bounded VC-dimension (Section 4), arbitrary-square graphs (Section 5), unit-square graphs (Section 6), and unit-disk graphs (Section 7). Sections 4, 5–6, and 7 can be read independently, depending on the interest of the reader. The sparse graph case is perhaps the simplest, not requiring geometric data structures. The unit-disk case is the most involved and requires overcoming a number of (interesting) technical challenges.

# 4 Diameter/Eccentricities in Sparse Graphs of Bounded VC-dimension

We begin by applying the framework in Section 3 to sparse graphs of bounded VC-dimension. In this setting, the low-diameter decomposition could be constructed in $O(m)$ time using Theorem 2.2. Computing the BFS tree for every boundary vertex takes $\widetilde{O}(mn/\Delta)$ total time where $\Delta$ is the diameter parameter in the low-diameter decomposition. Thus we focus on the third step of performing ball expansion.

To begin, we construct a global ordering $\lambda$ on all the vertices for our stabbing path data structure. The following is analogous to Corollary 15 in [DKP24] that we tailor to our setting.

**Lemma 4.1.** *We can compute in $\widetilde{O}(mn^{1/d})$ time an ordering $\lambda$ of the vertices in $V$ such that for the system $\mathcal{S} = \bigcup_{P\in\mathcal{L}} \mathcal{S}_P$ such that $\sum_{P\in\mathcal{L}} \sum_{s\in P} \sum_{r=\mathrm{ecc}_P-3\Delta}^{\mathrm{ecc}_P+\Delta} \deg(s) \cdot |\mathsf{Rep}_\lambda(N^r[s])| = \widetilde{O}(\Delta m n^{1-1/d})$.*

**Proof:** Let $\check{\mathcal{S}}$ be the set obtained by taking each set $N^r[s]$ in $\mathcal{S}$ and adding $\deg(s)$ copies of $N^r[s]$ to $\check{\mathcal{S}}$. Observe that:

$$|\check{\mathcal{S}}| = \sum_{P\in\mathcal{L}} \sum_{s\in P} \sum_{r=\mathrm{ecc}_P-3\Delta}^{\mathrm{ecc}_P+\Delta} \deg(s) = O(\Delta \cdot m)$$

We then apply Lemma 2.10 to $X = V(G)$ and $\check{\mathcal{S}}$ with $\rho = n^{1/d}$. Since we can implement the element reporting oracle in $O(m)$ time via BFS, the result follows. □

Now for every relevant piece $P$ in the low-diameter decomposition $\mathcal{L}$, we will restrict our attention to only the relevant region $R_P$ for the eccentricity computation. To do so, we consider the ordering $\lambda_P$ of $R_P$ obtained from $\lambda$ by restricting to the vertices of $R_P$. Observe that doing so does not increase the size of the interval representation of any sets.

**Observation 4.2.** *Let $R$ be a subset of $V$. Let $\lambda$ be an ordering of the vertices $V$, and $\lambda'$ be an ordering of $R$ obtained by restricting $\lambda$ to the vertices in $R$. Then for any set $S \subseteq V$, $|\mathsf{Rep}_{\lambda'}(S \cap R)| \leq |\mathsf{Rep}_\lambda(S)|$.*

**Proof:** For any interval $I \in \mathsf{Rep}_\lambda(S)$, $I \cap R$ is also an interval in $\lambda'$. □

**Ball expansion data structure.** To implement the ball expansion data structure, we will store each neighborhood ball in interval form. For a boundary vertex $s \in \partial P$ , we can compute $\hat{N}^r[s]$ for all $r \in [\mathrm{ecc}_P - 3\Delta, \mathrm{ecc}_P + \Delta]$ in $O(n)$ time using the BFS tree we have computed from step 2, and in addition represent these balls in interval form.

Next we describe how to perform the ball expansion operation. For vertices $s \in P \setminus \partial P$, each neighbor $v \in N[s]$ is also in $P$ and we have a compact interval representation for $\hat{N}^{r-1}[s]$. We can take the union of the set of intervals by doing a line sweep in time $\widetilde{O}\left(\sum_{v \in N[s]} |\mathsf{Rep}_{\lambda_p}(N^{r-1}[v])|\right)$.

Furthermore, it is easy to detect if $\hat{N}^r[s] = R_P$ if the interval representation is all of $\lambda_P$.

**Time analysis.** The amount of time taken for computing the global ordering in Lemma 4.1 is $\widetilde{O}(mn^{1/d})$. The runtime for ball expansion of the boundary vertices is:

$$\sum_{P \in \mathcal{L}} \sum_{s \in \partial P} O(n) \;=\; \widetilde{O}(n^2/\Delta) \tag{2}$$

By Observation 4.2, the ball expansion for a non-boundary vertex $s \in P$ and $s \notin \partial P$ from radius $r-1$ to $r$ takes time

$$\widetilde{O}\left(\sum_{v \in N[s]} |\mathsf{Rep}_{\lambda_p}(\hat{N}^{r-1}[v])|\right) \leq \widetilde{O}\left(\sum_{v \in N[s]} |\mathsf{Rep}_{\lambda}(N^{r-1}[v])|\right).$$

The total time taken for all ball expansion steps for non-boundary vertices across all the pieces is at most:

$$\begin{aligned}\sum_{P \in \mathcal{L}} \sum_{s \in P} \sum_{r=\mathrm{ecc}_P - 3\Delta}^{\mathrm{ecc}_P + \Delta} \widetilde{O}\left(\sum_{v \in N[s]} |\mathsf{Rep}_{\lambda}(N^{r-1}[v])|\right) &= \widetilde{O}\left(\sum_{P \in \mathcal{L}} \sum_{s \in P} \sum_{r=\mathrm{ecc}_P - 3\Delta}^{\mathrm{ecc}_P + \Delta} \deg(s) \cdot |\mathsf{Rep}_{\lambda}(N^{r}[s])|\right) \\ &= \widetilde{O}(\Delta m n^{1-1/d}).\end{aligned}$$

The last equality follows from Lemma 4.1.

Recall that the first two steps of the framework can be implemented in $\widetilde{O}(mn/\Delta)$ time. The total runtime for all three parts is:

$$\widetilde{O}(mn/\Delta + mn^{1/d} + n^2/\Delta + \Delta m n^{1-1/d}) \;=\; \widetilde{O}(mn/\Delta + \Delta m n^{1-1/d}).$$

Setting $\Delta = O(n^{1/(2d)})$ yields a that this algorithm runs in $\widetilde{O}(mn^{1-1/(2d)})$ time.

**Theorem 4.3.** *The diameter problem in a sparse undirected graph $G$ with $n$ vertices and $m$ edges and general distance VC-dimension at most $d$ can be solved in $\widetilde{O}(mn^{1-1/(2d)})$ time.*

**Remark 4.4.** We can also obtain similar results (albeit with possibly worse exponents) for other VC-dimension bounds. If the distance VC-dimension is bounded by $d$ or even if the $k$-neighborhood VC-dimension is bounded by $d$ for all $k$, we can follow an approach similar to what we have for unit-disk graphs (see Section 7). The main difference is an extra step to reorder the vertices when we transition from $k-1$-neighborhoods to $k$-neighborhoods using Appendix D.

# 5 Diameter/Eccentricities in Square Graphs

Next, we apply the framework in Section 3 to intersection graphs of squares. In step 1, we apply Theorem 2.3 to obtain our low-diameter decomposition $\mathcal{L}$ of $G$ in $\widetilde{O}(n)$ time into pieces of size $\log n \leq \Delta \leq n$, where $\Delta$ is a parameter we will choose later. In step 2, we compute BFS trees from each $v \in \bigcup_P \partial P$ using Lemma 2.1. The algorithm takes $\widetilde{O}(n)$ time per vertex, so this step takes $\widetilde{O}(n^2/\Delta)$ time. Note that we also explicitly store all distances from these vertices, which takes $\widetilde{O}(n^2/\Delta)$ space.

Recall that when $s \in \partial P$ then we can explicitly compute $\hat{N}^r[s]$ for all values of $r$ in $O(n)$ time using the distances computed in step 2 of our framework.

**Stabbing path.** We now compute a global stabbing path $\lambda$. We use the following lemma.

**Lemma 5.1.** *Let $G$ be a graph with generalized distance VC-dimension $d$, and a single-source distance finding algorithm with running time $T(n)$. Then the modified neighborhood ball system has a path $\lambda$ such that we have*

$$\sum_{P\in\mathcal{L}}\sum_{s\in P}\sum_{r=\mathrm{ecc}_P-3\Delta}^{\mathrm{ecc}_P+\Delta}|\mathsf{Rep}_\lambda(\hat{N}^r[s])| \;=\; \widetilde{O}(\Delta\cdot n^{2-1/d})$$

*with high probability, i.e., $\lambda$ is a stabbing path of the modified $r$-balls. Furthermore, $\lambda$ can be computed with a randomized algorithm in $\widetilde{O}(n^{1/d}T(n))$ time.*

**Proof:** We apply Lemma 2.10 with $\rho = n^{1/d}$ for the set system

$$\mathcal{S} := \left\{N^r[s] : P\in\mathcal{L}, s\in P\setminus\partial P, r\in[\mathrm{ecc}_P-3\Delta, \mathrm{ecc}_P+\Delta]\right\}.$$

Notice that the system has size at most $|\mathcal{S}| = 3\Delta n$, and we can use the BFS algorithm to report the squares in the modified ball. By Observation 4.2, as $\hat{N}^r[s] = R_P\cap N^r[s]$, we obtain the bound

$$\sum_{P\in\mathcal{L}}\sum_{s\in P}\sum_{r=\mathrm{ecc}_P-3\Delta}^{\mathrm{ecc}_P+\Delta}|\mathsf{Rep}_\lambda(\hat{N}^r[s])| \;\le\; \sum_{P\in\mathcal{L}}\sum_{s\in P}\sum_{r=\mathrm{ecc}_P-3\Delta}^{\mathrm{ecc}_P+\Delta}|\mathsf{Rep}_\lambda(N^r[s])| = \widetilde{O}(\Delta\cdot n^{2-1/d})$$

□

In all of the intersection graphs studied in this paper, we have $T(n) = \widetilde{O}(n)$ and $d = 4$. This leads to a stabbing path $\lambda$ that is computed in $\widetilde{O}(n^{5/4})$ time and has the property that the total size of the representation is $\widetilde{O}(\Delta\cdot n^{7/4})$. Given that there are $O(\Delta\cdot n)$ modified balls we consider, the *amortized* interval count to represent a single modified ball is $O(n^{3/4})$.

**Growing balls.** To grow the modified neighborhood balls, we will design a data structure for solving the interval searching problem (Problem 2.11) for squares, which we restate here: we are given a set of square $S$, where each square $s\in S$ is associated with a set of intervals (of integer points) of $[1:n]$, denoted by $\mathcal{I}_s$. Design a data structure that answers the following query:

- IntervalSearch($q$): Given a square $q$, return (the interval representation of) all the integer points in $[1:n]$ associated with squares in $S$ that intersect $q$.

We will be querying the data structure once for each square $s\in S$. Therefore, we are interested in minimizing the total query time. Let $\mathcal{I}_{out}(s)$ be the set of output intervals for a query square $s$. Let $N := \sum_{s\in S}(|\mathcal{I}(s)| + |\mathcal{I}_{out}(s)|)$ be the total number of input and output intervals. Let $L := \sum_{s\in S}\sum_{I\in\mathcal{I}(s)\cup\mathcal{I}_{out}(s)}|I|$ be the total length of the input and output intervals.

**Lemma 5.2.** *For any parameter $b\in[1:n]$, we can construct a data structure $\check{\mathcal{D}}$ for solving the interval searching problem for squares such that the total time to (i) construct $\check{\mathcal{D}}$ and (ii) answer $|S|$ queries, one for each square $s\in S$, is $\widetilde{O}(N\cdot b + L/b)$.*

**Proof:** By Lemma 2.14, it suffices to construct a data structure $\mathcal{D}_{RC}$ for the rainbow colored intersection searching for squares that has nearly linear preprocessing time and poly-logarithmic query time. We provide such a data structure in Appendix C.2. □

Next, we present a simpler (but slower) algorithm for computing all eccentricities. Then we show how to improve the running time.

**First version.** To compute all eccentricities, for each piece $P$ we restrict $\lambda$ to $R_P$ in $O(n)$ time, and denote the resulting ordering by $\lambda_P$. Next, we set $r = \mathrm{ecc}_P - 3\Delta$ and compute the balls $\{\hat{N}^r[s]\}$ for each $s$. In general, once the representations of $\hat{N}^{r-1}[s]$ are known, the data structure to set up for computing modified balls of radius $r$ will have $\sum_{s\in P} |\mathsf{Rep}_{\lambda_p}(\hat{N}^{r-1}[s])|$ intervals in it.

To compute the representations of $\{\hat{N}^r[s]\}_{s\in P}$, we setup the data structure $\breve{\mathcal{D}}$ that takes as input: (i) a set of squares corresponding to vertices of $P$ and (ii) the interval representation $\{\mathsf{Rep}_{\lambda_p}(\hat{N}^{r-1}[s])\}_{s\in P}$ for radius $r-1$. Then we apply $|P|$ queries $\{\textsc{IntervalSearch}(s) : s \in P\}$ to output the interval representations of $\{\hat{N}^r[s]\}_{s\in P}$. Observe that the total length of all the intervals is at most $2|P|\cdot|R_P| = O(|P|\cdot n)$. Thus, the total running time for each $r$ is:

$$\widetilde{O}\left(b\cdot\sum_{s\in P}(|\mathsf{Rep}_{\lambda_p}(\hat{N}^{r-1}[s])| + |\mathsf{Rep}_{\lambda_p}(\hat{N}^{r}[s])|) + |P|n/b\right)$$

Therefore, the total running time of computing all-vertex eccentricities, including the running time of the first two steps in the framework, is:

$$\begin{aligned}
&\widetilde{O}(n^2/\Delta + n^{5/4}) + \sum_{P\in\mathcal{L}}\sum_{r=\mathrm{ecc}_P-3\Delta}^{\mathrm{ecc}_P+\Delta}\widetilde{O}\left(b\cdot\sum_{s\in P}(|\mathsf{Rep}_{\lambda_p}(\hat{N}^{r-1}[s])| + |\mathsf{Rep}_{\lambda_p}(\hat{N}^{r}[s])|) + |P|n/b\right)\\
=\ &\widetilde{O}(n^2/\Delta + n^{5/4}) + \widetilde{O}(b)\left(\sum_{P\in\mathcal{L}}\sum_{r=\mathrm{ecc}_P-3\Delta}^{\mathrm{ecc}_P+\Delta}|\mathsf{Rep}_{\lambda}(\hat{N}^{r}[s])|\right) + \widetilde{O}(n^2\Delta/b) \qquad (3)\\
=\ &\widetilde{O}(n^2/\Delta + n^{5/4}) + \widetilde{O}(b\Delta\cdot n^{7/4}) + \widetilde{O}(n^2\Delta/b) \quad \text{(by Lemma 5.1 and } d=4)\\
=\ &\widetilde{O}(n^2/\Delta + b\Delta\cdot n^{7/4} + n^2\Delta/b)\\
=\ &\widetilde{O}(n^{2-1/16}). \quad \text{(for optimal choices of } b=\Delta^2 \text{ and } \Delta = n^{1/16})
\end{aligned}$$

**Improved version.** We improve the running time by reducing the $\widetilde{O}(n^2\Delta/b)$ in Equation (3), which is the *total length of the intervals*, to $\widetilde{O}(n^2/b)$ by keeping track of the sets $\hat{N}^r[s]\setminus\hat{N}^{r-1}[s]$ instead of $\hat{N}^r[s]$. Notice that the eccentricity of $s$ will be the largest $r$ where $\hat{N}^r[s]\setminus\hat{N}^{r-1}[s]$ is non-empty. Let $\hat{N}^{=r}[s] := \hat{N}^r[s]\setminus\hat{N}^{r-1}[s]$. Then $|\mathsf{Rep}_{\lambda_p}(\hat{N}^{=r}[s])| \le |\mathsf{Rep}_{\lambda_p}(\hat{N}^{r}[s])| + |\mathsf{Rep}_{\lambda_p}(\hat{N}^{r-1}r[s])|$ and therefore, $\{\hat{N}^{=r}[s]\}_{s\in P, P\in\mathcal{L}}$ has a compact representation:

$$\sum_{P\in\mathcal{L}}\sum_{r=\mathrm{ecc}_P-3\Delta}^{\mathrm{ecc}_P+\Delta}|\mathsf{Rep}_{\lambda}(\hat{N}^{=r}[s])| \le \sum_{P\in\mathcal{L}}\sum_{r=\mathrm{ecc}_P-3\Delta}^{\mathrm{ecc}_P+\Delta}(|\mathsf{Rep}_{\lambda_p}(\hat{N}^{r-1}[s])| + |\mathsf{Rep}_{\lambda_p}(\hat{N}^{r}[s])|) = \widetilde{O}(\Delta\cdot n^{7/4}).$$

Observe that

$$\hat{N}^{=r}[s] = \Big(\bigcup_{v\in N[s]}\hat{N}^{=r-1}[v]\Big)\setminus\Big(\hat{N}^{=r-1}[s]\cup\hat{N}^{=r-2}[s]\Big).$$

Thus, we could apply the same growing ball process for $\hat{N}^r[s]$. More precisely, we compute the interval representation of $\bigcup_{v\in N[s]}\hat{N}^{=r-1}[v]$ by querying the interval searching data structure, and then remove elements from $\hat{N}^{=r-1}[s]\cup\hat{N}^{=r-2}[s]$ using the interval representations of $\hat{N}^{=r-1}[s]$ and $\hat{N}^{=r-2}[s]$ computed from the previous iterations. On the other hand, the intervals in this representation are disjoint, so we can bound the total length $L$ over all $3\Delta$ iterations as $L \le O(|P|n)$ instead of $O(\Delta|P|n)$. Therefore, by applying the same calculation in Equation (3), the final running time is:

$$\widetilde{O}(n^2/\Delta + n^{5/4}) + \widetilde{O}(b)\cdot N + \widetilde{O}(n^2/b) \ =\ \widetilde{O}\left(n^2/\Delta + b\Delta n^{7/4} + n^2/b\right) \ =\ \widetilde{O}(n^{2-1/12})$$

for $b = \Delta = n^{1/12}$.

**Theorem 5.3.** *Computing the diameter and all-vertex eccentricities of square graphs with $n$ vertices can be done in $\widetilde{O}(n^{2-1/12})$ time.*

# 6 Diameter/Eccentricities in Unit-square Graphs

For unit squares, we can obtain a slightly faster algorithm. The first two steps and the stabbing path computation are the same as the algorithm for square graphs. The total running time of these steps is $\widetilde{O}(n^2/\Delta + \Delta n^{5/4})$. The growing ball step is more efficient since we can develop a better geometric data structure for unit squares.

**Growing balls.** For unit squares, we design a more efficient data structure for the interval cover problem (Problem 1.3), and as a result, we obtain a more efficient data structure for the interval searching problem. Let $S$ be a given set of unit squares where each square $q \in S$ is associated with a set of intervals of $[1:n]$. Each query IntervalSearch$(q)$ returns the intervals of integer points in $[1:n]$ associated with unit squares intersecting $q$. Let $\tilde{N} := \sum_{s\in S} |\mathcal{I}(s)|$ be the number if input intervals.

**Lemma 6.1.** *We can construct in $\tilde{N}^{1+o(1)}$ time a data structure $\check{\mathcal{D}}$ for solving the interval searching problem for unit squares that can answer each query* IntervalSearch$(q)$ *in $\tilde{N}^{o(1)} \cdot |\mathcal{I}_{out}(q)|$ time.*

**Proof:** In Appendix C.2 (and more specifically Theorem C.5), we construct a data structure for the interval cover problem (Problem 1.3) for unit square with $\tilde{N}^{1+o(1)}$ preprocessing time and $\widetilde{O}(1)$ query time. Then by Lemma 2.12, we obtain the preprocessing time and query time as in the lemma. □

**The algorithm.** The algorithm is essentially the same for the square graphs in the previous section: restricting the ordering $\lambda$ to $R_p$ to get $\lambda_P$, and growing balls $\{\hat{N}^r[s]\}_{s\in P}$ for $r = \mathrm{ecc}_P - 3\Delta$ to $\mathrm{ecc}_P + \Delta$ by applying a query IntervalSearch$(s)$ for each unit square $s \in P$ to the interval searching data structure in Lemma 6.1 built for $\hat{N}^{r-1}[s]$. Let $\tilde{N}_{r-1} := \sum_{s\in P}(|\mathsf{Rep}_{\lambda_p}(\hat{N}^{r-1}[s])|$ be the number of input intervals to the data structure; the total output size is $\tilde{N}_n$. Since $\tilde{N}_{r-1}^{o(1)} = n^{o(1)}$, the total running time to grow all the balls per piece is

$$n^{o(1)} \cdot \sum_{r=\mathrm{ecc}_P-3\Delta}^{\mathrm{ecc}_P+\Delta} |\mathsf{Rep}_\lambda(\hat{N}^r[s])|.$$

The total running time to compute all eccentricities is:

$$\begin{aligned} & \widetilde{O}(n^2/\Delta + \Delta n^{5/4}) + n^{o(1)} \sum_{P\in\mathcal{L}} \sum_{r=\mathrm{ecc}_P-3\Delta}^{\mathrm{ecc}_P+\Delta} |\mathsf{Rep}_\lambda(\hat{N}^r[s])| \\ = \; & O^*(n^2/\Delta + \Delta n^{5/4} + \Delta n^{7/4}) \quad \text{(by Lemma 5.1 and } d = 4) \\ = \; & O^*(n^{2-1/8}) \quad \text{(for } \Delta = n^{1/8}.) \end{aligned}$$

**Theorem 6.2.** *Computing the diameter and all-vertex eccentricities in unit square graphs with $n$ vertices can be done in $O^*(n^{2-1/8})$ time.*

# 7 Diameter/Eccentricities in Unit-disk Graphs

We now describe how to adapt the framework in Section 3 to the more complicated setting of computing diameter and eccentricities for unit-disks. The computation of LDD and BFS remains unchanged because unit-disks are fat pseudo-disks; we follow Step 1 and Step 2 of the framework (Section 3). For LDD we use Theorem A.3; for BFS we use Lemma 2.1. However, Step 3 requires drastic changes in order to implement the ball expansion step.

In this section we assume the unit-disk graphs are in *center-disk intersection model*: Unlike a typical geometric intersection graph where we create an edge between two objects if they intersect, here we add an edge between two unit-disks if the center of one disk lies in the other disk. It is straightforward to see that the two models are equivalent by doubling the radii of all unit disks. For the sake of simplicity, we will scale the disks so that the radius is still one unit.

## 7.1 Restriction to Fixed Types

**Partition into modulo classes.** We partition the plane into square *cells*: every unit-square is divided into $2 \times 2$ many cells, each of side length $1/2$. Each cell is indexed by the coordinates of its bottom-left corner modulo 3; notice that the coordinates are multiples of $1/2$ and thus there are 6 modulo classes per coordinate. We collect all cells of the same index $(i, j)$ into a set $\mathit{Cell}_{i,j}$; in other words,

$$\mathit{Cell}_{i,j} := \left\{\text{cell } \square : \text{square } \square \text{ is located at } (x, y) \text{ where } x \equiv i \text{ and } y \equiv j \pmod 3\right\}.$$

We then classify the set of unit-disks $\mathcal{D}$ based on the cell classes where the center of the unit-disk lies:

$$\mathcal{D}_{i,j} := \left\{D \in \mathcal{D} : \text{disk } D \text{ has its center located in some cell in } \mathit{Cell}_{i,j}\right\}.$$

Notice that $\{\mathcal{D}_{i,j}\}_{i,j}$ is a partition of $\mathcal{D}$. We say a disk in $\mathcal{D}_{i,j}$ has *type* $(i, j)$. Denote the number of types to be $\sigma$; there are exactly $\sigma = 36$ types.

The disks intersecting a query disk $D_q$ whose center point $q$ lies inside a cell $\square$ come in two flavors: those disks that completely contain the cell $\square$, and those that partially intersect the cell. We call the cells where the centers of these intersecting disks belong *relevant* to $\square$; among them, we call those cells with disks partially intersecting $\square$ *perimetric*. Observe that there are only constantly many cells relevant to any fixed cell $\square$, because we set the side length of each cell to be $1/2$.

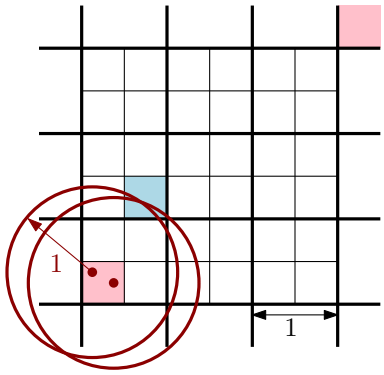


**Figure 2.** The 36 cells formed by partitioning a $3 \times 3$ square. Cells of the same color are of the same type. The disks in $\mathcal{D}_{0,0}$ (in pink) intersects the blue cell as a pseudoline arrangement.

Fixed an arbitrary cell $\square$. A *pseudoline arrangement* $\mathcal{L}$ inside $\square$ is a collection of boundary-to-boundary simple curves in $\square$, such that every pair of curves in $\mathcal{L}$ intersect each other at most once. We now establish the main combinatorial property for disks of the same type: while two unit-disks intersect up to two times in the plane, if we focus on a single cell $\square$ and two disks whose centers are in some other cells of the same type, at most one intersection will appear in $\square$. (Indeed, based on the way we partition cells into modulo classes, at most one cell of each type is relevant to $\square$.)

**Lemma 7.1.** *Let $\mathcal{D}$ be any set of unit-disks, partitioned into types as described above. Given any cell $\square$ and a fixed type $(i, j)$, the boundary of the disks in $\mathcal{D}_{i,j}$ intersects inside $\square$ as a pseudoline arrangement.*

**Proof:** Assume for contradiction that there are two intersecting unit-disks $D$ and $D'$ with two intersection points inside $\square$ simultaneously. As $\square$ has side-length $1/2$ and diameter at most $\sqrt{2}/2 \leq 0.71$, the centers of $D$ and $D'$ must be at least $2 \cdot \left(1^2 - (\sqrt{2}/4)^2\right)^{1/2} = \sqrt{14}/2 \geq 1.87$ units away. Thus the two centers cannot be in the same cell (which has diameter at most 0.71). On the other hand, we reach a contradiction as any two distinct cells of the same type must be at least 2.5 units away (because we index the cells by modulo 3), while the centers of the intersecting disks $D$ and $D'$ can be at most 2 units away. $\square$

If we focus on one perimetric cell $\square'$ of $\square$ and rotate the plane so that $\square'$ lie about vertically above $\square$ (the cells might not be parallel to the axis anymore), we can safely assume each pseudoline formed by the partial intersection by a disk in $\square'$ with $\square$ to be *x-monotone*, that is, any vertical line intersects the pseudoline at most once.

## 7.2 Implementation of the Neighborhood Growing Step

We first describe how to implement the ball growing process (Step 3.3) in the framework using the inductive formula (1), which we recall here:

$$\hat{N}^r[s] = \bigcup_{v \in N[s]} \hat{N}^{r-1}[v]. \tag{1}$$

Fix a piece $P$ from the LDD, and some vertex $s$ in $P \setminus \partial P$. Each modified neighborhood ball $\hat{N}^r[s]$ can be written as the union of a collection of $r$-balls with restrictions on the type of the second vertex in length-$r$ paths. More precisely, recall that a disk $D$ is of type $(i, j)$ if the center of the disk $D$ is located in some cell whose bottom-left corner has coordinates in the modulo class $(i, j)$. We arbitrarily order the constantly many types and label them from 1 to $\sigma$. We say that a path in the intersection graph $G$ from a vertex $s$ to a vertex $v$ is a *$\tau$-path* if the disk corresponding to the vertex following $s$ (the second vertex) in the path is of type $\tau \in [1 : \sigma]$. For each vertex $s$ and an arbitrary subset $M \subseteq [1 : \sigma]$, define

$$\hat{N}^r_M[s] := \left\{ v \in V_P : \text{there is a } \tau\text{-path from } s \text{ to } v \text{ of length at most } r, \text{ for some } \tau \in M \right\}. \tag{4}$$

We often use the $_{\leq T}$ subscript to represent the subset $[1 : T]$ below. Notice that by restricting to $\tau$-paths from $s$ to $v$ in the definition, we can implement the inductive formula using

$$\hat{N}^r_T[s] = \bigcup_{v \in N[s] \cap Cell_T} \hat{N}^{r-1}_{\leq \sigma}[v].$$

We prove the following lemma in Appendix B (Lemma B.2).

**Lemma 7.2.** *For every $r$ and every $T \in [1 : \sigma]$, both set systems*

$$\left(V_G, \left\{\hat{N}^r_T[s] : s \in V_P\right\}\right) \quad \text{and} \quad \left(V_G, \left\{\hat{N}^r_{\leq T}[s] : s \in V_P\right\}\right)$$

*have dual VC-dimension at most* 4.

Notice that $\hat{N}^r[s] = \hat{N}^r_{\leq \sigma}[s]$ and we can compute $\hat{N}^r_{\leq T}[s]$ iteratively by

$$\hat{N}^r_{\leq T}[s] = \hat{N}^r_{\leq T-1}[s] \cup \hat{N}^r_T[s] = \hat{N}^r_{\leq T-1}[s] \cup \bigcup_{v \in N[s] \cap Cell_T} \hat{N}^{r-1}_{\leq \sigma}[v]. \tag{5}$$

Now the strategy should be clear: We will compute $\hat{N}^r_T[s]$ from the previously stored $\hat{N}^{r-1}_{\leq \sigma}[v]$ for every 1-neighbor $v$ of $s$, then take the union with $\hat{N}^r_{\leq T-1}[s]$. The first operation is done with the help of the geometric data structure; to do so, one has to first switch the interval representation for the relevant

neighborhood balls to be with respect to a unifying stabbing path over some *combined* set system that has a bounded VC-dimension. After $\hat{N}^r_T[s]$ is computed, we switch the interval representation back then proceed to compute $\hat{N}^r_T[s] = \hat{N}^r_{\le T-1}[s] \cup \hat{N}^r_T[s]$, this time switching the interval representations to be with respect to another unifying stabbing path over some *auxiliary* set system. Both set systems must have $O(1)$ (dual) VC-dimension. This is the main technical hurdle which we will explain next.

**Geometric data structure.** Fix a cell $\square$ containing $s$ and a perimetric cell $\square'$ of $\square$ which lies vertically above $\square$ (after a rotation). By Lemma 7.1, the collection of disks whose center lies in $\square'$ intersects $\square$ as a collection of $N$ pseudolines $\mathcal{L}$. Assume each pseudoline $\ell_v$ in $\mathcal{L}$ has an associated interval $I_v$ in some given stabbing path $\lambda$. Our next goal is to describe how to build a stabbing path data structure $\mathcal{D}_\lambda$ that answers the COVERS?$(s, I)$ query: whether the union of intervals $I_v$ for every object $v$ intersecting $s$ covers the whole $I$. (This data structure would then be used to solve the interval searching problem, which gives us the interval representation of the modified neighborhood of $s$.) A disk $D_v$ intersecting a query disk $D_s$ with center $s$ in $\square$ corresponds to a pseudoline $\ell_v$ that lies below the center $s$ of the query disk. Therefore it is equivalent if we can support the following query:

- COVERS?$(s, I)$: Given a query point $s$ and an interval $I$, test whether $\bigcup_{\substack{\ell_v \in \mathcal{L} \\ \ell_v \text{ below } s}} I_v$ contains $I$.

**Lemma 7.3.** *Fix a radius $r$. Let $\lambda^\flat$ be a stabbing path defined for the union of set systems*

$$\left\{\hat{N}^{r-1}_{\le\sigma}[v] : v \in V_P\right\} \cup \left\{\hat{N}^r_T[s] : s \in V_P\right\}.$$

*A stabbing path data structure $\mathcal{D}^r_T$ (with respect to $\lambda^\flat$) can be constructed in $n^{1+o(1)}$ time, and support each* COVERS?$(s, I)$ *query in $n^{o(1)}$ time for any $s$.*

A proof of Lemma 7.3 can be found in Appendix C.3. By the same argument in Lemma 6.1, we can augment $\mathcal{D}^r_T$ to construct the interval representation of $\hat{N}^r_T[s]$ with respect to $\lambda^r_T$ for every $s$ by calling an interval search query INTERVALSEARCH$(s)$ to $\mathcal{D}^r_T$. The readers might have noticed that the stabbing order maintained by the data structure $\mathcal{D}^r_T$ is not the same as the stabbing order we would like to represent the neighborhoods $\hat{N}^r_T[s]$ in. This discrepancy leads to the need for tool that allows the switch between different interval representations.

**Switching between interval representations.** We gain the ability to switch between different interval representations by computing a special kind of stabbing paths that "respect" some common $\rho$-sampling of the set system $(X, \mathcal{S})$. This requires us to compute stabbing paths not using Lemma 2.10, but something more sophisticated. Ultimately we will be able to shrink from $\mathcal{S}$ to a subcollection $\mathcal{S}'$ of $\mathcal{S}$, and vice versa. First we set up the terminologies.

We are given a set system $(X, \mathcal{S})$ with at most $n = |X|$ elements and $m = |\mathcal{S}|$ sets with dual shatter dimension of $(X, \mathcal{S})$ is $d$. We fix a *unique $\rho$-sampling* $\mathcal{R}$ of $\mathcal{S}$, where each set in $\mathcal{S}$ chosen with probability $\rho/m$. (Later on we will restrict $\mathcal{R}$ to subcollection $\mathcal{S}'$ of $\mathcal{S}$ and obtain $\mathcal{R}'$; we can still think of $\mathcal{R}'$ as obtained from $\mathcal{S}'$ by sampling each element with probability $\rho/m$, even though we do not explicitly sample from $\mathcal{S}'$. Notice that the parameter $m$ does *not* change even if $\mathcal{S}'$ gets smaller.)

Let $\lambda$ be an ordering of $X$. We say that a set $S$ *crosses* a pair $(x, y)$ if $x \in S$ and $y \notin S$, or vice versa. The number of consecutive pairs in $\lambda$ crossed by $S$ is at most twice the size $|\mathsf{Rep}_\lambda(S)|$. For any collection $\mathcal{R}$, define the *equivalence relation* $\equiv_\mathcal{R}$ over $X$, where $x \equiv_\mathcal{R} y$ if and only if no set in $\mathcal{R}$ crosses $(x, y)$. (In other words, $\{S \in \mathcal{R} : x \in S\} = \{S \in \mathcal{R} : y \in S\}$.) Then $\equiv_\mathcal{R}$ has $O(|\mathcal{R}|^d)$ equivalence classes since the dual shatter dimension is at most $d$.

Let $\mathcal{S}'$ be an arbitrary subcollection of $\mathcal{S}$. Denote the restriction of the unique $\rho$-sampling $\mathcal{R}$ of $\mathcal{S}$ in $\mathcal{S}'$ as $\mathcal{R}'$; in notation, $\mathcal{R}' := \mathcal{S}' \cap \mathcal{R}$. Notice that $\mathcal{R}'$ is also a $\rho$-sampling. Given any set system $(X, \mathcal{S})$, a stabbing path $\lambda$ of $(X, \mathcal{S})$ is *$\mathcal{R}'$-respecting* if each equivalence class of $\equiv_{\mathcal{R}'}$ appears contiguously in $\lambda$ for the restriction $\mathcal{R}'$. (The equivalence classes of $\equiv_{\mathcal{R}'}$ are defined with respect to the restriction $\mathcal{R}'$, not $\mathcal{R}$.)

We compute specialized stabbing paths using the following lemma.

**Lemma 7.4.** *Assume the existence of an element reporting oracle that, given $S \in \mathcal{S}$, can enumerate all elements of $S$ in $T_0(n)$ time. Consider a fixed $\rho$-sampling $\mathcal{R}$ of $\mathcal{S}$. We can compute the equivalence classes of $\equiv_{\mathcal{R}}$ and construct an $\mathcal{R}$-respecting stabbing path $\lambda$ of $(X, \mathcal{S})$ such that $\sum_{S \in \mathcal{S}} |\mathsf{Rep}_\lambda(S)| = \widetilde{O}(mn/\rho + m\rho^{d-1})$ in $\widetilde{O}(T_0(n) \cdot \rho)$ time with high probability. In other words, one can compute a sampled $\rho$-stabbing path $\lambda$ of $(X, \mathcal{S})$ and the equivalence classes of $\equiv_{\mathcal{R}}$ as byproducts.*

For the case of unit-disks, we have the element reporting oracle with query time $T_0(n) = \widetilde{O}(n)$ by computing a BFS tree using Lemma 2.1. Once both stabbing paths (and their corresponding equivalence classes) were computed, respecting a common $\rho$-sampling (and its restriction), we can convert one interval representation to the other efficiently.

**Lemma 7.5.** *[Conversion of interval representations.] Let $(X, \mathcal{S})$ be a set system with $|X| \le n$ and $|\mathcal{S}| \le m$. Let $\mathcal{S}'$ be a subcollection of $\mathcal{S}$ and $\mathcal{T}$ be a subcollection of $\mathcal{S}'$. Let $\mathcal{R}$ be the unique $\rho$-sampling of $\mathcal{S}$, and $\mathcal{R}'$ be its restriction in $\mathcal{S}'$. We are given an $\mathcal{R}$-respecting stabbing path $\lambda$ of $(X, \mathcal{S})$, and an $\mathcal{R}'$-respecting ordering $\lambda'$ of $(X, \mathcal{S}')$ (along with the equivalence classes of $\equiv_{\mathcal{R}}$ and $\equiv_{\mathcal{R}'}$).*

*(1) [Shrinking from $\mathcal{S}$ to $\mathcal{S}'$.] Given $\mathsf{Rep}_\lambda(S)$ for all $S \in \mathcal{T}$, we can compute $\mathsf{Rep}_{\lambda'}(S)$ for all $S \in \mathcal{T}$ in $\widetilde{O}(mn/\rho + m\rho^d)$ total time with high probability.*

*(2) [Expanding from $\mathcal{S}'$ to $\mathcal{S}$.] Given $\mathsf{Rep}_{\lambda'}(S)$ for all $S \in \mathcal{T}$, we can compute $\mathsf{Rep}_\lambda(S)$ for all $S \in \mathcal{T}$ in $\widetilde{O}(mn/\rho + m\rho^d)$ total time with high probability.*

The proof of the two lemmas can be found in Appendix D.

**Neighborhood growing algorithm.** Assuming we are equipped with the geometric data structure (Lemma 7.3) and the ability to switch between interval representations with respect to different stabbing paths (Lemma 7.5), we can now formally describe the algorithm.

Fix a piece $P$. For simplicity of the proof, we use $\mathcal{S}^r$ to denote the collection $\left\{\hat{N}^r[v] : v \in V_P\right\}$ and $\mathcal{S}^r_M$ to denote $\left\{\hat{N}^r_M[v] : v \in V_P\right\}$ for any subset $M \subseteq [1 : \sigma]$. (Recall that the modified balls are defined by intersecting with the relevant region $R_P$, and thus are dependent on $P$.) Similarly, we define $\lambda^r$ be a stabbing path for the $(V_G, \mathcal{S}^r)$, and $\lambda^r_M$ be a stabbing path for $(V_G, \mathcal{S}^r_M)$ for any subset $M \subseteq [1 : \sigma]$. Since all the set systems we need here are with respect to the same ground set $V_G$, we will slightly abuse the notation and use the shorthand $\mathcal{S}$ to denote the set system $(V_G, \mathcal{S})$, and use $\mathcal{S}_1 \cup \mathcal{S}_2$ to denote the union of the two set systems $(V_G, \mathcal{S}_1 \cup \mathcal{S}_2)$.

The algorithm has an *outer-loop* and an *inner-loop*. The *outer-loop* has $4\Delta + 1$ rounds, iterating over every relevant radii $r \in [\mathrm{ecc}_P - 3\Delta : \mathrm{ecc}_P + \Delta]$; at the start of round $r$, we maintain the following *invariants* that we have computed,

(1) an $\mathcal{R}^{r-1}_{\le\sigma}$-respecting $\rho$-stabbing path $\lambda^{r-1}_{\le\sigma}$ for the set system $\mathcal{S}^{r-1}_{\le\sigma}$ and its $\rho$-sampling $\mathcal{R}^{r-1}_{\le\sigma}$; and

(2) $\lambda^{r-1}_{\le\sigma}$-representation of the modified neighborhood balls $\hat{N}^{r-1}_{\le\sigma}[v]$ for every vertex $v$ in $P$.

For each round with radius $r$, our algorithm now performs an *inner-loop* by repeating the following steps for $\sigma$ *iterations*, where $T$ ranges from 1 to $\sigma$; in iteration $T$ we take into account type-$T$ shortest paths using Eq. (5), until we include all $\sigma$ types and thus finish computing the $\lambda^r$-representation of $\mathcal{S}^r$. At the start of iteration $T$, we maintain the following *invariants* that we have computed, for each type $T$,

(i) an $\mathcal{R}^r_{\le T-1}$-respecting $\rho$-stabbing path $\lambda^r_{\le T-1}$ for the set system $\mathcal{S}^r_{\le T-1}$ and its $\rho$-sampling $\mathcal{R}^r_{\le T-1}$;

(ii) $\lambda^r_{\le T-1}$-representation of the modified neighborhood balls $\hat{N}^r_{\le T-1}[s]$ for every vertex $s$ in $P \setminus \partial P$.

At every iteration $T$, we perform the following steps in order. For the base case when $T = 1$, objects with the $_{\le T-1}$ subscript are considered to be null, which we omit from the algorithm.

1. Consider the *combined set system* $\mathcal{S}^{r-1}_{\le \sigma} \cup \mathcal{S}^r_T$. Compute a $\rho$-sampling $\mathcal{R}^r_T$ of $\mathcal{S}^r_T$, and take union with the $\rho$-sampling $\mathcal{R}^{r-1}_{\le \sigma}$ of $\mathcal{S}^{r-1}_{\le \sigma}$ from invariant (1) to form a $2\rho$-sampling $\mathcal{R}^\flat$ of $\mathcal{S}^{r-1}_{\le \sigma} \cup \mathcal{S}^r_T$.
2. Compute an $\mathcal{R}^\flat$-respecting $2\rho$-stabbing path $\lambda^\flat$ along with the equivalence classes of $\equiv_{\lambda^\flat}$ for the combined set system using Lemma 7.4.
3. Convert the $\lambda^{r-1}_{\le \sigma}$-representation of $\hat{N}^{r-1}_{\le \sigma}[v]$ for every $v$ in $P$ given by invariant (2) into $\lambda^\flat$-representation, using Lemma 7.5(2) and the fact that $\mathcal{S}^{r-1}_{\le \sigma}$ is a subcollection of the combined set system.
4. Compute the geometric data structure $\mathcal{D}^r_T$ with respect to the $\rho$-stabbing path $\lambda^\flat$ using Lemma 7.3.
5. Compute the $\lambda^\flat$-representation of
$$\hat{N}^r_T[s] = \bigcup_{v \in N[s] \cap Cell_T} \hat{N}^{r-1}_{\le \sigma}[v]$$
for every vertex $s$ in $P \setminus \partial P$ with the help of geometric data structure $\mathcal{D}^r_T$.
6. Convert the $\lambda^\flat$-representation of $\hat{N}^r_T[s]$ for every $s$ in $P \setminus \partial P$ back into $\lambda^r_T$-representation, using Lemma 7.5(1).

(At this point, we have successfully computed the spanning path $\lambda^r_T$ for $\mathcal{S}^r_T$ and its interval representation with respect to $\lambda^r_T$. We now proceed to take union with $\mathcal{S}^r_{\le T-1}$, currently in $\lambda^r_{\le T-1}$-representation.)

7. Define the *auxiliary set system*:
$$\mathcal{S}^r_{\le T-1} \cup \mathcal{S}^r_T \cup \mathcal{S}^r_{\le T} = \left(V_G, \left\{\hat{N}^r_{\le T-1}[v] : v \in V_P\right\} \cup \left\{\hat{N}^r_T[v] : v \in V_P\right\} \cup \left\{\hat{N}^r_{\le T}[v] : v \in V_P\right\}\right). \quad (6)$$
Compute a $\rho$-sampling $\mathcal{R}^r_{\le T}$ of $\mathcal{S}^r_{\le T}$, and take union with the $\rho$-sampling $\mathcal{R}^r_{\le T-1}$ of $\mathcal{S}^r_{\le T-1}$ from invariant (i) and $\rho$-sampling $\mathcal{R}^r_T$ of $\mathcal{S}^r_T$ computed in Step 1 to form a $3\rho$-sampling $\mathcal{R}^\sharp$ of the auxiliary set system $\mathcal{S}^r_{\le T-1} \cup \mathcal{S}^r_T \cup \mathcal{S}^r_{\le T}$.
8. Compute a $\mathcal{R}^\sharp$-respecting $3\rho$-stabbing path $\lambda^\sharp$ along with the equivalence classes of $\equiv_{\lambda^\sharp}$ for the auxiliary set system, using Lemma 7.4.
9. Convert $\lambda^r_{\le T-1}$-representation of $\hat{N}^r_{\le T-1}[s]$ for every $s$ in $P \setminus \partial P$ given by invariant (ii) to $\lambda^\sharp$-representations, using Lemma 7.5(2).
10. Convert $\lambda^r_T$-representation of $\hat{N}^r_T[s]$ for every $s$ in $P \setminus \partial P$ given by Step 6 to $\lambda^\sharp$-representations, using Lemma 7.5(2).
11. Compute $\hat{N}^r_{\le T}[s]$ by taking the union of $\hat{N}^r_{\le T-1}[s]$ and $\hat{N}^r_T[s]$ as $\lambda^\sharp$-representations for every $s$ in $P \setminus \partial P$. The output $\hat{N}^r_{\le T}[s]$ is again in the auxiliary set system and thus have $\lambda^\sharp$-representation.
12. Compute an $\mathcal{R}^r_{\le T}$-respecting $\rho$-stabbing path $\lambda^r_{\le T}$ for set system $\mathcal{S}^r_{\le T}$, by restricting $\lambda^\flat$ to $\mathcal{S}^r_{\le T}$. Convert the $\lambda^\sharp$-representation of $\hat{N}^r_{\le T}[s]$ for every $s$ in $P \setminus \partial P$ into $\lambda^r_{\le T}$-representation, using Lemma 7.5(1) and the fact that $\mathcal{S}^r_{\le T}$ is a subcollection of the auxiliary set system.

Notice that Step 12 of the algorithm maintains invariants (i) and (ii). After $\sigma$ iterations, the inner-loop ends. We perform one extra step:

13. Convert the $\lambda^\flat$-representation of $\hat{N}^{r-1}_{\le\sigma}[s]$ for every $s$ in $\partial P$ computed in Step 3 into $\lambda^r_{\le\sigma}$-representation, using Lemma 7.5(1). Insert elements in the difference $\hat{N}^r_{\le\sigma}[s]\backslash\hat{N}^{r-1}_{\le\sigma}[s]$ to create $\lambda^r_{\le\sigma}$-representation of $\hat{N}^r_{\le\sigma}[s]$ for every $s$ in $\partial P$.

We then proceed to the next round of the outer-loop. Notice that invariant (1) follows directly from invariant (i), and invariant (2) follows from invariants (ii) together with Step 13 from the previous round.

**Handling small pieces.** The most time-consuming part of our algorithm is to compute the $\rho$-stabbing paths; each computation takes $\widetilde{O}(n\rho)$ time. But we have to compute $O(1)$ many stabbing paths *for each piece and each round of the outer-loop*; there are $\widetilde{O}(n/\Delta)$ many pieces (remember the modified neighborhood balls were defined differently for each piece $P$), but also for $O(\Delta)$ many rounds. Therefore the computation of stabbing paths alone already takes $\widetilde{O}(n\rho\cdot(n/\Delta)\cdot\Delta) = O(n^2\rho)$ time.

To handle the issue, we only apply the above algorithm to *large* pieces whose size is above certain threshold $A \ge \Delta$. This way, the number of such pieces is at most $\widetilde{O}(n/A)$ instead of $\widetilde{O}(n/\Delta)$. (We eventually set $A = \Delta^{O(1)}$.) To compute diameter or eccentricities for small pieces, we use Lemma 2.15.

## 7.3 Analysis for Eccentricities

We make the following observations about the shatter dimension of unions of set systems.

**Observation 7.6.** *Let $X$ be a ground set and let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two set systems on $X$. Let us denote the set of ranges obtained by taking unions of ranges from $\mathcal{S}^1$ and $\mathcal{S}^2$ by $\widehat{\mathcal{S}} := \{S_1 \cup S_2 : S_1 \in \mathcal{S}_1, S_2 \in \mathcal{S}_2\}$. Suppose that the shatter dimension of $\mathcal{S}_1$ is $d_1$ and the shatter dimension of $\mathcal{S}_2$ is $d_2$. Then the shatter dimension of $\mathcal{S}_1 \cup \mathcal{S}_2$ is at most $d_1 + d_2$, and the shatter dimension of $\mathcal{S}_1 \cup \mathcal{S}_2 \cup \widehat{\mathcal{S}}$ is also at most $d_1 + d_2$.*

The observation shows that the combined set system $\mathcal{S}^{r-1}_T \cup \mathcal{S}^r_T$ has dual shatter dimension 8, because both $\mathcal{S}^{r-1}_T$ and $\mathcal{S}^r_T$ individually has dual VC-dimension (and thus dual shatter dimension) at most 4 by Lemma 7.2. It also shows that the dual shatter dimension of the auxiliary set system $\mathcal{S}^r_{\le T-1} \cup \mathcal{S}^r_T \cup \mathcal{S}^r_{\le T}$ is also 8 again as the individual dual VC-dimensions are at most 4 Lemma 7.2. Thus we set $d = 8$ for the time analysis that follows.

**Lemma 7.7.** *Fix a piece $P$, a radius $r$, and some arbitrary parameter $\rho$. We can maintain invariants (1) and (2) between iterations $T-1$ and $T$, in time $O^*(n\cdot\rho + |P|\cdot(n/\rho + \rho^8))$, by computing (1) an $\mathcal{R}^r$-respecting $\rho$-stabbing path $\lambda^r_T$ for the set system $\mathcal{S}^r_T$ for some $\rho$-sampling $\mathcal{R}^r$; and (2) $\lambda^r_T$-representation of the modified neighborhood balls $\hat{N}^r_T[v]$ for every vertex $v$.*

**Proof:** Steps 1 and 7 take $O(n)$ time to compute $\rho$-samplings. Steps 2 and 8 take $\widetilde{O}(n\cdot\rho)$ time to compute $O(\rho)$-stabbing paths. Steps 3, 6 and 13 take $\widetilde{O}(|P|\cdot(n/\rho + \rho^8))$ time to convert interval representations, because the combined set system $\mathcal{S}^{r-1}_{\le\sigma} \cup \mathcal{S}^r_T$ has size $2|P|$. Step 4 takes $O(n^{1+o(1)})$ time to compute the geometric data structure $\mathcal{D}^r_T$. Step 5 takes $O(n\cdot n^{o(1)})$ time to compute the union using $\mathcal{D}^r_T$. Steps 9, 10, 12 take $\widetilde{O}(|P|\cdot(n/\rho + \rho^8))$ time to convert interval representations, because the auxiliary set system $\mathcal{S}^r_{\le T-1} \cup \mathcal{S}^r_T \cup \mathcal{S}^r_{\le T}$ has size $3|P|$. Step 11 takes $O(n)$ time to compute the union of two sets with the same $\lambda^\sharp$-representation. Overall the neighborhood growing step can be implemented in $O^*(n\cdot\rho + |P|\cdot(n/\rho + \rho^8))$ time per piece per radius. □

To analyze the total running time, we separate the pieces of LDD into large and small, based on whether the size of the piece is at least $A$ or not for some parameter $A$. For large piece $P$ of size at least $A$, we run the ball growing algorithm in Section 7.2. The outer-loop is executed for $O(\Delta)$

rounds, each taking $O^*(n\cdot\rho+|P|\cdot(n/\rho+\rho^8))$ time by Lemma 7.7, followed by Step 13, which takes $\widetilde{O}(\sum_{s\in\partial P}\left|\hat{N}^r_{\le\sigma}[s]\setminus\hat{N}^{r-1}_{\le\sigma}[s]\right|)$ time to compute $\hat{N}^r_{\le\sigma}[s]$ for every $s$ in $\partial P$ using binary search in the $\lambda^r_{\le\sigma}$-representation. There are $O(n/A)$ large pieces. Overall, for each large piece, this takes time

$$\begin{aligned}&\widetilde{O}\left(\sum_r\sum_{s\in\partial P}\left|\hat{N}^r_{\le\sigma}[s]\setminus\hat{N}^{r-1}_{\le\sigma}[s]\right|\right)+O(\Delta)\cdot O^*(n\cdot\rho+|P|\cdot(n/\rho+\rho^8))\\&=O^*\left(|\partial P|\cdot n+\Delta\left(n\cdot\rho+|P|\cdot(n/\rho+\rho^8)\right)\right).\end{aligned}$$

By Lemma 2.15, for each small piece $P$ of size less than $A$, it takes $\widetilde{O}(n\cdot|\partial P|+|P|\cdot(|P|+(|\partial P|\Delta)^4))$ time to compute eccentricity of all vertices in $P$. There are at most $\widetilde{O}(n/\Delta)$ small pieces.

The final running time of the all-vertex eccentricities algorithm for unit-disk graphs is:

$$\begin{aligned}&O^*\left(\sum_{P:|P|>A}\left(|\partial P|\cdot n+\Delta\cdot(n\cdot\rho+|P|\cdot(n/\rho+\rho^8))\right)+\sum_{P:|P|\le A}\left(n\cdot|\partial P|+|P|\cdot(|P|+(|\partial P|\Delta)^4)\right)\right)\\&\le O^*\left(n^2/\Delta+\left(\frac{n}{A}\cdot\Delta\cdot n\rho\right)+\sum_{P:|P|>A}\left(|P|\cdot(n/\rho+\rho^8)\right)+\sum_{P:|P|\le A}(n\cdot|\partial P|+|P|^2)+\sum_{P:|P|\le A}\left(|\partial P|\cdot A^4\Delta^4\right)\right)\\&=O^*\left(n^2/\Delta+\Delta n^2\rho/A+\Delta n^2/\rho+\Delta n\rho^8+n^2/\Delta+nA^2/\Delta+\Delta^3A^4n\right).\end{aligned}$$

Balancing cost by setting parameters $\Delta=n^{1/20}$, $\rho=\Delta^2$ and $A=\Delta^4$ then yields $O^*(n^{2-1/20})$.

**Theorem 7.8.** *Computing eccentricities of an $n$-node unit-disk graph can be done in $O^*(n^{2-1/20})$ time.*

## 7.4 Analysis for Diameter

For the special case of computing the diameter of unit-disk graphs, we can get a slight improvement in running time by making the following observation:

> In the analysis for the all-vertex eccentricities algorithm, computing $\rho$-stabbing paths in Step 2 and Step 8 using Lemma 7.4 takes $\widetilde{O}(n\cdot\rho)$ time per piece per $r$, which is the bottleneck. We can instead compute a *global* $\rho$-stabbing path per type for both the combined set system and the auxiliary set system at the start of each iteration of the inner-loop, then restrict these stabbing paths to each piece $P$.

More specifically, at the start of iteration $T$:

0.1. Consider the *global combined set system* $\left(V_G,\left\{N^{r-1}_{\le\sigma}[v]:v\in V_G\right\}\cup\left\{N^r_T[v]:v\in V_G\right\}\right)$. This set system differs from $\mathcal{S}^{r-1}_{\le\sigma}\cup\mathcal{S}^r_T$ in two places: the neighborhood balls are *not* modified, and the ball centers range over all vertices in $G$, not just in $P$.

Compute a $\rho$-sampling $\check{\mathcal{R}}^r_T$ of $\left(V_G,\left\{N^r_T[v]:v\in V_G\right\}\right)$, then take union with the $\rho$-sampling of $\left(V_G,\left\{N^{r-1}_{\le\sigma}[v]:v\in V_G\right\}\right)$ computed from the previous round $r-1$ to form a $\rho$-sampling $\check{\mathcal{R}}^\flat$ of the global combined set system.

0.2. Compute an $\check{\mathcal{R}}^\flat$-respecting $\rho$-stabbing path $\check{\lambda}^\flat$ along with the equivalence classes of $\equiv_{\check{\lambda}^\flat}$ for the global combined set system using Lemma 7.4.

0.3. Consider the *global auxiliary set system*

$$\left(V_G,\left\{N^r_{\le T-1}[v]:v\in V_G\right\}\cup\left\{N^r_T[v]:v\in V_G\right\}\cup\left\{N^r_{\le T}[v]:v\in V_G\right\}\right).$$

Compute a $\rho$-sampling $\check{\mathcal{R}}^r_{\le T}$ of $\left(V_G, \left\{N^r_{\le T}[v] : v \in V_G\right\}\right)$, then take union with the $\rho$-sampling of $\left(V_G, \left\{N^r_{\le T-1}[v] : v \in V_G\right\}\right)$ computed from the previous iteration $T-1$ and $\rho$-sampling of $\left(V_G, \left\{N^r_T[v] : v \in V_G\right\}\right)$ from Step 0.1 to form a $\rho$-sampling $\check{\mathcal{R}}^\sharp$ of the global auxiliary set system.

0.4. Compute an $\check{\mathcal{R}}^\sharp$-respecting $\rho$-stabbing path $\check{\lambda}^\sharp$ along with the equivalence classes of $\equiv_{\check{\lambda}^\sharp}$ for the global auxiliary set system using Lemma 7.4.

Then, for each piece $P$, we modify the following steps in iteration $T$:

2. Restrict the $\check{\mathcal{R}}^\flat$-respecting $\rho$-stabbing path $\check{\lambda}^\flat$ to another stabbing path $\lambda^\flat$ for the combined set system $\mathcal{S}^{r-1}_{\le\sigma} \cup \mathcal{S}^r_T$ for piece $P$. This is done by first removing every neighborhood ball $N^r_T[v]$ not centered in $P$, then taking intersection between $N^r_T[v]$ and the relevant region $R_P$ to form $\hat{N}^r_T[v]$.

8. Restrict the $\check{\mathcal{R}}^\sharp$-respecting $\rho$-stabbing path $\check{\lambda}^\sharp$ to another stabbing path $\lambda^\sharp$ for the auxiliary set system $\mathcal{S}^{r-1}_{\le\sigma} \cup \mathcal{S}^r_T$ for piece $P$.

**Analysis.** We only count for the new changes in the diameter case; for the remaining steps, see the time analysis for computing eccentricities.

In the new Step 2, the removal of balls not centered in $P$ does not increase the stabbing number of $\check{\lambda}^\flat$. Taking intersection with $R_P$ does not change the stabbing number, because this is equivalent to restricting the stabbing path range from $[1:n]$ to $R_P$ (in the same order), and what was one interval in $[1:n]$ remains one interval in $R_P$. So $\lambda^\flat$ is still a $\rho$-stabbing path. The $\rho$-sampling $\mathcal{R}^\flat$ can be obtained by restricting $\check{\mathcal{R}}^\flat$ to $\mathcal{S}^{r-1}_{\le\sigma} \cup \mathcal{S}^r_T$. The removal of balls not centered in $P$ only decreases the number of sets in consideration and thus makes $\equiv_{\lambda^\flat}$ coarser than $\equiv_{\check{\lambda}^\flat}$. Taking intersection with $R_P$ does not change the status of $\hat{N}^r_T[v]$ being chosen in the sample or not. Thus $\lambda^\flat$ remains $\mathcal{R}^\flat$-respecting. As a result, $\lambda^\flat$ is an $\mathcal{R}^\flat$-respecting $\rho$-stabbing path.

For the new Step 8, using similar reasoning, $\lambda^\sharp$ is an $\mathcal{R}^\sharp$-respecting $\rho$-stabbing path.

Steps 0.2 and 0.4 take $\widetilde{O}(n\cdot\rho)$ time. Steps 2 and 8 now take $O(n)$ time to carry out the restriction. (Only the part about restricting $[1:n]$ to $R_P$ needs to be implemented, not the removal of balls centered outside $P$.) Overall the neighborhood growing step can be implemented in $O^*(n+|P|\cdot(n/\rho+\rho^8))$ time per piece, plus another $\widetilde{O}(n\rho)$ time across all pieces.

The final running time of the diameter algorithm for unit-disk graphs is:

$$O^*\left(\Delta\cdot n\rho + \sum_{P:|P|>A}\left(|\partial P|\cdot n + \Delta\cdot(n+|P|\cdot(n/\rho+\rho^8))\right) + \sum_{P:|P|\le A}\left(n\cdot|\partial P| + |P|\cdot(|P|+(|\partial P|\Delta)^4)\right)\right)$$
$$= O^*\left(n^2/\Delta + \Delta n\rho + \Delta n^2/A + \Delta n^2/\rho + \Delta n\rho^8 + n^2/\Delta + nA^2/\Delta + \Delta^3A^4n\right).$$

Balancing cost by setting parameters $\Delta = n^{1/18}$ and $\rho = A = \Delta^2$ then yields $O^*(n^{2-1/18})$.

**Theorem 7.9.** *Computing diameter of an $n$-node unit-disk graph can be done in $O^*(n^{2-1/18})$ time.*

# 8 Framework for Distance Oracles (and Wiener Index)

Our algorithm for computing the Wiener index is a simple extension of our algorithm for computing the exact distance oracle. Therefore, in the following, we focus exclusively on describing the framework for computing an exact distance oracle. Then we give more details on how to compute the Wiener index with the same running time.

For distance oracles, the first two steps are the same as in the framework for eccentricities described in Section 3. Nonetheless, we present a full description of the framework since there are significant differences in step 3. The key difference is that instead of a specific set of relevant vertices $R_P$ for piece $P$, we need to consider all vertices $V_G$. The distances we need to consider will vary depending on the vertex $t \in V_G$, and therefore, we could not use the same definition of $\hat{N}^r[s]$ in the diameter computation for distance oracles. However, we observe that since we have a good additive estimate $\hat{d}$ that is within $\pm\Delta$ of the true distance between $t \in V_G$ and a vertex $s \in P$, we only need to consider distances in an $O(\Delta)$ range around $\hat{d}$. Our idea is to add a weight to each vertex $t$ and use vertex weights to define $\hat{N}^r[s]$ (Equation (7)).

In our oracle construction, it is important to distinguish between large and small pieces (determined by some size threshold) in the LDD $\mathcal{L}$. For large pieces, we will use the interval representation. For small pieces, we use the oracle construction of Lemma 2.16.

**Oracle construction.** Let $A$ be the parameter chosen later. For each piece $P$ in an LDD $\mathcal{L}$.

1. Compute a low-diameter decomposition $\mathcal{L}$ of $G$ with a diameter parameter $\Delta > 0$.
2. For each vertex $v \in \bigcup_{P\in\mathcal{L}} \partial P$ compute a breath-first search tree in $G$ rooted at $v$.
3. For each piece $P \in \mathcal{L}$ where $|P| > A$, let $s_P$ be an arbitrary vertex of $\partial P$. For every vertex $v \in V_G$, we compute and store a weight $w_P(v) = d_G(v, s_P)$. Observe by the triangle inequality that for any vertex $s \in P$:
$$w_P(v) - \Delta \le d_G(s, v) \le w_P(v) + \Delta$$
Now we define an adjusted neighborhood ball as follows:
$$\hat{N}^r[s] := \{v \in V : d_G(v, s) \le r + w_P(v)\} \qquad \forall r \in [-\Delta, \Delta] \tag{7}$$
Then we compute $\hat{N}^r[s]$ with the ball expansion data structure $\mathcal{D}$ and store all intermediate balls in the following procedure:
   - 3.1 For every $s \in \partial P$, we can explicitly compute the modified balls $\hat{N}^r[s]$ for all $r \in [-\Delta, \Delta]$ as well as compute and store a compact interval representation with respect to an ordering $\lambda$.
   - 3.2 As a base case, we initialize $\hat{N}^r[s] = \varnothing$ for every $s \in P$ when $r = -\Delta - 1$.
   - 3.3 For other values of $r \in [-\Delta, \Delta]$, compute $\mathsf{Rep}_\lambda(\hat{N}^r[s])$ using the inductive formula
$$\hat{N}^r[s] = \bigcup_{v\in N[s]} \hat{N}^{r-1}[v] \tag{8}$$
   by taking the union of the intervals.
4. For each piece $P \in \mathcal{L}$ with $|P| < A$, construct the distance oracle of Lemma 2.16.

**Correctness of ball expansion initialization.** Since $d_G(s_P, t) - \Delta \le d_G(s, t) \le d_G(s_P, t) + \Delta$ for every $t \in V$, $t \notin \hat{N}^{-\Delta-1}[s]$ and $t \in \hat{N}^{\Delta}[s]$. Thus, the initialization is correct, and we have correctly computed the desired modified neighborhood balls.

**Answering queries.** Suppose we get a distance query between a vertex $s$ that is in a piece $P \in \mathcal{L}$, and any other vertex $t \in G$. If $|P| < A$, we query the distance oracle for small pieces, and by Lemma 2.16 the query time is $O(\log n)$. Otherwise, for any $r \in [-\Delta, \Delta]$, we can detect if $t \in \hat{N}^r[s]$ by checking if $t$ lies in an interval of $\mathsf{Rep}_\lambda(\hat{N}^r[s])$ by binary search in $O(\log n)$ time[14]. Thus, we can binary search for the first radius $r_t$ such that $t \in \hat{N}^{r_t}[s]$ and $t \notin \hat{N}^{r_t-1}[s]$. By the definition of $\hat{N}$ of Equation (7), we can conclude:

$$d_G(s,t) = d_G(t,s_p) + r_t.$$

In either case, we spend $\widetilde{O}(1)$ time.

**Computing the Wiener index.** In the oracle construction, we compute and store $\hat{N}^r[s]$ for every $s$ in a large piece $P$. For every vertex $t \in \hat{N}^r[s] \setminus \hat{N}^{r-1}[s]$, the exact distance from $s$ to $t$ is $d_G(t,s_p) + r$, and hence $\sum_{t \in \hat{N}^r[s] \setminus \hat{N}^{r-1}[s]} d_G(s,t) = \sum_{t \in \hat{N}^r[s] \setminus \hat{N}^{r-1}[s]} d_G(t,s_p) + |\hat{N}^r[s] \setminus \hat{N}^{r-1}[s]| \cdot r$. This allows us to compute $\sum_{v \in V} d_G(s,v)$ in the same running time as it takes to construct the interval representation of $\{\hat{N}^r[s]\}_{r=-\Delta}^{\Delta}$. For small pieces, Le and Wulff-Nilsen [LW24] provided an algorithm for computing $\sum_{v \in V} d_G(s,v)$ that has the same running time as the construction time for exact oracles of small pieces. Therefore, the time to compute the Wiener index is the same as the time to construct an exact distance oracle.

**Organization.** In the next four sections, we will apply our framework to devise algorithms for exact distance oracles (and thus Wiener index) for different graph classes: sparse graphs of bounded VC-dimension (Section 9), arbitrary-square graphs (Section 10), unit-square graphs (Section 11), and unit-disk graphs (Section 12). There will be similarities with earlier sections on diameter (Sections 4–7).

# 9 Distance Oracles for Sparse Graphs of Bounded VC-dimension

We begin by considering sparse graphs of bounded VC-dimension.

**Stabbing path construction.** For a piece $P \in \mathcal{L}$, let $\mathrm{vol}(P) = \sum_{s \in P} \deg(s)$ be the total degree of vertices in $P$, i.e., the *volume* of $P$. We will construct a stabbing path $\lambda_P$ for each piece $P \in \mathcal{L}$ satisfying $\widetilde{O}(1) \cdot \sum_{r=-\Delta}^{\Delta} \sum_{s \in P} \deg(s) \cdot |\mathsf{Rep}_{\lambda_p}(\hat{N}^r[s])| = \widetilde{O}(\Delta \,\mathrm{vol}(P)(n/\rho + \rho^{d-1}))$ for a parameter $\rho$ to be specified later using Lemma 7.4.

**Construction time.** Computing the low diameter decomposition and the boundary distances stored in step 2 takes $\widetilde{O}(mn/\Delta)$ time. For large pieces, the total construction time involves computing the ordering $\lambda$ in $\widetilde{O}(m\rho)$ time (by Lemma 7.4) and the ball expansion procedure which takes $O(\Delta \,\mathrm{vol}(P) \cdot (n/\rho + \rho^{d-1}))$ time per piece. Thus, the total running time is:

$$\sum_{\substack{P \in \mathcal{L} \\ |V_p| \geq A}} \widetilde{O}(m\rho + \Delta \,\mathrm{vol}(P) \cdot (n/\rho + \rho^{d-1})) \;=\; \widetilde{O}(nm\rho/A + \Delta mn/\rho + \Delta m\rho^{d-1}).$$

For small pieces, we observe that we can compute a vertex weighted BFS on $P$ with weights at most $\Delta$ in time $\widetilde{O}(\mathrm{vol}(P))$. Therefore, in Lemma 2.16, $T(P) = \widetilde{O}(\mathrm{vol}(P))$, giving the total running time for all the small pieces:

$$\begin{aligned}\sum_{\substack{P \in \mathcal{L} \\ |V_p| \leq |A|}} O(n|\partial P| + (|\partial P|^d \Delta^d + |P|) \cdot T(P)) \;&=\; \widetilde{O}(n^2/\Delta) + \widetilde{O}(A^d \Delta^d + A) \cdot \sum_{P \in \mathcal{L}} \mathrm{vol}(P) \\ &=\; \widetilde{O}(n^2/\Delta + mA^d \Delta^d).\end{aligned}$$

[14] We can reduce this running time to $O(1)$ by using the fractional cascading technique; this would complicate the details.

The total running time for the algorithm is:

$$\widetilde{O}(mn/\Delta + nm\rho/A + \Delta mn/\rho + \Delta m\rho^{d-1} + mA^d\Delta^d) = O(mn^{1-1/(4d+1)})$$

for $\Delta = n^{1/(4d+1)}$, $\rho = \Delta^2$, and $A = \Delta^3$.

**Space usage.** The boundary distances that we store in step 2 take $\widetilde{O}(n^2/\Delta)$ space. For large pieces, in step 3, we use $\widetilde{O}(\Delta \operatorname{vol}(P)(n/\rho + \rho^{d-1}))$ space to store compact representations of the neighborhood balls and $O(n)$ space to store distances from each vertex to $s_P$. We also use $O(n)$ space per boundary vertex to store $\hat{N}^r[s]$ for all $r \in [-\Delta, \Delta]$ in step 3(0) by storing $\{\hat{N}^r[s] \setminus \hat{N}^{r-1}[s]\}$ for every $r$. Thus, the total space is:

$$\begin{aligned}\sum_{\substack{P\in\mathcal{L}\\ |V_P|\geq|A|}} \widetilde{O}(n\cdot|\partial P| + \Delta\operatorname{vol}(P)(n/\rho+\rho^{d-1})) &= \widetilde{O}(n^2/\Delta + \Delta m(n/\rho+\rho^{d-1}))\\ &= \widetilde{O}(n^2/\Delta + mn/\Delta + m\Delta^{2d-1}) \quad (\text{since } \rho=\Delta^2).\end{aligned}$$

For each small piece, step 4 requires $O(n|\partial P| + |V_P|^d)$ space by Lemma 2.16. The total space required for all small pieces is

$$\sum_{\substack{P\in\mathcal{L}\\ |V_P|\leq|A|}} O(n|\partial P| + |V_P|^d) = \widetilde{O}(n^2/\Delta + nA^{d-1}) = \widetilde{O}(n^2/\Delta + n\Delta^{3d-3}) \quad \text{space as} \quad A=\Delta^3.$$

Therefore, the total space of our oracle is:

$$\widetilde{O}\left(n^2/\Delta + mn/\Delta + m\Delta^{2d-1} + n\Delta^{3d-3}\right) = \widetilde{O}(mn^{1-1/(4d+1)})$$

for $\Delta = n^{1/(4d+1)}$.

**Theorem 9.1.** *Given undirected graph $G$ with $n$ vertices and $m$ edges that has generalized distance VC-dimension at most $d$, we can construct in $\widetilde{O}(mn^{1-1/(4d+1)})$ time an exact distance oracle of $\widetilde{O}(mn^{1-1/(4d+1)})$ space and $\widetilde{O}(1)$ query time.*

**Remark 9.2.** We chose our parameters to minimize the construction time. We can trade off between space and query time. In the extreme, if construction time does not matter, we can apply the large piece solution to all pieces to obtain a distance oracle using $\widetilde{O}(mn^{1-1/(2d)})$ space.

**Remark 9.3.** In this section, we assumed the graph has a bounded distance VC-dimension. The exponent can be slightly optimized when the time it takes to perform BFS in a piece $P$ is $O(|P|)$ instead of $O(\operatorname{vol}(P))$. This is the case for minor-free graphs, where the space can be improved to $\widetilde{O}(mn^{1-1/(4d)})$. We can also obtain similar results (albeit with worse exponents) if we make other bounds on VC-dimension, such as the distance VC-dimension, and even if we only assume that the $k$-neighborhood VC-dimension is bounded by $d$ for all $k$.

# 10 Distance Oracles for Square Graphs

For square graphs, we follow the construction of the oracle in Section 8. We note that the VC-dimension $d = 4$ in this case. We only analyze the construction time since space is bounded by it.

**Stabbing path construction.** For a piece $P \in \mathcal{L}$. We will construct a stabbing path $\lambda_P$ for each piece $P \in \mathcal{L}$ satisfying:

$$\widetilde{O}(1) \cdot \sum_{r=-\Delta}^{\Delta} \sum_{s \in P} |\mathsf{Rep}_{\lambda_p}(\hat{N}^r[s])| = \widetilde{O}(\Delta |P|(n/\rho + \rho^3)) \tag{9}$$

for a parameter $\rho$ to be specified later using Lemma 7.4. The running time is $\widetilde{O}(\rho n)$ as we show in Appendix A that we can find a BFS tree in square graphs in $\widetilde{O}(n)$ time.

Given the interval representation $\{\hat{N}^{r-1}[s]\}_{s \in P}$ for radius $r-1$, we compute the interval representation of $\{\hat{N}^r[s]\}_{s \in P}$ using the data structure $\check{\mathcal{D}}$ for the interval search problem for squares (Lemma 5.2) in the eccentricities computation with the same setup: the input contains a set of squares corresponding to vertices of $P$ and the interval representation $\{\mathsf{Rep}_{\lambda_p}(\hat{N}^{r-1}[s])\}_{s \in P}$ for radius $r-1$. The queries are $\{\textsc{IntervalSearch}(s) : s \in P\}$ whose outputs are the interval representations of $\{\hat{N}^r[s]\}_{s \in P}$. The total time to grow balls for all radii, using the same efficient encoding as in the improved algorithm for computing eccentricities in Section 5, is:

$$\begin{aligned} &\sum_{r=-\Delta}^{\Delta} \widetilde{O}(b \cdot \sum_{s \in P} (|\mathsf{Rep}_{\lambda_p}(\hat{N}^{r-1}[s])| + |\mathsf{Rep}_{\lambda_p}(\hat{N}^r[s])|)) + \widetilde{O}(|P|n/b) \\ =\ &\widetilde{O}(b\Delta |P|(n/\rho + \rho^3) + |P|n/b) \quad \text{(by Equation (9))}. \end{aligned} \tag{10}$$

**Construction time.** For small pieces, we show (in Lemma E.3 in the appendix) that we can compute a vertex weighted BFS on $P$ with weights at most $\Delta$ in time $\widetilde{O}(|P|)$. Therefore, in Lemma 2.16, $T(P) = \widetilde{O}(|P|)$, giving the total running time for all the small pieces:

$$\begin{aligned} \sum_{\substack{P \in \mathcal{L} \\ |V_P| \le |A|}} \widetilde{O}(n|\partial P| + (|\partial P|^4 \Delta^4 + |P|) \cdot |P|) &= \sum_{P \in \mathcal{L}} \widetilde{O}(n|\partial P| + A^4 \Delta^4 \cdot |\partial P|) \\ &= \widetilde{O}(n^2/\Delta + nA^4\Delta^3). \end{aligned}$$

For large pieces, the running time to grow balls (Equation (10)) plus the running time of $\widetilde{O}(n\rho)$ to compute $\lambda_P$ for each piece $P$ is:

$$\sum_{\substack{P \in \mathcal{L} \\ |V_P| \ge |A|}} \widetilde{O}(n\rho + b\Delta |P|(n/\rho + \rho^3) + |P|n/b) = \widetilde{O}(n^2\rho/A + b\Delta n(n/\rho + \rho^3) + n^2/\rho).$$

Therefore, the total running time to construct the oracle is:

$$\widetilde{O}(n^2/\Delta + nA^4\Delta^3 + n^2\rho/A + b\Delta n(n/\rho + \rho^3) + n^2/\rho) = \widetilde{O}(n^{2-1/20})$$

by setting $b = \Delta = n^{1/20}$, $\rho = \Delta^3$, $A = \Delta^4$.

**Theorem 10.1.** *Given a square graph with $n$ vertices, we can construct in $\widetilde{O}(n^{2-1/20})$ time an exact distance oracle of $\widetilde{O}(n^{2-1/20})$ space and $\widetilde{O}(1)$ query time.*

# 11 Distance Oracles for Unit-square Graphs

For unit square graphs, we follow the oracle construction for square graphs above. The only difference is that we use Lemma 6.1 for solving the interval searching problem for unit squares. Therefore, the running time to construct all the intervals is within an $n^{o(1)}$ factor of the total number of intervals. Since the stabbing path $\lambda_P$ for each piece $P$ still satisfies Equation (9), the total running time to grow all

the balls for each large piece is $O^*(\Delta|P|(n/\rho+\rho^3))$. Therefore, the construction time for large pieces becomes:

$$\sum_{\substack{P\in\mathcal{L}\\ |V_p|\geq A}} O^*(n\rho+\Delta|P|(n/\rho+\rho^3)) \;=\; O^*(n^2\rho/A+\Delta n^2/\rho+\Delta n\rho^3).$$

The construction time for small pieces is the same: $\widetilde{O}(n^2/\Delta+nA^4\Delta^3)$. Thus, the total construction time of the oracle is:

$$O^*(n^2/\Delta+nA^4\Delta^3+n^2\rho/A+\Delta n^2/\rho+\Delta n\rho^3) \;=\; O^*(n^{2-1/16})$$

for $\Delta=n^{1/16}, A=\Delta^3, \rho=\Delta^2$.

**Theorem 11.1.** *Given a unit square with $n$ vertices, we can construct in $\widetilde{O}(n^{2-1/16})$ time an exact distance oracle of $\widetilde{O}(n^{2-1/16})$ space and $\widetilde{O}(1)$ query time.*

# 12 Distance Oracles for Unit-disk Graphs

For unit-disk graphs, we follow the same strategy in Section 7, adapted to the distance oracle framework Section 8.

- We partition the neighborhood balls into types, so that within any cell, balls of a fixed type intersect the cell as a pseudoline arrangement.
- We use the same geometric data structure (Appendix C.3) and interval representation switching technique (Appendix D), to implement the ball growing step (Step 3.3) in the framework using the inductive formula (1).
- We switch to a different distance oracle construction using Lemma 2.16 when the piece has size at most $A$.

We only analyze the construction time since space is bounded by it.

**Ball expansion step.** We now need to deal with modified balls of a fixed radius $r$ for different types with vertex weights on their endpoints. To be precise:

$$\hat{N}^r_M[s] := \{(v,w(v)) : v\in V \text{ where the } \tau\text{-walk from } s \text{ to } v \text{ is at most } r+w(v) \text{ for } \tau\in M\}$$

We bound the dual VC-dimension of the set system $((v,w(v))_{v\in V}, \{N^r_M[s]\}_{s\in P})$ in Lemma B.3.

For each type $T$, given the $\lambda_T^{r-1}$-representation for every modified balls in the set system $\mathcal{S}_T^{r-1}$, we compute the $\lambda_T^r$-representation for every modified balls in the set system $\mathcal{S}_T^r$, using the same interval representation switching strategy and the data structure $\mathcal{D}_T^r$ for the interval cover problem for unit-disks, similar to Section 7.2. The total time to grow balls for all radii is $n\cdot\rho+|P|\cdot(n/\rho+\rho^8)$.

**Construction time.** For small pieces, we show (in Observation E.2 in the appendix) that we can compute a vertex weighted BFS on $P$ with weights at most $\Delta$ in time $\widetilde{O}(|P|)$. Therefore, in Lemma 2.16, $T(P)=\widetilde{O}(|P|)$, giving the total running time for each small piece to be $n\cdot|\partial P|+|P|\cdot(|P|+(|\partial P|\Delta)^4)$.

For large pieces, we grow the balls for $O(\Delta)$ rounds, each taking time $n\cdot\rho+|P|\cdot(n/\rho+\rho^8)$. Therefore, the total running time to construct the oracle is:

$$O^*\left(n^2/\Delta+\sum_{P:|P|>A}\Delta(n\cdot\rho+|P|\cdot(n/\rho+\rho^8))+\sum_{P:|P|\leq A}\left(n\cdot|\partial P|+|P|\cdot(|P|+(|\partial P|\Delta)^4)\right)\right)$$

$$= O^*\left(n^2/\Delta + \Delta n^2\rho/A + \Delta n^2/\rho + \Delta n\rho^8 + n^2/\Delta + nA^2/\Delta + \Delta^3 A^4 n\right).$$

Balancing cost by setting parameters $\Delta = n^{1/20}$, $\rho = \Delta^2$ and $A = \Delta^4$ then yields $\widetilde{O}(n^{2-1/20})$.

**Theorem 12.1.** *Given a unit-disk graph with $n$ vertices, we can construct in $\widetilde{O}(n^{2-1/20})$ time an exact distance oracle of $\widetilde{O}(n^{2-1/20})$ space and $\widetilde{O}(1)$ query time.*

# 13 Conclusion and Open Questions

In this paper, we have presented the first truly subquadratic algorithms for diameter and related problems for many classes of geometric intersection graphs. Naturally, many open questions follow, for example, improving the exponents of the time bounds of any of our algorithms. More intriguingly:

- Is there a truly subquadratic algorithm for computing the diameter of arbitrary disk graphs? Our algorithm can be extended to the case when the number of different radii is $n^{o(1)}$, but the general case appears more difficult.
- Could we prove any conditional lower bound on the running time of the form $\Omega(n^{1+\delta})$ for computing the diameter of unit-disk graphs? Bringmann *et al.* [BKK+22] proved a near-quadratic conditional lower bound for 3D unit-ball graphs under the orthogonal vector (OV) hypothesis.

  If one considers more difficult problems than diameter, e.g., counting the number of pairs with shortest-pair distance at most $r$ (which can be solved by our algorithms in subquadratic time), an $\Omega(n^{4/3})$ conditional lower bound follows for unit-disk graphs if one believes certain offline range searching problems similar to Hopcroft's problem require $\Omega(n^{4/3})$ time (namely, counting the number of pairs of points with Euclidean distance at most 1 in $\mathbb{R}^2$).
- Is there a near-linear-time algorithm for distinguishing between diameter 2 vs. 3 for unit-disk graphs? Bringmann *et al.* [BKK+22] proved a near-quadratic conditional lower bound for 12D unit-hypercube graphs under the hyperclique hypothesis, and obtained an $O(n \log n)$-time algorithm for unit-square graphs.

There are a few specific open questions related to our algorithms. For example:

- Is the VC-dimension of the set system in Lemma 7.2 bounded when we do not restrict to a fixed $r$ and $T$? If so, this might simplify our algorithms for unit disks.
- Could we solve the interval cover data structure problem (Problem 1.3) for arbitrary squares with $N^{1+o(1)}$ preprocessing time and $N^{o(1)}$ query time? If so, this would improve the exponent for our algorithms for arbitrary squares. This appears difficult.
- Less importantly, on the interval cover problem data structure problem for unit disks from a fixed modulo class, could the extra $2^{O(\sqrt{\log N \log \alpha(N)})} \leq N^{o(1)}$ factors be reduced to polylogarithmic? A related question is to determine tight bounds on the combinatorial complexity of the "generalized envelopes" from Appendix C.3.

Besides unit squares, Duraj, Konieczny, and Potępa [DKP24] also considered translates of a convex polygon with constant complexity. It is not difficult to similarly extend our algorithms for unit/arbitrary squares to translates/homothets of other convex polygonal shapes with constant complexity (and our algorithms for unit disks to translations of fat convex non-polygonal shapes with constant complexity).

# References


[ACIM99] Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal On Computing*, 28(4):1167–1181, 1999. doi:10.1137/S0097539796303421.

[ACM+21] A. Karim Abu-Affash, Paz Carmi, Anil Maheshwari, Pat Morin, Michiel Smid, and Shakhar Smorodinsky. Approximating maximum diameter-bounded subgraph in unit disk graphs. *Discrete & Computational Geometry*, 66(4):1401–1414, 2021. doi:10.1007/s00454-021-00327-y.

[AdT24] Boris Aronov, Mark de Berg, and Leonidas Theocharous. A clique-based separator for intersection graphs of geodesic disks in $R^2$. In *40th International Symposium on Computational Geometry (SoCG)*, volume 293, pages 9:1–9:15, 2024. doi:10.4230/LIPIcs.SoCG.2024.9.

[ADW+25] Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication. In *36th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2005–2039, 2025. doi:10.1137/1.9781611978322.63.

[AE99] Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, pages 1–56. AMS Press, 1999.

[AKPW95] Noga Alon, Richard M. Karp, David Peleg, and Douglas West. A graph-theoretic game and its application to the $k$-server problem. *SIAM Journal on Computing*, 24(1):78–100, 1995. doi:10.1137/s0097539792224474.

[AS00] Pankaj K. Agarwal and Micha Sharir. Davenport-Schinzel sequences and their geometric applications. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 1–47. North-Holland, Amsterdam, 2000. doi:10.1016/B978-044482537-7/50002-4.

[Awe85] Baruch Awerbuch. Complexity of network synchronization. *Journal of The ACM*, 32(4):804–823, 1985. doi:10.1145/4221.4227.

[BCE15] Drago Bokal, Sergio Cabello, and David Eppstein. Finding all maximal subsequences with hereditary properties. In *31st International Symposium on Computational Geometry (SoCG)*, volume 34 of *LIPIcs*, pages 240–254, 2015. doi:10.4230/LIPICS.SOCG.2015.240.

[BDG+14] Michael J. Bannister, William E. Devanny, Michael T. Goodrich, Joseph A. Simons, and Lowell Trott. Windows into geometric events: Data structures for time-windowed querying of temporal point sets. In *26th Canadian Conference on Computational Geometry (CCCG)*, 2014. URL: http://www.cccg.ca/proceedings/2014/papers/paper02.pdf.

[BKK+22] Karl Bringmann, Sándor Kisfaludi-Bak, Marvin Künnemann, André Nusser, and Zahra Parsaeian. Towards sub-quadratic diameter computation in geometric intersection graphs. In *38th International Symposium on Computational Geometry (SoCG)*, pages 21:1–21:16, 2022.

[BKK+24] Alexander Baumann, Haim Kaplan, Katharina Klost, Kristin Knorr, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Dynamic connectivity in disk graphs. *Discrete & Computational Geometry*, 71(1):214–277, 2024. doi:10.1007/s00454-023-00621-x.

[BNW22] Aaron Bernstein, Danupon Nanongkai, and Christian Wulff-Nilsen. Negative-weight single-source shortest paths in near-linear time. In *63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 600–611, 2022. doi:10.1109/focs54457.2022.00063.

[BT15] Nicolas Bousquet and Stéphan Thomassé. VC-dimension and Erdős–Pósa property. *Discrete Mathematics*, 338(12):2302–2317, 2015. doi:10.1016/j.disc.2015.05.026.

[Cab18] Sergio Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances in planar graphs. *ACM Transactions on Algorithms*, 15(2):1–38, December 2018.

[CEV07] Victor Chepoi, Bertrand Estellon, and Yann Vaxes. Covering planar graphs with a fixed number of balls. *Discrete & Computational Geometry*, 37(2):237–244, 2007. doi:10.1007/s00454-006-1260-0.

[CG86a] Bernard Chazelle and Leonidas J Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1(1-4):133–162, 1986.

[CG86b] Bernard Chazelle and Leonidas J Guibas. Fractional cascading: II. Applications. *Algorithmica*, 1(1-4):163–191, November 1986.

[CGL+23] Panagiotis Charalampopoulos, Paweł Gawrychowski, Yaowei Long, Shay Mozes, Seth Pettie, Oren Weimann, and Christian Wulff-Nilsen. Almost optimal exact distance oracles for planar graphs. *Journal of the ACM*, 70(2):1–50, 2023. doi:10.1145/3580474.

[CGL24] Hsien-Chih Chang, Jie Gao, and Hung Le. Computing diameter+2 in truly-subquadratic time for unit-disk graphs. In *40th International Symposium on Computational Geometry (SoCG)*, pages 38:1–38:14, 2024. doi:10.4230/LIPIcs.SoCG.2024.38.

[Cha86] Bernard Chazelle. Filtering search: A new approach to query-answering. *SIAM J. Comput.*, 15(3):703–724, 1986. doi:10.1137/0215051.

[CHN20] Timothy M. Chan, Qizheng He, and Yakov Nekrich. Further results on colored range searching. In *36th International Symposium on Computational Geometry (SoCG)*, volume 164 of *LIPIcs*, pages 28:1–28:15, 2020. doi:10.4230/LIPICS.SOCG.2020.28.

[CHP19] Timothy M. Chan, John Hershberger, and Simon Pratt. Two approaches to building time-windowed geometric data structures. *Algorithmica*, 81(9):3519–3533, 2019. doi:10.1007/S00453-019-00588-3.

[CJ15] Sergio Cabello and Miha Jejčič. Shortest paths in intersection graphs of unit disks. *Computational Geometry*, 48(4):360–367, 2015. doi:10.1016/j.comgeo.2014.12.003.

[CK97] V. Chepoi and S. Klavžar. The Wiener index and the Szeged index of benzenoid systems in linear time. *Journal of Chemical Information and Computer Sciences*, 37(4):752–755, 1997. doi:10.1021/ci9700079.

[CK09] Sergio Cabello and Christian Knauer. Algorithms for graphs of bounded treewidth via orthogonal range searching. *Computational Geometry*, 42(9):815–824, 2009. doi:10.1016/j.comgeo.2009.02.001.

[CS16] Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In *27th International Symposium on Algorithms and Computation (ISAAC)*, volume 64, pages 24:1–24:13, 2016.

[CS19] Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in geometric intersection graphs. *J. Comput. Geom.*, 10(1):27–41, 2019. doi:10.20382/JOCG.V10I1A2.

[CW89] Bernard Chazelle and Emo Welzl. Quasi-optimal range searching in spaces of finite VC-dimension. *Discrete & Computational Geometry*, 4(5):467–489, 1989. doi:10.1007/BF02187743.

[dBCvKO08] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008. doi:10.1007/978-3-540-77974-2.

[DHV22] Guillaume Ducoffe, Michel Habib, and Laurent Viennot. Diameter, eccentricities and distance oracle computations on $h$-minor free graphs and graphs of bounded (distance) Vapnik–Chervonenkis dimension. *SIAM Journal on Computing*, 51(5):1506–1534, 2022. doi:10.1137/20M136551X.

[dKMT23] Mark de Berg, Sándor Kisfaludi-Bak, Morteza Monemizadeh, and Leonidas Theocharous. Clique-based separators for geometric intersection graphs. *Algorithmica. An International Journal in Computer Science*, 85(6):1652–1678, 2023.

[DKP24] Lech Duraj, Filip Konieczny, and Krzysztof Potępa. Better diameter algorithms for bounded VC-dimension graphs and geometric intersection graphs. In *32nd Annual European Symposium on Algorithms (ESA)*, volume 308, pages 51:1–51:18, 2024.

[EIK01] Alon Efrat, Alon Itai, and Matthew J Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.

[Eri96] Jeff Erickson. New lower bounds for Hopcroft's problem. *Discrete & Computational Geometry*, 16(4):389–418, 1996. doi:10.1007/bf02712875.

[For87] Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987. doi:10.1007/BF01840357.

[Fre87] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16(6):1004–1022, 1987. doi:10.1137/0216064.

[GJRS18] Prosenjit Gupta, Ravi Janardan, Saladi Rahul, and Michiel H. M. Smid. Computational geometry: Generalized (or colored) intersection searching. In *Handbook of Data Structures and Applications*, chapter 67, pages 1042–1057. CRC Press, 2nd edition, 2018. URL: https://www-users.cs.umn.edu/~sala0198/Papers/ds2-handbook.pdf.

[GJS95] Prosenjit Gupta, Ravi Janardan, and Michiel H. M. Smid. Further results on generalized intersection searching problems: Counting, reporting, and dynamization. *Journal of Algorithms*, 19(2):282–317, 1995. doi:10.1006/jagm.1995.1038.

[GKM+21] Paweł Gawrychowski, Haim Kaplan, Shay Mozes, Micha Sharir, and Oren Weimann. Voronoi diagrams on planar graphs, and computing the diameter in deterministic $\tilde{O}(n^{5/3})$ time. *SIAM Journal on Computing*, 50(2):509–554, 2021. doi:10.1137/18M1193402.

[Har11] Sariel Har-Peled. *Geometric Approximation Algorithms*. Number 173. American Mathematical Soc., 2011.

[Kle89] Rolf Klein. *Concrete and Abstract Voronoi Diagrams*, volume 400 of *Lecture Notes in Computer Science*. Springer, 1989. doi:10.1007/3-540-52055-4.

[Klo23] Katharina Klost. An algorithmic framework for the single source shortest path problem with applications to disk graphs. *Computational Geometry*, 111:101979, 2023.

[KRSV08] Haim Kaplan, Natan Rubin, Micha Sharir, and Elad Verbin. Efficient colored orthogonal range counting. *SIAM Journal on Computing*, 38(3):982–1011, 2008. doi:10.1137/070684483.

[KZ25] Adam Karczmarz and Da Wei Zheng. Subquadratic algorithms in minor-free digraphs: (Weighted) distance oracles, decrementai reachability, and more. In *36th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4338–4351, 2025. doi:10.1137/1.9781611978322.147.

[LP19] Jason Li and Merav Parter. Planar diameter via metric compression. In *51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 152–163, Phoenix, AZ, USA and New York, NY, USA, 2019.

[LPS+24] Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, Jie Xue, and Meirav Zehavi. A 1.9999-approximation algorithm for vertex cover on string graphs. In *40th International Symposium on Computational Geometry (SoCG)*, volume 293, pages 72:1–72:11, 2024. doi:10.4230/LIPIcs.SoCG.2024.72.

[LW22] Hung Le and Christian Wulff-Nilsen. Optimal approximate distance oracle for planar graphs. In *62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 363–374, 2022.

[LW24] Hung Le and Christian Wulff-Nilsen. VC set systems in minor-free (di)graphs and applications. In *35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 5332–5360, 2024. doi:10.1137/1.9781611977912.192.

[Pet15] Seth Pettie. Sharp bounds on Davenport-Schinzel sequences of every order. *Journal of The ACM*, 62(5), 2015. doi:10.1145/2794075.

[RW13] Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *45th Annual ACM Symposium on Theory of Computing (STOC)*, pages 515–524, 2013.

[Sau72] N. Sauer. On the density of families of sets. *J. Comb. Theory Ser. A.*, 13(1):145–147, 1972. doi:10.1016/0097-3165(72)90019-2.

[Sei95] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, 1995. doi:10.1006/jcss.1995.1078.

[She72] Saharon Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41(1):247–261, 1972. doi:10.2140/pjm.1972.41.247.

[SJ05] Qingmin Shi and Joseph JaJa. Novel transformation techniques using q-heaps with applications to computational geometry. *SIAM Journal On Computing*, 34(6):1474–1492, 2005. doi:10.1137/S0097539703435728.

[ST86] Neil Sarnak and Robert E. Tarjan. Planar point location using persistent search trees. *Communications of The ACM*, 29(7):669–679, July 1986. doi:10.1145/6138.6151.

[WN09] Christian Wulff-Nilsen. Wiener index and diameter of a planar graph in subquadratic time. In *Proceedings of the 25th European Workshop on Computational Geometry*, pages 25–28, 2009.

# A Low-diameter Decompositions

In this section, we construct the low-diameter decomposition of sparse graphs and geometric intersection graphs. Recall that we use $N^r[v] = \{u : d_G(u,v) \le r\}$ to denote the set of vertices in the neighborhood ball of radius $r$ centered at $v$.

We give an algorithm for computing a low-diameter decomposition as claimed in Section 2.1. Our low-diameter decomposition for graphs is perhaps most similar to the low-diameter decomposition in [AKPW95]; we are not aware of any work stating the exact guarantees with our definition of LDD. We first present the general algorithm and the properties of the LDD. We will discuss the detailed implementation and running time for sparse graphs and geometric intersection graphs separately.

**Basic algorithm.** Let $\phi$ be a parameter with $0 < \phi < 1$. We will choose $\phi := 24 \log n/\Delta$. Start with the entire graph $G_1 = G$. Pick an arbitrary vertex $u \in V$, perform a BFS to compute neighborhood balls centered at $v$: $N^1[v]$, $N^2[v]$, ..., $N^\ell[v]$. Stop when $|N^\ell[v]|/|N^{\ell-2}[v]| \le 1+\phi$, and set $V_1 = N^{\ell-1}[v]$ as one piece in the decomposition. Note that this is guaranteed to eventually happen because when $N^{\ell-2}[v]$ is the entire connected component of $v$, then $N^\ell[v] = N^{\ell-1}[v] = N^{\ell-2}[v]$. We mark the vertices in $N^\ell[v] \setminus N^{\ell-2}[v]$ as boundary vertices[15]. Repeat this procedure on $G_2 = G_1 \setminus V_1$ to find $V_2$, then on $G_3 = G_2 \setminus V_2$, and so on.

**Low diameter property.** First we bound the strong diameter of one such ball $N^{\ell-1}[v]$ that we included in our low diameter decomposition. Observe that for all $2 \le i < \ell$, the ball $N^i[v]$ of radius $i$ satisfies $|N^i[v]| > (1+\phi)|N^{i-2}[v]|$. Since the size of the largest ball is at most $n$, if $\ell$ is odd, we have that:

$$n \ge |N^\ell(v)| \ge |N^{\ell-1}(v)| > (1+\phi)\cdot|N^{\ell-3}(v)| > (1+\phi)^{(\ell-1)/2}\cdot|N^0(v)| = (1+\phi)^{(\ell-1)/2}$$

Taking logarithms on both sides, and using the fact that $\phi < 1$, we obtain:

$$\log n > \frac{\ell-1}{2}\cdot\log(1+\phi) \ge \frac{\ell-1}{2}\cdot(\phi-\phi^2/2) > \frac{(\ell-1)\cdot\phi}{4} = \frac{6(\ell-1)\cdot\log n}{\Delta}$$

Rearranging the inequality yields $\ell \le \Delta/6$. The diameter is at most $2\ell \le \Delta/3$.

**Small boundary property.** Let $N^{r_1}_{G_1}[v_1]$ be the ball of largest radii we compute in $G_1$, $N^{r_2}_{G_2}[v_2]$ the ball in $G_2$, ..., $N^{r_k}_{G_k}[v_k]$ the ball in $G_k$. Observe that when we choose $P_i = G[N^{r_i-1}_{G_i}[v_i]]$, the vertices of $N^{r_i-1}_{G_i}[v_i] \setminus N^{r_i-2}_{G_i}[v_i]$ are potentially boundary vertices $\partial P_i$, and a vertex in $N^{r_i}_{G_i}[v_i] \setminus N^{r_i-1}_{G_i}[v_i]$ is a boundary vertex in $\partial P_j$ for some piece $P_j$ with $j > i$. Assume we have a total of $k$ pieces in the LDD. Thus it can be seen that:

$$\begin{aligned}
\sum_{i=1}^{k}|\partial P_i| &\le \sum_{i=1}^{k}|N^{r_i}_{G_i}[v_i] \setminus N^{r_i-2}_{G_i}[v]| \\
&\le \sum_{i=1}^{k}\phi\cdot|N^{r_i-2}_{G_i}[v_i]| && \text{(since } |N^{r_i}_{G_i}[v_i]| \le (1+\phi)|N^{r_i-2}_{G_i}[v_i]|\text{)} \\
&\le \sum_{i=1}^{k}\phi\cdot|N^{r_i-1}_{G_i}[v_i]| && \text{(since } N^{r_i-2}_{G_i}[v_i] \subseteq N^{r_i-1}_{G_i}[v_i]\text{)} \\
&\le \phi\cdot n = 24n\log n/\Delta.
\end{aligned}$$

[15] The vertices in $N^\ell[v] \setminus N^{\ell-1}[v]$ are boundary vertices of later pieces constructed in the process. The vertices in $N^{\ell-1}[v] \setminus N^{\ell-2}[v]$ are boundary vertices of $V_1$, although there may be other boundary vertices in $N^{\ell-2}(v)$ that we accounted for earlier in the process.

**No small pieces.** To ensure that no piece is small, we will do some post-processing of the pieces obtained from the basic algorithm. We use the following claim.

**Lemma A.1.** *Let $P_i$ be a piece found in the basic LDD algorithm found by taking the vertices $N^{r-1}[v]$ in $G_i$. Either $P_i$ has at least $\Omega(\Delta/\log n)$ vertices, or $P_i$ is an entire connected component of $G_i$.*

**Proof:** Suppose that $|N^r(v)| > |N^{r-2}(v)|$. Then as $(1+\phi)\cdot|N^{r-2}(v)| \geq |N^r(v)| \geq |N^{r-2}(v)|+1$, we conclude that $|N^{r-2}(v)| \geq 1/\phi = O(\Delta/\log n)$, so $P$ has size at least $O(\Delta/\log n)$. Otherwise $|N^r(v)| = |N^{r-1}(v)| = |N^{r-2}(v)|$ and thus $P_i$ is an entire connected component of $G_i$. □

In our post-processing, we will merge $P_i$ with an arbitrary neighboring component. Observe that since $P_i$ is an entire connected component of $G_i$, no later piece $P_j$ with $j > i$ will merge into $P_i$. Now consider a piece $P_j$ with multiple pieces $P_{i_1}, P_{i_2}, \cdots, P_{i_t}$ merging into it in the post-processing step, $j < i_1, \cdots, j < i_t$. Since all pieces have diameter at most $\Delta/3$, the resulting merged $P_j$ has diameter at most $\Delta$.

## A.1 Sparse Graphs

Consider the standard BFS algorithm that computes $N^r[v]$ by adding all neighbors incident to $N^{r-1}[v]$ into a queue. For every vertex $v$, the basic algorithm will add all its neighbors into a queue at most once, so the basic algorithm can be implemented in $O(m+n)$ time. The post-processing step involving merging components can also be done in $O(m+n)$ time.

**Theorem 2.2.** *Let $G$ be a graph with $n$ vertices and $m$ edges. For any parameter $24\log n < \Delta \leq n$, we can compute a low-diameter decomposition for $G$ in $O(m+n)$ time.*

## A.2 Geometric Intersection Graphs

Here we consider geometric intersection graphs of fat pseudo-disks of similar size and squares of varying sizes. A family of objects are called *pseudo-disks* if each one is the interior of a simple closed Jordan curve and two objects are either disjoint, have one object fully inside the other, or properly intersect each other at two boundary points. Disks are by definition pseudo-disks. The geometric intersection graph of a family of pseudo-disks can be considered, combinatorially, as a set of vertices $V$ representing the pseudo disks and two vertices are connected if their corresponding pseudo-disks have non-empty intersection. For the algorithm below, we consider fat pseudo-disks that are of roughly the same size and have constant complexity. Specifically, a fat pseudo-disk is sandwiched between two disks of the same center $p$ of radius $r$ and $R$ with two fixed constants $r, R$ and $r \leq R$ and the boundary can be described by a constant number of algebraic curves. We call this pseudo-disk centered at $p$ as $C_p$. The input to our algorithm consists of the description of a family of $n$ fat pseudo-disks with input size $O(n)$. We assume that one can compute in $O(1)$ time whether two pseudo-disks have an edge or not. The geometric intersection graph of such pseudo-disks can be dense (i.e., having edges of size $\Theta(n^2)$). We show that the low diameter decomposition can still be computed in near linear time, similar to the running time for sparse graphs (Appendix A.1).

Recall that the basic idea is to perform BFS from a vertex $v$ to compute balls centered at $v$: $N^0[v] = \{v\}$, $N^1[v]$, $N^2[v]$, …, $N^\ell[v]$, and stop when $|N^\ell[v]|/|N^{\ell-2}[v]| \leq 1+\phi$. Let $V_1 = N^{\ell-1}[v]$. Then repeat this procedure on $G_2 = G_1 \setminus V_1$ to find $V_2$, then on $G_3 = G_2 \setminus V_2$, and so on.

We have to be careful in implementing the basic idea: we do not want to spend $\widetilde{O}(n)$ time per iteration as the number of iterations could be $\Omega(n)$. This is achievable by not explicitly constructing all the edges, an idea that is generally adopted for computing a breadth-first search tree for geometric intersection graphs (for fat objects of similar sizes) [EIK01, CJ15, CS16]. We use the same algorithm

as in [CGL24] for pseudo-disks of similar sizes. The core step in the BFS is to find the vertices that are exactly $j$-hops away from $v$, denoted by $Y_j$ – from the vertices that are exactly $j-1$ hops away from $v$, $Y_{j-1}$. Put a grid of size $r\sqrt{2}$. Two pseudo-disks with centers in the same grid cell are connected by an edge for sure. Thus, if a pseudo disk centered at $p$ in one cell appears in $Y_{j-1}$, all pseudo-disks centered in the same cell will be included in $Y_j$ if they are not yet covered in $B^{j-1}(v)$. In addition, the other vertices to be included in $Y_j$ will come from cells that have distance at most $2R$ away from cells touched by $Y_{j-1}$. Since $R^2/r^2$ is a constant, we only need to check for each cell touched by $Y_{j-1}$, at most a constant number of nearby cells. This step can be implemented by using an operation called the red-blue intersection problem, which finds all the blue pseudo-disks that intersect at least one red pseudo-disks, where all red pseudo-disks and blue pseudo-disks are separated by a horizontal (or a vertical) line. We use the following lemma from [CGL24].

**Lemma A.2 ([CGL24]).** *In time $O(n_b \log n_b + n_r \alpha(n_r) \log n_r + n_r 2^{\alpha(n_r)})$, we can solve the red-blue intersection problem of $n_r$ pseudo-disks and $n_b$ blue pseudo-disks. Here $\alpha(n)$ is the inverse Ackermann function.*

With this Lemma we can conclude the following theorem.

**Theorem A.3.** *Let $G$ be the intersection graphs of $n$ fat pseudo-disks of similar size. For any parameter $24 \log n < \Delta \leq n$, we can compute a low-diameter decomposition for $G$ in $\widetilde{O}(n)$ time.*

**Proof:** We argue that for the entire algorithm, a non-empty cell in the grid of size $r\sqrt{2}$ is only visited a constant number of times. First, if cell $c$ has a vertex $p \in V_i$ and $p$ is not on the boundary of this piece $V_i$, then all pseudo-disks centered in the cell will be included in $V_i$. After $V_i$ is removed, cell $c$ becomes empty and will not be visited again in later iterations. Therefore, a cell $c$ visited by $V_i$ is only visited again by pieces $V_j$ with $j > i$ if $c$ has only vertices that are at the boundary of $V_i$. That says, the cell $c$ has a neighboring cell $c'$ (within distance $2R$ from $c$), such that $c'$ contains a vertex $p$ of $V_i$ and $p$ is not on the boundary of $V_i$. In this case all vertices in $c'$ are entirely in $V_i$ (or earlier pieces). Thus, after the $i$-th iteration, at least one of the neighboring cells of $c$ is wiped out. Since $c$ has only a constant of such neighboring cells, $c$ is only visited a constant number of times. This finishes the argument. □

As a corollary, since unit squares and unit disks are fat pseudo-disks of similar size, we conclude that a low diameter decomposition can be computed for these classes of intersection graphs in $\widetilde{O}(n)$ time.

**Axis-aligned squares.** We will need a similar theorem for axis-aligned squares (which might not be of similar size.) A BFS on the intersection graph of axis-parallel squares can be done in time $O(n \log n)$ [Klo23], by using data structures developed in [BKK+24]. Again we focus on how to find the objects of $j$-hops away from a starting vertex $v$ from the objects of $j-1$ hops away. When the squares have different sizes, instead of a grid of a single size, one can use a hierarchical structure such as the (compressed) quadtree. Each square is associated with a quad whose size is comparable with its size. Further the compressed quadtree can be decomposed into $O(n)$ canonical paths such that each root to leaf path can be represented by $O(\log n)$ disjoint canonical paths. A canonical path has a smallest cell $\sigma$ and largest cell $\tau$, and is associated with a constant number of regions, classified as inner, middle and outer regions. The inner region is a disk centered at the smallest cell $\sigma$ of the canonical path. Further, each region $A$ is associated with two sets, the first type $S_1(A)$ contains a collection of objects centered inside $A$ that form a clique, and the second type $S_2(A)$ contains objects that intersect with at least one site in $S_1(A)$. A similar red-blue intersection problem can be solved in linear time for axis-parallel squares, assuming sorting along $x$ and $Y$ coordinates is performed already, as shown in [Klo23]. In summary, to implement a BFS step, for each region $A$ touched by the vertices in $Y_{i-1}$, include all objects that are

in $S_1(A)$ and then perform red-blue intersection modules with $A$ against a constant number of other regions. Since each object stays in at most $O(\log n)$ sets of the first type and at most $O(\log n)$ sets of the second type, the total running time carries an extra $O(\log n)$ factor. We can use this algorithm for the low-diameter decomposition and obtain the following.

**Theorem A.4.** *Let $G$ be the intersection graphs of $n$ axis-aligned squares. For any parameter $24 \log n < \Delta \leq n$, we can compute a low-diameter decomposition for $G$ in $\widetilde{O}(n)$ time.*

**Proof:** The same argument as in Theorem A.3 applies here: for each region $A$, either the type one objects $S_1(A)$ are completely included in a piece $V_i$ and this region disappears; or, one of the (constantly many) nearby regions are completely included in $V_i$ and disappears. By a charging argument, each region is only touched a constant number of times. Thus the total running time is in the order of $\widetilde{O}(n)$. □

# B VC-dimension Lemma

In this section, we prove a lemma bounding the VC-dimension of certain set systems (Lemma 7.2) from Section 7, which is needed in our algorithms for unit-disk graphs.

Let $G$ be the geometric intersection graphs of unit-disk graphs. Let $M$ be a subset of vertices, called *a type*. We say that a walk $W$ from a vertex $v$ to a vertex $u$ is a *Type-1 $M$-walk* if the vertex preceding $u$ (the second to last vertex) in the walk is in $M$. We say that the walk is a *Type-2 $M$-walk* if the vertex following $v$ (the second to first vertex) in the walk is in $M$.

For a technical reason explained later, we assume that every vertex in $G$ has a self-loop attached to it. For every vertex $v$, define:

$$\begin{aligned} B_M^{(1)}(v,r) &= \{u \in V \mid \text{there is a Type-1 } M\text{-walk from } v \text{ to } u \text{ of length exactly } r\} \\ B_M^{(2)}(v,r) &= \{u \in V \mid \text{there is a Type-2 } M\text{-walk from } v \text{ to } u \text{ of length exactly } r\} \end{aligned} \tag{11}$$

The reason for attaching a self-loop to every vertex is that if $d_G(v,x) \leq r-1$ for some vertex $x$ in $M$, then $x \in B_M^{(1)}(v,r)$ since we can make a Type-1 $M$-walk of length $r$ by traversing from $v$ to $x$ along the shortest path (of length at most $r-1$) and then along the self-loop to get a walk of length at most $r$. The second to last vertex of the walk is $x$ itself, which is in $M$. Furthermore, if there is a Type-1 $M$-walk from $v$ to $u$ of length less than $r$, then there is a Type-1 $M$-walk from $v$ to $u$ of length exactly $r$ by traveling the self-loop attached to the vertex in $M$ preceding $u$. The same holds for Type-2 $M$-walk. The main result of this section is to show that the system of balls deriving from Type-1 $M$-walk has a bounded VC-dimension.

**Lemma B.1.** $(V, \{B_M^{(1)}(v,r)\}_{r\in\mathbb{R}, v\in V})$ *has VC-dimension at most* 4.

Observe that Type-1 and Type-2 $M$-walks are dual to each other: a Type-1 $M$-walk from $v$ to $u$ of length $r$ is a Type-2 $M$-walk from $u$ to $v$ of length $r$. Therefore, for *a given $r$*, $(V, \{B_M^{(2)}(v,r)\}_{v\in V})$ is the dual set system of $(V, \{B_M^{(1)}(v,r)\}_{v\in V})$, and therefore, has VC-dimension at most $2^4 = 16$. By modifying the proof of Lemma B.1, get an improved bound for balls from Type-2 $M$-walk:

**Lemma B.2.** *For any $r \in \mathbb{N}$, $(V, \{B_M^{(2)}(v,r)\}_{v\in V})$ has VC-dimension at most* 4.

The set system in Lemma B.2 only includes balls of fixed radius. It is possible that the more general set system $(V, \{B_M^{(2)}(v,r)\}_{r\in\mathbb{R}, v\in V})$, which includes all balls of all radii, has VC-dimension at most 4. However, for a technical reason, our proof of Lemma B.1 does not extend to this general case. See Remark B.5 for more details.

For the distance oracle application, we will need to handle vertices with weights. So we define the following set system for weighted vertices. Suppose each vertex $u$ has a weight $w(u)$, and the ground set is $\{(u, w(u))\}_{u\in V}$. Recall for the distance oracle, we maintain the adjusted neighborhood ball as follows (see Section 8 Equation (7), copied below):

$$\hat{N}^r[s] := \{v \in V : d_G(v,s) \le r + w_P(v)\} \qquad \forall r \in [-\Delta, \Delta]$$

Further, the path connecting $s$ (the center of the neighborhood ball) to $v$ has the second vertex (adjacent to $s$) of a special type. Thus, we consider a Type-2 walk from $s$ to $v$. Therefore the set system we work with will be $(\{(u, w(u))\}_{u\in V}, \{B^{(2)}_{M,w}(v, r)\}_{v\in V})$ where

$$B^{(2)}_{M,w}(s, r) = \{v \in V | \text{ there is a Type-2 } M\text{-walk from } s \text{ to } v \text{ of length exactly } r + w(v)\} \tag{12}$$

Take this set system as the primal system, we can define the dual system as follows. Specifically, $v \in B^{(2)}_{M,w}(s, r)$ if and only if $s \in B^{(1)}_{M,w}(v, r)$ where

$$B^{(1)}_{M,w}(v, r) = \{s \in V | \text{ there is a Type-1 } M\text{-walk from } v \text{ to } s \text{ of length exactly } r + w(v)\} \tag{13}$$

Notice that $B^{(1)}_{M,w}(v, r) = B^{(1)}_{M}(v, r + w(v))$. Therefore, the VC-dimension bound we need is provided precisely by Lemma B.1 for Type-1 walks, which fortunately works for neighborhood balls of varying radii. With this we immediately have the following.

**Lemma B.3.** *For any $r \in \mathbb{N}$, $(\{(u, w(u))\}_{u\in V}, \{B^{(2)}_{M,w}(v, r)\}_{v\in V})$ has dual VC-dimension at most 4.*

## B.1 Type-1 $M$-Walks

In this section, we prove Lemma B.1. As all $M$-walks in this section are of Type-1, we will drop the prefix Type-1, and only refer to Type-1 $M$-walks as $M$-walk. We also call the last edge of an $M$-walk to $u$ as an *$M$-edge*.

**Proof (Sketch Proof of Lemma B.1):** The strategy is basically the same as [CGL24][16]. We only show the steps needed for adapting the proof here. Consider four vertices $a, b, c, d$ representing four disks $D_a, D_b, D_c, D_d$ and assume that there are two (Type 1) $M$-walks $P(b, a)$ *from $b$ to $a$* and $P(c, d)$ *from $c$ to $d$*. (The vertices preceding $a$ and $d$ in the walks are in $M$.) We define a *local crossing pattern* to be four distinct vertices $a', b', c', d'$ with $a', b'$ on $P(a, b)$ (with $a'$ closer to $a$ than $b'$) and $c', d'$ on $P(c, d)$ (with $c'$ closer to $c$ than $d'$) such that one of the four vertices $a', b', c', d'$ has edges to all the other three vertices; see Figure 3. The central claim is the following; if the claim holds, then the rest of the argument is standard.

**Claim B.4.** *Either there is an $M$-walk $P'(c, a)$ whose hop length is at most $|P(c, d)|$ or there is an $M$-walk $P'(b, d)$ whose hop length is at most $|P(b, a)|$.*

We consider a case study depending on whether the local crossing pattern involves an $M$-edge. In the first case when the local crossing pattern does not involve the last edge (from a vertex in M to the endpoint of the walk) of the two M-walks (see Figure 3 (a) for an example), the proof follows exactly the same as that of [CGL24]. The second case, which is also easy, is when the local crossing pattern involves two $M$-edges. In this case, both $c', b' \in M$. Either we have the edge $c'a'$ or the edge $b'd'$. In both cases the claim is true. Figure 3 (b) shows the case with edge $c'a$ present. In this case, we can find an $M$-walk

[16] We refer to https://arxiv.org/pdf/2401.12881.

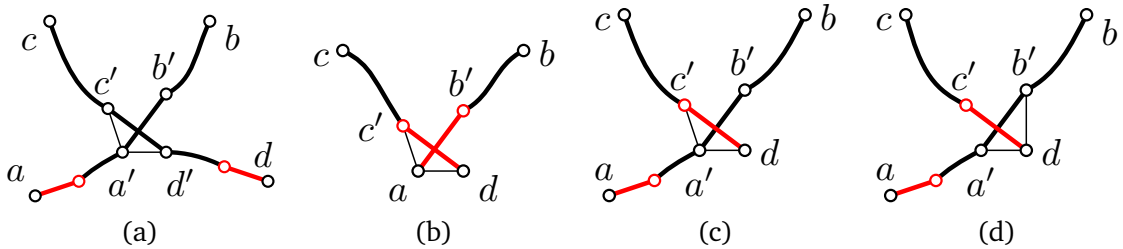


**Figure 3.** If two $M$-walk $P(b,a)$ and $P(c,d)$ intersect with a local crossing pattern $a', b', c', d'$, then there is an $M$-walk from $c$ to $a$ that are no longer than $|P(c,d)|$ or there is an $M$-walk from $b$ to $d$ that is no longer than $P(b,a)$. The vertices in $M$ are highlighted red.

$P'(c,a)$ through $P(c,c')$ and then take edge $c'a$, which is not longer than path $P(c,d)$. If $b'd$ is present, then the path that follows $P(b,b')$ and then edge $b'd$ is an $M$-walk and not longer than $P(b,a)$.

The difficult case for the proof of the claim is when the local crossing pattern happens at an M-edge of one path with the non M-edge part of the other path. Without loss of generality, assume that the $M$-edge involved in a local crossing pattern is the edge $c'd$. See case (c) and (d) in Figure 3 for an example.

We first consider the case when $c'a'$ and $a'd$ are present. We prove by contradiction. Consider an $M$-path from $b$ to $d$:

$$P'(b,d) = P(b,b') \circ (b'a') \circ (a'c') \circ (c'd').$$

Since the claim does not hold, the following holds:

$$|P'(b,d)| > |P(b,a)| \Leftrightarrow |P(b,b')| + 3 > |P(b,b')| + 1 + |P(a,a')| \Leftrightarrow |P(a',a)| < 2.$$

If $|P(a',a)| = 0$, then $a = a'$ and $b'$ must be a vertex in $M$. This is a contradiction, as the crossing occurs between two $M$-edges.

If $|P(a,a')| = 1$, then $a' \in M$. This means $a'a$ is an $M$-edge. Now we define another walk $Q(b,d) = P(b,b') \circ (b'a') \circ (a'd)$. Since $a' \in M$, $Q(b,d)$ is an $M$-walk and, furthermore, $|Q(b,d)| = |P(b,a)|$. Thus, $Q(b,d)$ is the $M$-walk that satisfies the claim. Now, if $Q(b,d)$ is still longer than $P(b,a)$, by the same analysis we have $|P(a',a)| < 1$. This leads to a contradiction.

The next case we consider is where the edges $a'd, b'd$ are present. See Figure 3 (d). Consider an $M$-path from $b$ to $d$:

$$P'(b,d) = P(b,b') \circ (b'd) \circ (dc') \circ (c'd).$$

Notice that this is an $M$-walk. If it is longer than $|P(b,a)|$, we have

$$|P'(b,d)| > |P(b,a)| \Leftrightarrow |P(b,b')| + 3 > |P(b,b')| + 1 + |P(a,a')| \Leftrightarrow |P(a',a)| < 2.$$

For the same reason as explained earlier, $|P(a',a)| = 0$ is not possible and $|P(a',a)| = 1$ means that $a' \in M$ and we now find an $M$-walk $Q(b,d)$ by following $P(b,b')$ and then edges $b'a'$ and $a'd$. This path is one shorter than $P'(b,d)$ and this again gives a contradiction.

The other two cases, when either edges $c'b', c'a'$ or edges $c'b', b'd$ are present, are easy. Basically the edge $b'c'$ provides an $M$-walk from $b$ to $d$ which is not longer than $P(b,a)$. □

**Remark B.5.** If we apply the same proof to Type-2 $M$-walks, Claim B.4 remains true. However, what we need is a slightly different version: Either there is a Type-2 $M$-walk $P'(a,c)$ whose hop length is at most $|P(a,b)|$ or there is a Type-2 $M$-walk $P'(d,b)$ whose hop length is at most $|P(d,c)|$. The proof does not extend to show this version. On the other hand, if we fix a radius $r$, then everything goes through; see the next section.

### B.2 Type-2 $M$-Walks

$M$-walks in this section are referred to Type-2 $M$-walks.

**Proof (Proof of Lemma B.2):** We follow the same setup in the proof of Lemma B.1. Assume that there are two (Type 2) $M$-walks $P(a,b)$ *from* $a$ *to* $b$ and $P(d,c)$ *from* $d$ *to* $c$. (We switch the roles of $a$ and $b$, and of $c$ and $d$, so that we can reuse Figure 3.) The following claim implies the lemma:

**Claim B.6.** *Either there is an $M$-walk $P'(a,c)$ whose hop length is at most $|P(a,b)|$ or there is an $M$-walk $P'(d,b)$ whose hop length is at most $|P(d,c)|$.*

Observe that $|P(a,b)|$ and $|P(d,c)|$ have length exactly $r$ each since they are from balls of radius exactly $r$. Therefore, $|P(a,b)| = |P(d,c)|$, and hence Claim B.6 follows directly from Claim B.4. □

## C Geometric Data Structures

In this section, we describe how to solve the interval searching problem (Problem 2.11), the main geometric data structure problem used by our diameter algorithms and distance oracles, for different types of geometric objects. In Appendix C.1, we first describe how to reduce the interval cover to the rainbow colored intersection searching (Problem 2.13) and then describe how to reduce the interval searching problem to the interval cover problem (Problem 1.3), though with some loss of efficiency. For squares, we solve the rainbow colored intersection searching problem in Appendix C.2. For unit disks of a fixed modulo class and for unit disks, we solve the interval cover problem directly (without going through rainbow colored searching), and thus more efficiently, in Appendix C.3 and Appendix C.4 respectively, using an interesting recursive approach.

### C.1 Reductions Between Data Structure Problems

In this subsection, we provide the reductions between the data structure problems in Section 2.4, and in particular, proving Lemma 2.12 and Lemma 2.14.

**Interval cover to rainbow colored intersection searching.** We reduce the interval cover problem (Problem 1.3) to the rainbow colored intersection searching problem (Problem 2.13).

**Lemma 2.14.** *If we can construct in $\widetilde{O}(|\mathcal{O}_{RC}|)$ time a data structure $\mathcal{D}_{RC}$ with $\tilde{O}(1)$ query time for solving Problem 2.13, then for any parameter $b \in [1,n]$, we can construct a data structure $\mathcal{D}_{IC}$ for solving Problem 1.3 that has total run time $\widetilde{O}(N_{IC} \cdot b + L_{IC}/b)$.*

**Proof:** Consider an instance of Problem 1.3. Divide the range $[1,n]$ into $n/b$ blocks of length $b$, denoted by intervals $B_1, \ldots, B_{n/b}$, with $[1,n] = \bigcup_{k=1}^{n/b} B_k$. Denote by $\mathcal{S}$ the set of all intervals associated with objects in $\mathcal{O}$ and $\mathcal{S}(q)$ the set of intervals associated with objects intersecting $q$. Consider the query interval $I$. It intersects with a set of blocks $B_i, \ldots, B_j$ such that $I$ overlaps with at most two of the blocks partially, namely, the two blocks at the end ($B_i$ or $B_j$), and fully contains all the middle chunks $B_{i+1}, \ldots, B_{j-1}$. To verify if the union of the intervals of $\mathcal{S}(q)$ covers the query interval $I$, we need to check for each of the blocks, $B_k$, $i \le k \le j$, if $B_k \cap I$ is covered by the union of the intervals of $\mathcal{S}(q)$, limited within block $B_k$. If for each $B_k$ the answer is true, we answer Yes. Otherwise, we answer No. In the following we focus on answering the coverage query for a fixed block $B$ and check if the union of the intervals $\{I_s \cap B \mid I_s \in \mathcal{S}(q)\}$ covers $I \cap B$.

Now fix a block $B$. Take $I' = I \cap B$. Similarly, for each object $s$ we restrict the interval $I_s$ within $B$ and take $I'_s = I_s \cap B$ and take $\mathcal{S}' = \{I'_s \mid I_s \in \mathcal{S}\}$. Each interval $I_s$ fully covers a set of middle chunks and only partially covers at most two extreme blocks at the end of $I_s$. Thus we can write $\mathcal{S}' = \mathcal{S}'_1 \cup \mathcal{S}'_2$, with the first category $\mathcal{S}'_1$ containing the intervals $I'_s = B$ (i.e., $I_s$ fully covers $B$) and the second category $\mathcal{S}'_2$ containing the intervals $I'_s \neq B$ (i.e., $I_s$ partially covers $B$). We perform two queries for $I'$ against $\mathcal{S}'_1$ and $\mathcal{S}'_2$ respectively.

For $\mathcal{S}'_1$, all the intervals are given the same color and we just check if at least one of them is associated with an object intersecting $q$. We solve this problem by issuing RAINBOWCOVER?$(q)$ against the objects whose intervals appear in $\mathcal{S}'_1$. If this rainbow query returns a positive answer, $I'$ is covered and we are done. Otherwise, we check for $I'$ against $\mathcal{S}'_2$. This query is more complicated since the intervals $I'_s \in \mathcal{S}'_2$ only partially cover $B$. We give each of the elements in $B$ a unique color. There are at most $b$ colors. Also, for each object $s$ with $I'_s \in \mathcal{S}'_2$, we make a colored copy of the object $s$ for each element in $I'_s$; the color of the copy is equal to the color of the corresponding element. Now we discuss the case when $I' = B$ and when $I' \subset B$ separately. When $I' = B$, i.e., $B$ is an 'internal' block, we build a rainbow colored query structure for all color/elements in $B$ and issue a query RAINBOWCOVER?$(q)$ to see if all colors show up. If the query returns no, we return negative to the interval cover query. In the case when $I'$ is a boundary block ($I' \subset B$), we build the rainbow colored query structure for each color/element in $B$. To answer the query for $I'$, we issue RAINBOWCOVER?$(q)$ for each color in $I'$ against the corresponding data structure to see if this color appears among objects that intersect $q$. The total number of such queries is the number of elements in $I'$ and is at most $b$. If all the rainbow queries return True – that all colors in $I'$ appear – then all elements in $I'$ are covered by the union of intervals in $\mathcal{S}'_2$ for those objects intersecting $q$. If any rainbow query returns a no, we return a negative answer for the interval cover query.

To analyze the total running time, we need to account for the preprocessing time and the total query time for all the $n/b$ blocks. Recall that we solve the interval cover problem in an 'off-line' version and assume all input intervals and query intervals are given. $N_{IC}$ is the total number of input objects and query objects, and $L_{IC}$ is the total length of the input and query intervals. We issue a total of $O(L_{IC}/b)$ rainbow queries in the first category since for each query $(q, I)$ we only consider the blocks that overlap with $I$. For the second category we issue a total of $O(L_{IC}/b)$ rainbow queries for the blocks that are internal to the interval queries and $O(N_{IC}b)$ rainbow queries for the boundary blocks. Thus the total query time is $\widetilde{O}(N_{IC}b + L_{IC}/b)$.

For the preprocessing time, we consider the time spent to prepare for the rainbow query in the first and second category separately. For the second category, we have a total of $2N_{IC}$ intervals since each interval $I_s$ of an input object $s$ only contributes at most two boundary intervals. Each interval generates at most $b$ colored objects so we have a total of $O(N_{IC}b)$ objects, over all the $n/b$ blocks. We build the rainbow colored query data structure for each block separately. The total preprocessing time for rainbow query in the second category is thus $\widetilde{O}(N_{IC}b)$. For the rainbow query in the first category, we perform a linear scan of the blocks and only update the rainbow query data structure $\mathcal{D}_{RC}$ when needed – an input object appears (starts to fully cover a new block) or disappears (stops covering the current block). Each input object only triggers two updates. In fact, for each update, we simply rebuild the rainbow query data structure from scratch. For each input interval $I_s$, the amortized run time attributed to $I_s$ in these preprocessing and rebuilding efforts is $\widetilde{O}(|I_s|/b)$ and therefore the total running time remains $\widetilde{O}(L_{IC}/b)$, where $L_{IC}$ is the total length of the input and query intervals. Therefore, the total run time is bounded by $\widetilde{O}(N_{IC}b + L_{IC}/b)$. This finishes the proof. □

We also need a data structure that can answer interval avoidance queries. Specifically,

**Problem C.1 (Interval Avoidance Problem).** *Given a set of $N$ objects $\mathcal{O}$ and each object $o \in \mathcal{O}$ is associated with an interval $I_o \subseteq [1 : n]$. Design a data structure to answer the following query:*

- AVOIDS?$(q, I)$: *Given a query object $q$ and a query interval $I \subseteq [1:n]$, decide whether the union of intervals associated with the objects intersecting*[17] *$q$ in $\mathcal{O}$ is disjoint from the interval $I$.*

The interval avoidance problem is easier than the interval cover problem, as it is decomposible – we can partition the input objects into two sets and check the query $(q, I)$ against each set for avoidance separately.

**Lemma C.2.** *If we can construct in $\widetilde{O}(|\mathcal{O}_{RC}|)$ time a data structure $\mathcal{D}_{RC}$ with $\tilde{O}(1)$ query time for solving Problem 2.13, then we can construct a data structure $\mathcal{D}_{IA}$ for solving Problem C.1 that has a preprocesing time of $\widetilde{O}(N_{IC})$ such that each interval avoidance query takes time $\widetilde{O}(1)$.*

**Proof:** An interval $I_o$ intersects $I$ if either at least one endpoint of $I_o$ is inside $I$ or one endpoint of $I$ is inside $I_o$. Therefore, to answer the interval avoidance query, we run two types of queries. In the first type we verify if $I$ includes any endpoints of intervals whose associated objects intersect $q$. If yes, we immediately return no to the interval avoidance query. If not, we proceed to the second type of queries where we check if an endpoint of $I$ stabs any intervals whose associated objects intersect $q$. The first type is a range query, and the second type is an interval stabbing query. We explain the two operations separately.

For the range query, we take the set $\mathcal{S}$ of all intervals associated with objects in $\mathcal{O}$ and build a binary tree $\mathcal{T}$ on all the $2|\mathcal{S}|$ endpoints of the intervals. Further, for each node $v$ on the tree $\mathcal{T}$ we build a rainbow colored query structure on the objects in $\mathcal{O}$ whose associated intervals have at least one endpoint in the subtree of $v$. In particular, the data structure at the root of $\mathcal{T}$ includes all objects in $\mathcal{O}$. The total preprocessing time for these query data structures is $\tilde{O}(N_{IC})$, since each object in $\mathcal{O}$ only appears in $O(\log N_{IC})$ of the rainbow colored query structures. Next we run a standard range query with $I$ on tree $\mathcal{T}$ to find a set $Q(I)$ of $O(\log|\mathcal{S}|)$ vertices of $\mathcal{T}$ such that each vertex $v \in Q(I)$ has the entire subtree fully inside $I$, but its parent does not meet this condition. We issue a query of $q$ on the rainbow colored structure at each vertex in $Q(I)$. If any query returns a positive answer (indicating intersection), then $I$ does not avoid the objects intersecting $q$. We issue at most $O(\log N_{IC})$ rainbow colored range queries with a total cost of $\tilde{O}(1)$.

For the interval stabbing query, we build an interval tree on the intervals $\mathcal{S}$. Specifically, we have a binary tree $\mathcal{Y}$ where the root $r$ is associated with value $\ell(r) = \lfloor n/2 \rfloor$ (the median of $[1, n]$) as well as a subset of intervals $S(r)$ – all the intervals in $\mathcal{S}$ that are stabbed by $\ell(r)$. Recursively, we build the left (right) subtree by using all the intervals to the left (right) of $\ell(r)$ respectively. Further, for each node $v$ in the interval tree, we build two binary trees, $\mathcal{Z}_1(v)$ on the left endpoints of the intervals in $S(v)$ (that are all smaller than or equal to $\ell(v)$), and $\mathcal{Z}_2(v)$ on the right endpoints of the intervals in $S(v)$ (that are all greater than or equal to $\ell(v)$). For each node $u$ on a tree $\mathcal{Z}_i(v)$, $i = 1, 2$, we build a rainbow colored query structure for all the intervals in the subtree of $u$. Again, these objects are given the same color.

The total preprocessing time for these query data structures is $\tilde{O}(N_{IC})$, since each interval in $\mathcal{S}$ only appears in the set $S(v)$ of one vertex $v$ on tree $\mathcal{Y}$ and then at most $O(\log N_{IC})$ vertices in the secondary level trees $\mathcal{Z}_i(v)$.

Next we take one endpoint $p$ of $I$ and issue a stabbing query on $\mathcal{Y}$. We first issue stabbing query against the root vertex $r$ of $\mathcal{Y}$ and depending on whether $p$ is less than or greater than $\ell(r)$, recursively query either the left subtree or the right subtree of $\mathcal{Y}$. We just explain how to query $p$ against a node $v$ of $\mathcal{Y}$. The total query cost is just an extra log factor more. Specifically, if $p \leq \ell(v)$, we issue a query to $\mathcal{Z}_1(v)$; if $p \geq \ell(v)$, we issue a query to $\mathcal{Z}_2(v)$. Suppose we query $p$ on $\mathcal{Z}_1(v)$. The other case is symmetric. We take all the vertices $Z_1(p)$ of $\mathcal{Z}_1(v)$: $u \in Z_1(p)$ if all vertices in the subtree of $u$ are completely to the left of $p$ but $u$'s parent fails to meet this condition. $|Z_1(p)| = O(\log N_{IC})$. Now we query $q$ again the

[17] Here we mean the objects intersect, not their associated intervals.

rainbow colored range query structure for all vertices in $Z_1(p)$. If any of these queries return a Yes, the interval avoidance query is negative. In total the query cost adds a total factor of $O(\log^2 N_{IC})$ on top of the cost of a single rainbow colored range query.

In summary, we only add extra poly-logarithmic factors on top of the rainbow colored range query structure. Thus, we can implement the interval avoidance query with a preprocesing time of $\widetilde{O}(N_{IC})$ such that each interval avoidance query takes time $\widetilde{O}(1)$. □

**Interval searching to interval cover.** We reduce the interval searching problem (Problem 2.11) to the interval cover problem (Problem 1.3) with polylogarithmic loss.

**Lemma 2.12.** *If one can construct a data structure $\mathcal{D}_{IC}$ for solving Problem 1.3 with total run time $T(N_{IC}, n, L_{IC})$ (for some polynomial function $T$), then we can construct a data structure $\mathcal{D}_{IS}$ for solving Problem 2.11 in total run time $\widetilde{O}(T(N_{IS}, n, L_{IS}))$. Furthermore, if $\mathcal{D}_{IC}$ has preprocessing time $P(N)$ and query time $Q(N)$, then $\mathcal{D}_{IS}$ has preprocessing time $\widetilde{O}(P(\tilde{N}_{IS}))$ and query time $\widetilde{O}(Q(\tilde{N}_{IS}) \cdot |\mathcal{I}_{out}(q)|)$ where $\tilde{N}_{IS} := \sum_{o \in \mathcal{O}_{IS}} |\mathcal{I}(o)|$ is the total number of input intervals and $\mathcal{I}_{out}(q)$ is the set of output intervals from the interval search query of $q$ to $\mathcal{D}_{IS}$.*

**Proof:** We take an instance of Problem 2.11. For each object $s \in \mathcal{O}_{IS}$ we duplicate it to $k$ copies if $s$ is associated with $k$ intervals. Each copy is now associated with a single interval of $\mathcal{I}_o$. This creates a total of $\tilde{N}_{IS} = \sum_{o \in \mathcal{O}_{IS}} |\mathcal{I}(o)|$ objects. Now we build a data structure $\mathcal{D}_{IC}$ to solve Problem 1.3 on this set of objects with preprocessing time $\widetilde{O}(P(\tilde{N}_{IS}))$. For each query IntervalSearch$(q)$, we recursively issue queries to $\mathcal{D}_{IC}$. Specifically, we start with $I = [1, n]$. If $I$ is completely covered by the union of the intervals associated with objects in $\mathcal{O}_{IS}$ that intersect $q$ (which is checked by a query to $\mathcal{D}_{IC}$ with $q$ and $I$), we output $I$ and we are done. Otherwise, if $I$ is completely avoided, we output $\emptyset$ and we are also done. For the other case, we will recurse. We divide $I$ into two intervals of equal length, $I_1$ and $I_2$, and issue queries $(q, I_1)$ and $(q, I_2)$ with $\mathcal{D}_{IC}$. In the end, we will output the union of all the intervals that are fully covered by the intervals associated with objects in $\mathcal{O}_{IS}$ that intersect $q$.

The running time for a query $q$ is dependent on the total number of queries issued to $\mathcal{D}_{IC}$ recursively. Notice that all query intervals are dyadic intervals. In addition, recursion stops when an interval $I$ is completely covered by the union of intervals $\mathcal{S}(q)$ or completely avoided. Thus only the dyadic intervals whose parent partially overlaps with a query output interval will ever trigger a query. The total number of such intervals is in the order of $O(|\mathcal{I}_{out}(q)| \cdot \log n)$. Recall that each query to $\mathcal{D}_{IC}$ takes time $Q(\tilde{N}_{IS})$. Summing up everything, we have the claim in the Lemma. □

## C.2 Data Structure for Square Graphs

We now solve the rainbow colored intersection searching problem (Problem 2.13) for a set of axis-parallel squares of possibly different size. We use an approach that can be commonly found in previous work on colored range searching [GJS95, GJRS18]: for each color class, we build a set of new objects, so that colored range searching reduces to standard range searching on all the new objects.

Consider the input squares as being in the plane $z = 0$ in 3D. For each square $s$ of center $(x, y, 0)$ and side length $2r$ (or $\ell_\infty$-radius $r$) consider the point $a_s = (x, y, -r) \in \mathbb{R}^3$ and the cone $C_s$ with apex $a_s$ whose intersection with the plane $z = 0$ is the square $s$. (If we imagine the $z$ axis pointing vertically up, then this cone opens upward.) See Figure 4 for an example. For a collection $S$ of squares and the corresponding cones, consider a new square $q$ with center $(x_q, y_q, 0)$ and side length $2r_q$. Notice that the normals of the planes bounding any cone $C_s$ is the intersection of four upper half-spaces, and the normals of these upper-half-spaces are $(1, 0, 1), (0, 1, 1), (0, -1, 1)$ and $(-1, 0, 1)$.

**Observation C.3.** *The square $q$ intersects $s$ if and only if $\dot{q} := (x_q, y_q, r_q) \in C_s$.*

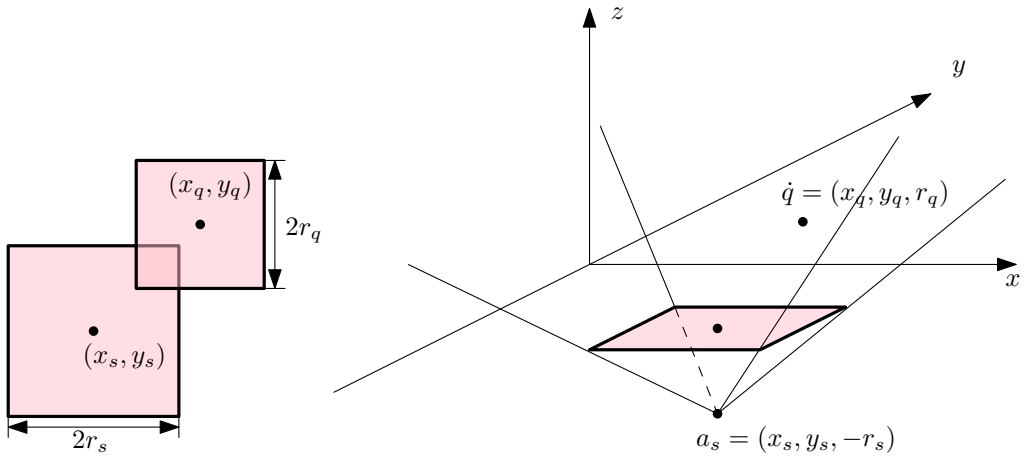


**Figure 4.** Left: a square centered at $(x_s, y_s)$ with side length $2r_s$ and a query square centered at $(x_q, y_q)$ with side length $2r_q$. Right: the cone $C_s$ and point $\dot{q} = (x_q, y_q, r_q)$.

**Proof:** For a point $(x_q, y_q, 0)$ outside $s$ we have that its $\ell_\infty$ distance to the square $\partial s$ is $\min(|x_s - x_q|, |y_s - y_q|) - r_s$. On the other hand, the vertical line through $(x_q, y_q, 0)$ intersects the cone at exactly

$$(x_q, y_q, \max(|x_s - x_q|, |y_s - y_q|) - r_q),$$

that is, the signed vertical distance from $(x_q, y_q, 0)$ to $\partial C_s$ is equal to the $\ell_\infty$ distance from $(x_q, y_q, 0)$ to $s$. In particular, the square of $\ell_\infty$ radius $r_q$ centered at $(x_q, y_q, 0)$ intersects $s$ if and only if $(x_q, y_q, r_q)$ is above $\partial C_s$, i.e., if and only if $(x_q, y_q, r_q) \in C_s$. □

As a consequence of the above observation, the square $q$ intersects some square among some set $S$ of squares if and only if $\dot{q} \in \bigcup_{s \in S} C_s$. In particular, if we have a convenient data structure to represent $\bigcup_{s \in S} C_s$, then we can quickly answer the query: given an axis-aligned square $q$, does it intersect at least one square from $S$?

**Detecting intersection with some square from $S$.** We will now work on representing $U_S := \bigcup_{s \in S} C_s$. Observe that $U_S$ is the union of translates of a fixed convex cone of constant complexity, thus it has linear union complexity. Indeed, each face $f$ of $\partial U_S$ is bounded from below, and the bottommost vertex (i.e., the vertex of minimum $z$-coordinate) on $f$ cannot be the intersection of a cone edge and a cone face nor the intersection of three cone faces, as a simple case distinction shows that all such vertices have an incident edge in $f$ where this vertex is strictly above the other endpoint. Thus the bottommost vertex of $f$ is the apex of some cone. On the other hand, each cone apex can be assigned to at most 4 faces (as there cannot be two faces $f, f'$ of $\partial U_S$ within the same cone face). We conclude that there are at most $4|S|$ faces in $\partial U_S$. By Euler's formula we have that $\partial U_S$ has complexity $O(|S|)$.

Consider the vertical projection $U_S^0$ of $\partial U_S$ into the plane $z = 0$. Notice that this is exactly an additively weighted $\ell_\infty$ Voronoi diagram (where weights are the radii of the squares). Using standard techniques [For87, Kle89] this diagram and $\partial U_S$ itself can be computed in $\widetilde{O}(|S|)$ time.

We obtain a planar subdivision where edges are either axis-aligned or they are aligned with a 45 degree rotation of the axes. We decompose this subdivision into $O(|S|)$ trapezoids with two vertical sides (or right-angle isoceles triangles with axis-aligned legs, as well as some unbounded polygons with at most two non-vertical sides) using the standard trapezoidation used for point location data structures [ST86] in $O(|S| \log |S|)$ time; let $T_S$ denote the resulting subdivision of size $O(|S|)$. More precisely, in order to get a *partition* of the plane into faces, on boundary edges with normals $(0,1), (1,0), (1,1), (-1,1)$

we require weak inequalities, while we require strong inequalities for boundary edges with normals $(0,-1),(-1,0),(-1,-1),(1,-1)$.

We project $T_S$ vertically to get a 3-dimensional subdivision $\mathcal{T}$ of $U_S$ into convex vertical slabs: here each region is a vertical slab bounded by one face of $U_S$ from below and $\partial f \times \mathbb{R}$ on the sides, where $f$ is a face of $T_S$.

Note that the complexity of $\mathcal{T}$ is $O(|S|)$ and it was computed in $O(|S|\log|S|)$ time. Moreover, each slab $T \in \mathcal{T}$ is bounded by faces whose normals can have 12 possible directions: there are 4 possible normals for faces coming from $\partial U_S$, and $4 \cdot 2$ for the vertical faces, as each of these are parallel to one of four directions in the plane $z = 0$.

To check whether a point $\dot{q}$ lies in some region $T \in \mathcal{T}$, we need to verify if it is contained in each half-space given by $\partial T$. Each such condition is of the form $\langle \dot{q}, \nu_j \rangle \leq c_T^j$ (or $< c_T^j$) where $\nu_j$ is one of 12 possible normals and $c_T^j$ is a constant that depends only on $T$. (We define $c_T^j = \infty$ if $T$ does not have a face with normal direction $\nu_j$.) For a fixed region $T$ all of these linear conditions can be written as

$$\dot{T} := (c_T^1, c_T^2, \ldots, c_T^{12}) \in \text{ort}_q := ((-\infty, \langle \dot{q}, \nu_1 \rangle] \times (-\infty, \langle \dot{q}, \nu_2 \rangle] \times (-\infty, \langle \dot{q}, \nu_3 \rangle) \cdots \times (-\infty, \langle \dot{q}, \nu_{12} \rangle)).$$

Thus, our problem of deciding if $q$ intersects at least one square from $S$ is reduced to the following: given a query square $q$, we compute a 12-dimensional orthogonal range that contains *exactly one* point among $\{\dot{T} | T \in \mathcal{T}\}$ if and only if $q$ intersects at least one square from $S$. This problem can be solved with 12-dimensional orthogonal range searching [dBCvKO08], which requires $\widetilde{O}(|\mathcal{T}|) = \widetilde{O}(|S|)$ pre-processing time and space and $\widetilde{O}(1)$ query time, to decide if the query range $\text{ort}_q$ contains some point $\dot{T}$.

**Solving rainbow colored intersection searching.** Suppose now that we are given a set of objects $\mathcal{O}$, each associated with some color; let $\mathcal{S}$ be the partition of $\mathcal{O}$ into its color classes. For each color class $S \in \mathcal{S}$ we set up the subdivision $\mathcal{T}_S$ and compute the corresponding points $\{\dot{T} \mid T \in \mathcal{T}_S\}$. Then we set up a standard orthogonal range counting data structure on the 12-dimensional point set $T_{\mathcal{S}} := \bigcup_{S \in \mathcal{S}} \{\dot{T} \mid T \in \mathcal{T}_S\}$. This takes $\sum_{S \in \mathcal{S}} \widetilde{O}(|S|) = \widetilde{O}(|\mathcal{O}|)$ preprocessing time and space, and for any orthogonal query we can return the number of points in the range in $\widetilde{O}(1)$ time.

Given a query square $\dot{q}$ we can compute the orthant query $\text{ort}_q$ and observe that the number of points in $\text{ort}_q$ is equal to the number of classes $S \in \mathcal{S}$ such that $q$ intersects at least one square from $S$. Thus, $q$ intersects all color classes if and only if $\text{ort}_q$ contains exactly $|\mathcal{S}|$ points from $T_{\mathcal{S}}$.

### C.3 Data Structure for Unit Disks

In this subsection, we directly solve the interval cover problem for unit disks restricted to a fixed modulo class, as needed in our diameter algorithm and distance oracle for unit-disk graphs. (We do so without going through rainbow colored intersection searching, to get better time bounds.) As noted in Section 7, this problem reduces to a corresponding interval cover problem about pseudolines:

**Problem C.4.** *We are given an input set $S$ of $N$ pseudolines[18] in the plane, where each pseudoline $s \in S$ has an associated interval $I_s$. We want to build a data structure to answer the following type of queries: given a query point $q$ and interval $I$, test whether $\bigcup_{\substack{s \in S \\ s \text{ below } q}} I_s$ contains $I$.* [19]

[18] We assume $O(1)$ time oracle access to deciding if a point is above/on/below a pseudoline, as well as to find the intersection of a pair of pseudolines (or determine that no itnersection exists).

[19] One way to interpret the problem is to think of each pseudoline $s$ as being "active" for a time window $I_s$; a query is to determine whether a given point $q$ stays above the upper envelope of the active pseudolines for the entire duration of the time window $I$. We will not need this viewpoint for our algorithm.

The rest of this subsection is dedicated to showing that Problem C.4 can be constructed in $N \cdot 2^{O(\sqrt{\log N \log \alpha(N)})}$ preprocessing time and answering a query takes $2^{O(\sqrt{\log N \log \alpha(N)})}$ time, where $\alpha(\cdot)$ is the slow-growing inverse Ackermann function. This is sufficient to prove Lemma 7.3.

To appreciate the difficulty of the problem, the reader may first consider the special case when $I = \mathbb{R}$, which is already nontrivial. Our idea is to explicitly construct the region of all query points $q$ for which the answer is yes. Interestingly, we are able to prove that this region has near-linear combinatorial complexity. After constructing the region, answering queries in the case when $I = \mathbb{R}$ would become easy.

To prove this combinatorial fact and at the same time design a data structure for general $I$, we will use a divide-and-conquer strategy.

**Decomposing intervals into canonical intervals.** Assume that the endpoints of all intervals $I_s$, as well as $I$, are integers bounded by $O(N)$ (by replacing numbers by their ranks). Fix a parameter $b$. A *canonical interval* refers to an interval of the form $[j \cdot b^i, (j+1) \cdot b^i)$ for some $i$ and $j$. Any interval can be expressed as a union of $O(b \log_b N)$ canonical intervals. This is a well-known fact (e.g., in analyzing a $b$-ary range tree [AE99, dBCvKO08]). For completeness, we include a quick proof in the following paragraph:

Let $J = [x, y]$ be an original interval, and suppose that the largest canonical interval covered by $J$ has size $b^k \le N$. Remove the maximum number of such intervals. Notice that this operation removes some middle part $M$ of $J$ consisting of at most $b$ intervals of size $b^{k_1}$, and leaves an interval $J_1$ on the left of $M$ and $J_2$ to the right of $M$, both having size less than $b^k$ and one endpoint that is an integer power of $b^k$. Now if $J_1 = [x, \ell_x \cdot b^k]$, then we can shift it to the interval $J_1 = [x - \ell_x \cdot b^k, 0]$. If $(-z_k \dots z_1 z_0)_b$ is the base-$b$ representation of the left endpoint of this interval, then it naturally decomposes this interval into $\sum_i z_i \le 1 + b \log_b N$ intervals. All of these intervals can be shifted by $\ell_y \cdot b^k$ to get a decomposition of $J_1$. Similarly, we can decompose $J_2 = [\ell_y \cdot b^k, y]$ by considering the base-$b$ representation of $J_2' = [0, y - \ell_y \cdot b^k]$. The resulting representation has at most $b + 2 + 2b \log_b N = O(b \log_b N)$ intervals, and it can be found in $O(b \log_b N)$ time.

We replace each interval in the input and queries by canonical intervals. If we do this procedure for all of our $N$ intervals then We end up with $O(N \cdot b \log_b N)$ canonical input intervals. For each pseudoline $s$ whose original interval $J_s$ has been decomposed into $k_s$ canonical intervals, we will have $k_s$ copies of $s$ instead, each associated with one such canonical interval. Thus, we have $N' = O(N b \log_b N)$ pseudolines, each associated with a single canonical interval. With slight abuse of notation, we will keep using $S$ for this set of pseudolines (where a single pseudoline may appear several times as long as their associated canonical intervals are different). Similarly, when a query interval is decomposed into canonical intervals, the query cost goes up by at most an $O(b \log_b N)$ factor.

**Preprocessing.** Let $\mathrm{LE}(X)$ and $\mathrm{UE}(X)$ denote the *lower and upper envelope* of a set $X$ of $x$-monotone pseudolines, respectively.

For each canonical interval $I$, let $S_{\subseteq I} := \{s \in S : I_s \subseteq I\}$ and $S_I := \{s \in S : I_s = I\}$. Let $\mathcal{E}_{\subseteq I}$ be the boundary of the region of all points $q \in \mathbb{R}^2$ such that

$$\bigcup_{\substack{s \in S_{\subseteq I} \\ s \text{ below } q}} I_s = I.$$

Then $\mathcal{E}_{\subseteq I}$ is an $x$-monotone chain in the arrangement of $S$—we can view this as a kind of "generalized envelope". We will show that this generalized envelope has near-linear combinatorial complexity and can be computed in near-linear time for a sufficiently large choice of $b$.

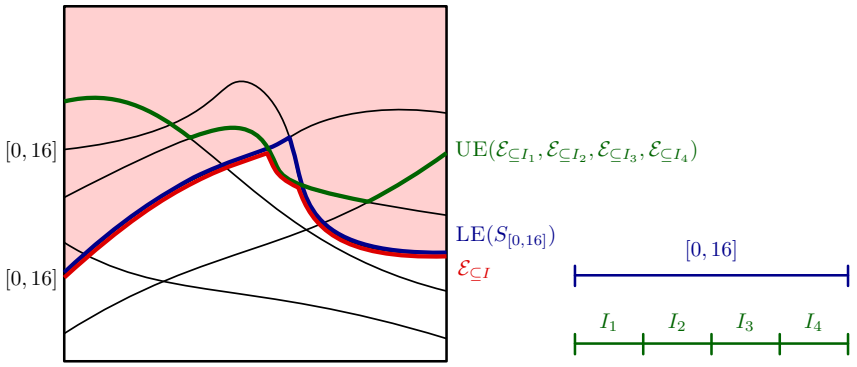


**Figure 5.** The region $\mathcal{E}_{\subseteq I}$ of points $q$ such that the intervals associated with the pseudolines under them covers $I = [0, 16]$ (red shaded region). The boundary of this region is below all pseudolines associated with $[0, 16]$ (blue envelope) and below the upper envelope of the regions associated with the canonical child intervals (green envelope).

Decompose $I$ into $b$ "child" canonical intervals $I_1, \ldots, I_b$. Note that $S_{\subseteq I} = S_I \cup S_{\subseteq I_1} \cup \cdots \cup S_{\subseteq I_b}$. Then we have the following recursive formula for the generalized envelope $\mathcal{E}_{\subseteq I}$ (see Figure 5):

$$\mathcal{E}_{\subseteq I} \;=\; \mathrm{LE}\big(\{\mathrm{LE}(S_I),\ \mathrm{UE}(\{\mathcal{E}_{\subseteq I_1}, \ldots, \mathcal{E}_{\subseteq I_b}\})\}\big).$$

Let $|\mathcal{E}_{\subseteq I}|$ denote the combinatorial complexity (number of arcs) of $\mathcal{E}_{\subseteq I}$. The upper envelope of $f$ pseudo-segments is known to have combinatorial complexity $O(f \cdot \alpha(f))$, through Davenport-Schinzel sequences [AS00, Pet15]. Thus, $\mathrm{UE}(\{\mathcal{E}_{\subseteq I_1}, \ldots, \mathcal{E}_{\subseteq I_b}\})$ has combinatorial complexity $O((|\mathcal{E}_{\subseteq I_1}| + \cdots + |\mathcal{E}_{\subseteq I_b}|) \cdot \alpha(N))$. Now, $\mathrm{LE}(S_I)$ has combinatorial complexity $O(|S_I|)$. Thus, $|\mathcal{E}_{\subseteq I}| \le O(|S_I| + (|\mathcal{E}_{\subseteq I_1}| + \cdots + |\mathcal{E}_{\subseteq I_b}|) \cdot \alpha(N))$. The maximum combinatorial complexity, $E(n)$, of $\mathcal{E}_{\subseteq I}$ among those with $|S_{\subseteq I}| = n$, satisfies the recurrence

$$E(n) \;\le \max_{n_0,\ldots,n_b:\ n_0+\cdots+n_b=n} \big(O(\alpha(N)) \cdot \big(E(n_1) + \cdots + E(n_b)\big) + O(n_0)\big),$$

which solves to $E(n) = n \cdot \alpha(N)^{O(\log_b N)}$.

For the data structure, we store $\mathcal{E}_{\subseteq I}$ as well as $\mathrm{LE}(S_I)$ for each canonical interval $I$. The preprocessing time satisfies the recurrence

$$T(n) \;\le \max_{n_0,\ldots,n_b:\ n_0+\cdots+n_b=n} \big(T(n_1) + \cdots + T(n_b) + \widetilde{O}(E(n_1) + \cdots + E(n_b) + n_0)\big),$$

which solves to $T(n) = \widetilde{O}(n \cdot \alpha(N)^{O(\log_b N)})$.

**Querying.** Given a query point $q$ and a canonical interval $I$, we check that $q$ is above $\mathcal{E}_{\subseteq I}$ by binary search in the generalized envelope, or that $q$ is above $\mathrm{LE}(S_{I'})$ for some "ancestor" canonical interval $I' \supset I$ (there are $O(\log_b N)$ such intervals $I'$). The query time is $\widetilde{O}(1)$.

**Conclusion.** After including the $O(b \log_b N)$ factor, the overall preprocessing time is $\widetilde{O}(bN \cdot \alpha(N)^{O(\log_b N)})$ and query time is $\widetilde{O}(b)$. Setting $b := 2^{\sqrt{\log N \log \alpha(N)}}$, we get $N 2^{O(\sqrt{\log N \log \alpha(N)})} \le N^{1+o(1)}$ preprocessing time and $2^{O(\sqrt{\log N \log \alpha(N)})} \le N^{o(1)}$ query time. This concludes the proof.

## C.4 Data Structure for Unit Squares

In this subsection, we directly solve the interval cover problem for unit squares. (Again, we do so without going through rainbow colored intersection searching, to get better time bounds.)

**Theorem C.5.** *There is a data structure $\mathcal{D}_{\text{square}}$ that solves the inteval cover problem (Problem 1.3) for axis-aligned unit square objects, each associated with a single interval with $N \cdot 2^{O(\sqrt{\log N})} = N^{1+o(1)}$ preprocessing time and $2^{O(\sqrt{\log N})} = N^{o(1)}$ query time.*

Observe that for the geometric intersection graph of unit side-length squares, we can replace each of the squares with squares of side-length 2 with the same center such that a pair $s, t$ of original squares intersect if and only if the center of $s$ is contained in the scaled square $t'$. As a result, the data structure problem is modified as follows: given a set $S$ of squares, where each $s \in S$ is associated with an interval $I_s$, we need a data structure to decide if the intervals of the squares containing the query *point* $q$ will cover the query interval $I$.

Instead of the above variant, we overlay a grid of side length 2 (such that no grid line is collinear with any square of $S$); let $\Gamma$ denote the set of grid cells. Notice that if $q$ is in a given grid cell $\square \in \Gamma$, then for each square $s \in S$ we have that $s \cap \square$ appears as an *orthant*, i.e., a rectangle[20] containing exactly one vertex of $\square$. Thus, in each cell $\square$ we have the following data structure problem. See Figure 6 for an example.

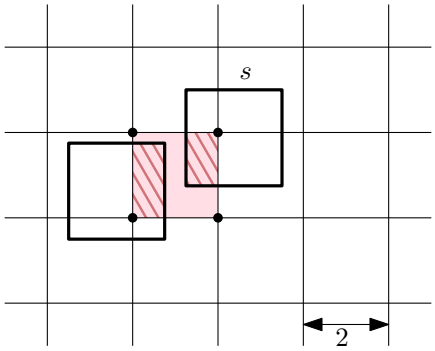


**Figure 6.** A square $s$ of side length of 2 intersecting cell $\square$ and the intersection (shaded) is an orthant containing exactly one vertex of $\square$.

**Problem C.6.** *We are given an input set $S$ of $N$ orthants in a cell $\square$ where each orthant covers exactly one vertex of $\square$ in $\mathbb{R}^2$, where each orthant $s \in S$ has an associated interval $I_s$. We want to build a data structure to answer the following type of queries: given a query point $q$ and interval $I$, test whether $\bigcup_{s \in S:\ q \in s} I_s$ contains $I$.* [21]

The rest of this subsection is dedicated to showing that Problem C.6 can be constructed in $N \cdot 2^{O(\sqrt{\log N})}$ preprocessing time and answering a query takes $2^{O(\sqrt{\log N})}$ time. This is sufficient to prove Theorem C.5, as we can use this data structure in each cell: the preprocessing time is $\sum_{\square \in \Gamma} N_\square \cdot 2^{O(\sqrt{\log N_\square})} = N 2^{O(\sqrt{\log N})}$ where $N_\square$ is the number of orthants in cell $\square$ and $\sum_\square N_\square = 4N$. To answer queries, we switch to the cell $\square_q$ containing $q$ in $O(1)$ time and answer the query using the data structure of $\square_q$ in $2^{O(\sqrt{\log N_\square})} \leq 2^{O(\sqrt{\log N})}$ time.

[20] We use the term *orthant* to distinguish these rectangles from other rectangles in the proof.

[21] Alternatively, if we think of intervals as living in a third dimension, the problem is equivalent to the following: given a set of axis-aligned boxes in $\mathbb{R}^3$ where the $xy$-projection of each box is an orthant, determine whether a query line segment parallel to the $z$-axis is completely contained in the union of the boxes. We will not need this viewpoint for our algorithm (though this type of 3-dimensional data structure problem seems interesting in its own right). As mentioned, unlike traditional range searching, this problem is not decomposable.

We will use a divide-and-conquer strategy like in Appendix C.3, but the combinatorial complexity of the regions we want may no longer have near linear complexity (because we do not restrict orthants to a fixed type), so extra ideas are needed.

Let $\text{UNION}(X)$ denote the union of a set $X$ of rectangles. As seen in Appendix C.3, we will later set some number $b$ and use canonical intervals of the form $[j \cdot b^i, (j+1) \cdot b^i)$ for some $i$ and $j$. As seen in Appendix C.3, for each orthant $s$ whose interval $I_s$ has been decomposed to $k_s$ canonical intervals, we will have $k_s$ copies of $s$ instead, each associated with one such canonical interval. Thus, we have $N' = O(N b \log_b N)$ objects, each associated with a single canonical interval. With slight abuse of notation, we will keep using $S$ for this set of objects. For an interval $I$ we again denote by $S_{\subseteq I}$ and $S_I$ the set of orthants whose intervals are subsets of $I$ or equal to $I$, respectively.

**Preprocessing.**

Let $\mathcal{Z}_{\subseteq I}$ be the region of all points $q \in \mathbb{R}^2$ such that $\bigcup_{s \in S_{\subseteq I}:\ q \in s} I_s \neq I$.

Unfortunately, the combinatorial complexity of $\mathcal{Z}_{\subseteq I}$ may be quadratic. Instead, we will maintain a set of rectangles $Z_{\subseteq I}$ with $\text{UNION}(Z_{\subseteq I}) = \mathcal{Z}_{\subseteq I}$. In other words, instead of maintaining the region $\mathcal{Z}_{\subseteq I}$ explicitly, we implicitly represent $\mathcal{Z}_{\subseteq I}$ as a union of (possibly overlapping) rectangles. We will show that a near linear number of rectangles suffices (for a sufficiently large $b$).

With this representation scheme, we can union two regions trivially. However, intersection is a trickier operation. In the lemma below, we show how to perform intersection with $\text{UNION}(S)^c$ for a set $S$ of orthants, which is sufficient for our purposes. Here, for a region $U$, we let $U^c := \square \setminus U$ denote the complement of $U$ in $\square$.

**Lemma C.7.** *Given a set $S$ of orthants and a set $Z$ of rectangles in $\square$ we can construct a set $Z'$ of $O(|S| + |Z|)$ rectangles in $\widetilde{O}(|S| + |Z|)$ time, such that $\text{UNION}(S)^c \cap \text{UNION}(Z) = \text{UNION}(Z')$.*

**Proof:** A *tallest-edge data structure* solves the following problem. We are given a set $Z$ of axis-aligned rectangles in the plane. Then, given a query segment $e$, we want to find the rectangle $z^* \in Z$ where the top side of $z^*$ has the maximal $y$ coordinate (i.e., $z^*$ is the tallest) among the rectangles $z \in Z$ covering $e$. We also allow queries in the other three axis directions, i.e., instead of the tallest reaching rectangle covering $e$, we also want to be able to find the leftmost, rightmost, or bottommost reaching rectangle covering $e$. Such queries can be answered using range trees in $\text{poly}(\log |Z|)$ query time and $\widetilde{O}(|Z|)$ preprocessing [dBCvKO08]. We start our construction by making a tallest-edge data structure $\mathcal{D}_{\text{tall}}$ for $Z$.

Let $S_1 \cup S_2 \cup S_3 \cup S_4$ be the partition of $S$ according to the vertex of $\square$ covered by the orthants. The *staircase* $i$ for $i = 1, 2, 3, 4$ is the polygonal path $\left(\partial \bigcup_{s \in S_i} s\right) \cap \square$.

Set $Z_0 := Z$ and $\mathcal{D}^0_{\text{tall}} := \mathcal{D}_{\text{tall}}$. For each set $S_i$ we will do the following computation in the order of their indices ($i = 1, \ldots, 4$). Suppose without loss of generality that $S_i$ covers the bottom left corner of $\square$; the other cases will be obtained from this via rotation. Observe that $\text{UNION}(S_i)^c$ is the region above a staircase. The staircase has $O(|S_i|)$ edges. Intersect the staircase with the boundaries of the rectangles of $Z$. Subdivide the edges of the staircase at those intersection points. Note that the edge of the staircase intersected by a given edge of a rectangle $z$ can be found with a simple binary search. The staircase now has $O(|S_i| + |Z_{i-1}|)$ edges, and it has been constructed in $\widetilde{O}(|S_i| + |Z_{i-1}|)$ time.

For each edge $e$ of the staircase, we query $\mathcal{D}^i_{\text{tall}}$ to find the rectangle $z_e \in Z_{i-1}$ containing $e$ with the highest top side in $\text{poly}(\log |Z_{i-1}|)$ time. Define $z'_e$ to be the rectangle with bottom side $e$ and top side touching the top side of $z_e$. Add $z_e$ to $Z_i$.

For each rectangle $z \in Z_{i-1}$, if the bottom side of $z$ intersects the staircase at a point $p_z$, define $z'$ to be the part of $z$ to the right of $p_z$. Add this rectangle $z'$ to $Z_i$. If $z$ is completely above the staircase, add $z$ to $Z_i$.

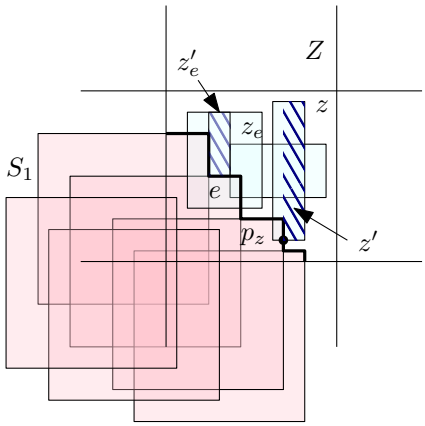


**Figure 7.** $S_1$ is the set of squares that cover the bottom left vertex of $\square$. The staircase of $S_1$ is shown in a solid polygonal path. The figure shows the rectangles $z'_e$ and $z'$ added to $Z_i$.

Then $Z_i$ has $O(|S_i|+|Z_{i-1}|)$ rectangles and satisfies the stated property. Finally, we set up a tallest-cover data structure for $Z_i$ in $\widetilde{O}(|Z_i|)$ time. The total time for step $i$ is therefore $\widetilde{O}(|S_i|+|Z_{i-1}|)$.

We can handle each of the sets $S_i$ one after another, and we set $Z' \coloneqq Z_4$. The resulting number of rectangles is

$$|Z_4| = O(|S_4|+|Z_3|) = O(|S_4|+O(|S_3|+|Z_2|)) = \cdots = O(\sum_i |S_i| + |Z_0|) = O(|S|+|Z|).$$

The total running time is $\sum_i \widetilde{O}(|S_i|+|Z_{i-1}|) = \widetilde{O}(|S|+|Z|)$. $\square$

Recall that the canonical interval $I$ can be decomposed into $b$ "child" canonical intervals $I_1, \ldots, I_b$. Suppose that there are $n_j$ orthants in $S_{\subseteq I_j}$ and $n_0$ orthants in $S_I$. We can compute $\mathcal{Z}_{\subseteq I}$ using the following recursive formula:

$$\mathcal{Z}_{\subseteq I} \;=\; \textsc{union}(S_I)^c \cap (\mathcal{Z}_{\subseteq I_1} \cup \cdots \cup \mathcal{Z}_{\subseteq I_b}).$$

We can apply the lemma to find a set $Z_{\subseteq I}$ of $O(|S_I|+|Z_{\subseteq I_1} \cup \cdots \cup Z_{\subseteq I_b}|)$ rectangles with $\textsc{union}(Z_{\subseteq I}) = \mathcal{Z}_{\subseteq I}$.

The number of rectangles in $Z_{\subseteq I}$, assuming $|S_{\subseteq I}| = n$, satisfies the recurrence

$$E(n) \;\le \max_{n_0,\ldots,n_b:\ n_0+\cdots+n_b=n} \big(O(1)\cdot\big(E(n_1)+\cdots+E(n_b)\big)+O(n_0)\big),$$

which solves to $E(n) = O(n \cdot 2^{O(\log_b N)})$.

To construct the data structure $\mathcal{D}_{\text{square}}$, we store $Z_{\subseteq I}$ and $S_I$ in individual rectangle stabbing data structures [Cha86, SJ05], for each canonical interval $I$. The data structure for a given canonical interval $I$ can therefore be made in $\widetilde{O}(E(n))$ time.

Consequently, the preprocessing time satisfies the recurrence

$$T(n) \;\le \max_{n_0,\ldots,n_b:\ n_0+\cdots+n_b=n} \big(T(n_1)+\cdots+T(n_b)+\widetilde{O}(E(n_1)+\cdots+E(n_b)+n_0)\big),$$

which solves to $T(n) = \widetilde{O}(n \cdot 2^{O(\log_b N)})$.

**Querying.** Given a query point $q$ and a canonical interval $I$, we check that $q$ is not stabbing any rectangle in $Z_{\subseteq I}$, or that $q$ stabs some orthant in $S_{I'}$ for some "ancestor" canonical interval $I' \supset I$. Since there are $O(\log_b N) = O(\log N)$ ancestor canonical intervals, the query time is $\widetilde{O}(1)$.

To answer the query about the original interval $J$, we make individual queries on each of the $O(b \log_b N)$ canonical intervals in its decomposition, and answer "yes" if and only if each canonical interval was covered.

**Conclusion.** After including the $O(b \log_b N)$ factor, the overall preprocessing time is $\widetilde{O}(bN \cdot 2^{O(\log_b N)})$ and the query time is $\widetilde{O}(b)$. Setting $b = 2^{\sqrt{\log N}}$, we get $N2^{O(\sqrt{\log N})} \leq N^{1+o(1)}$ preprocessing time and $2^{O(\sqrt{\log N})} \leq N^{o(1)}$ query time, and conclude the proof of Theorem C.5.

# D Switching Interval Representation between Different Stabbing Paths

We are given a set system $(X, \mathcal{S})$ with at most $n = |X|$ elements and $m = |\mathcal{S}|$ sets with dual shatter dimension of $(X, \mathcal{S})$ is $d$. Throughout the rest of the section we assume the existence of an *element reporting oracle* that, given $S \in \mathcal{S}$, can enumerate all elements of $S$ in $T_0(n)$ time, where $T_0(n) \geq n$.

Let $\lambda$ be an ordering of $X$. We say that a set $S$ *crosses* a pair $(x, y)$ if $x \in S$ and $y \notin S$, or vice versa. The number of consecutive pairs in $\lambda$ crossed by $S$ is at most twice the size $|\mathsf{Rep}_\lambda(S)|$. For any collection $\mathcal{R}$, define the *equivalence relation* $\equiv_{\mathcal{R}}$ over $X$, where $x \equiv_{\mathcal{R}} y$ if and only if no set in $\mathcal{R}$ crosses $(x, y)$. (In other words, $\{S \in \mathcal{R} : x \in S\} = \{S \in \mathcal{R} : y \in S\}$.) Then $\equiv_{\mathcal{R}}$ has $O(|\mathcal{R}|^d)$ equivalence classes since the dual shatter dimension is at most $d$. For every $x$ and $y$ in $X$, the *crossing number* $c_{\mathcal{S}}(x, y)$ is the number of sets in $\mathcal{S}$ crossing $(x, y)$. (Notice that $c_{\mathcal{S}}(\cdot, \cdot)$ forms a pseudometric.)

For the purpose of the remaining section, we will fix a *$\rho$-sampling* $\mathcal{R}$ of $\mathcal{S}$, where each set in $\mathcal{S}$ chosen with probability $\rho/m$. (Later on we will restrict $\mathcal{R}$ to subcollection $\mathcal{S}'$ of $\mathcal{S}$ and obtain $\mathcal{R}'$; we can still think of $\mathcal{R}'$ as obtained from $\mathcal{S}'$ by sampling each element with probability $\rho/m$, even though we do not explicit sample from $\mathcal{S}'$. Notice that the parameter $m$ does *not* change even if $\mathcal{S}'$ gets smaller.)

Our first goal is to prove that any $\rho$-sampling of $\mathcal{S}$ has low crossing number and thus can be used to construct a stabbing path for $(X, \mathcal{S})$.

**Lemma D.1.** *Let $\mathcal{R}$ be $\rho$-sampling of $\mathcal{S}$. Then for every $x, y \in X$ with $x \equiv_{\mathcal{R}} y$, crossing number $c_{\mathcal{S}}(x, y)$ is at most $O((m/\rho) \log n)$ with high probability.*

**Proof:** This follows by a standard hitting set argument. Consider any two elements $x, y \in X$ with crossing number $c_{\mathcal{S}}(x, y)$. By standard Chernoff bounds, if $c_{\mathcal{S}}(x, y) = \Omega((m/\rho) \log n)$, we would have sampled one of the sets in $\mathcal{R}$ that cross $(x, y)$ with high probability, i.e., probability $1 - 1/n^c$ for a large constant $c$, but $x \equiv_{\mathcal{R}} y$ which is a contradiction. The conclusion follows after taking a union bound over the $n^2$ pairs of elements. □

Let $\mathcal{S}'$ be an arbitrary subcollection of $\mathcal{S}$. Denote the restriction of the fixed $\rho$-sampling $\mathcal{R}$ of $\mathcal{S}$ in $\mathcal{S}'$ as $\mathcal{R}'$; in notation, $\mathcal{R}' := \mathcal{S}' \cap \mathcal{R}$. Notice that $\mathcal{R}'$ is also a $\rho$-sampling. Given any set system $(X, \mathcal{S})$, a stabbing path $\lambda$ of $(X, \mathcal{S})$ is *$\mathcal{R}'$-respecting* if each equivalence class of $\equiv_{\mathcal{R}'}$ appears contiguously in $\lambda$ for the restriction $\mathcal{R}'$. (The equivalent classes of $\equiv_{\mathcal{R}'}$ is with respect to the restriction $\mathcal{R}'$, not $\mathcal{R}$.) The above proof can be adapted so that the resulting stabbing path is $\mathcal{R}$-respecting (by choosing $\mathcal{S}' = \mathcal{S}$):

**Lemma 7.4.** *Assume the existence of an element reporting oracle that, given $S \in \mathcal{S}$, can enumerate all elements of $S$ in $T_0(n)$ time. Consider a fixed $\rho$-sampling $\mathcal{R}$ of $\mathcal{S}$. We can compute the equivalence classes of $\equiv_{\mathcal{R}}$ and construct an $\mathcal{R}$-respecting stabbing path $\lambda$ of $(X, \mathcal{S})$ such that $\sum_{S \in \mathcal{S}} |\mathsf{Rep}_\lambda(S)| = \widetilde{O}(mn/\rho + m\rho^{d-1})$ in $\widetilde{O}(T_0(n) \cdot \rho)$ time with high probability. In other words, one can compute a sampled $\rho$-stabbing path $\lambda$ of $(X, \mathcal{S})$ and the equivalence classes of $\equiv_{\mathcal{R}}$ as byproducts.*

**Proof:** Let $\mathcal{R}$ be a $\rho$-sampling of $\mathcal{S}$; then $|\mathcal{R}| = \widetilde{O}(\rho)$ with high probability. We first enumerate the elements in all $R \in \mathcal{R}$ in $\widetilde{O}(T_0(n) \cdot \rho)$ time, and compute the $O(\rho^d)$ equivalence classes of $\equiv_R$ in $\widetilde{O}(n\rho)$ time. (Each class will appear contiguously in the stabbing path $\lambda$ to be constructed.) Within each equivalence class $C_i$, we order its elements $x_1^{(C_i)}, \ldots, x_{|C_i|}^{(C_i)}$ arbitrarily. We recursively compute an ordering

of $\left\{x_1^{(C_i)} : i \in [1:\rho^d]\right\}$ by invoking the main statement of the lemma itself (which is inductively $\mathcal{R}$-respecting), with run time $\widetilde{O}(T_0(n)\cdot(\rho^d)^{1/d}) = \widetilde{O}(T_0(n)\cdot\rho)$. We then order the classes $C_i$ (as intervals of elements) according to the order of $\left\{x_1^{(C_i)} : i \in [1:\rho^d]\right\}$.

Since $2\cdot|\mathsf{Rep}_\lambda(S)|$ is equal to the number of consecutive pairs in $\lambda$ crossed by $S$, with high probability

$$\begin{aligned}
& \sum_{S\in\mathcal{S}} 2\cdot|\mathsf{Rep}_\lambda(S)| \\
= & \sum_{C_i}\left(c_{\mathcal{S}}(x_1^{(C_i)}, x_2^{(C_i)}) + \cdots + c_{\mathcal{S}}(x_{|C_i|}^{(C_i)}, x_1^{(C_{i+1})})\right) \\
\le & \sum_{C_i}\left(2c_{\mathcal{S}}(x_1^{(C_i)}, x_2^{(C_i)}) + \cdots + c_{\mathcal{S}}(x_1^{(C_i)}, x_1^{(C_{i+1})})\right) \\
\le & \widetilde{O}(mn/\rho) + \sum_{C_i} c_{\mathcal{S}}(x_1^{(C_i)}, x_1^{(C_{i+1})}),
\end{aligned}$$

where the first inequality follows from applying the triangle inequality of $c_{\mathcal{S}}(\cdot,\cdot)$ (because $c_{\mathcal{S}}(\cdot,\cdot)$ forms a pseudometric) on the last term $c_{\mathcal{S}}(x_{|C_i|}^{(C_i)}, x_1^{(C_{i+1})})$, and the second inequality is from Lemma D.1. Since we recurse on the first element of every equivalence class, by recursion we have

$$\sum_{S\in\mathcal{S}}|\mathsf{Rep}_\lambda(S)| \le \widetilde{O}(mn/\rho + m(\rho^d)^{1-1/d}) = \widetilde{O}(mn/\rho + m\rho^{d-1}).$$

The ordering is clearly $\mathcal{R}$-respecting. The total running time is $\widetilde{O}(T_0(n)\cdot\rho)$. □

**Lemma D.2.** *Consider a fixed $\rho$-sampling $\mathcal{R}$ of $\mathcal{S}$. We are given two $\mathcal{R}$-respecting stabbing paths $\lambda$ and $\lambda'$ of $(X,\mathcal{S})$ (along with the equivalence classes of $\equiv_{\mathcal{R}}$). Let $\mathcal{T}$ be an arbitrary subcollection of $\mathcal{S}$. Given $\mathsf{Rep}_\lambda(S)$ for all $S\in\mathcal{T}$, we can compute $\mathsf{Rep}_{\lambda'}(S)$ for all $S\in\mathcal{T}$ in $\widetilde{O}(mn/\rho + m\rho^d)$ total time with high probability.*

**Proof:** Given $S\in\mathcal{T}$ and an equivalent class $C$ of $\equiv_{\mathcal{R}}$, we compute the part of $\mathsf{Rep}_{\lambda'}(S)$ within $C$ as follows. Fix one representative element $x_C\in C$.

- Case 1: $x_C\notin S$. We enumerate all $x\in C$ in $S$, by examining the union of intervals of $\mathsf{Rep}_\lambda(S)$. We then concatenate $\langle x\rangle$ (singletons) over all such $x$ in the order determined by $\lambda'$.
- Case 2: $x_C\in S$. We enumerate all $x\in C$ not in $S$, by examining the complement of the union of intervals of $\mathsf{Rep}_\lambda(S)$. We then concatenate $\langle x\rangle$ (singletons) over all such $x$ in the order determined by $\lambda'$, and take the complement of the resulting union of intervals.

In both cases, the run time is linear in the number of $x\in C$ such that $S$ crosses $(x_C,x)$. So, the total run time over all $S\in\mathcal{T}$ is upper-bounded by $\sum_C\sum_{x\in C} c_{\mathcal{S}}(x_C,x) = \widetilde{O}(mn/\rho)$ with high probability by Lemma D.1.

Finally, we concatenate the different parts of $\mathsf{Rep}_{\lambda'}(S)$ over all the classes, in the order determined by $\lambda'$. This takes additional $O(m\rho^d)$ total time. □

**Lemma 7.5.** *[Conversion of interval representations.] Let $(X,\mathcal{S})$ be a set system with $|X|\le n$ and $|\mathcal{S}|\le m$. Let $\mathcal{S}'$ be a subcollection of $\mathcal{S}$ and $\mathcal{T}$ be a subcollection of $\mathcal{S}'$. Let $\mathcal{R}$ be the unique $\rho$-sampling of $\mathcal{S}$, and $\mathcal{R}'$ be its restriction in $\mathcal{S}'$. We are given an $\mathcal{R}$-respecting stabbing path $\lambda$ of $(X,\mathcal{S})$, and an $\mathcal{R}'$-respecting ordering $\lambda'$ of $(X,\mathcal{S}')$ (along with the equivalence classes of $\equiv_{\mathcal{R}}$ and $\equiv_{\mathcal{R}'}$).*

*(1) [Shrinking from $\mathcal{S}$ to $\mathcal{S}'$.] Given $\mathsf{Rep}_\lambda(S)$ for all $S\in\mathcal{T}$, we can compute $\mathsf{Rep}_{\lambda'}(S)$ for all $S\in\mathcal{T}$ in $\widetilde{O}(mn/\rho + m\rho^d)$ total time with high probability.*

(2) *[Expanding from $\mathcal{S}'$ to $\mathcal{S}$.] Given* $\mathsf{Rep}_{\lambda'}(S)$ *for all* $S \in \mathcal{T}$*, we can compute* $\mathsf{Rep}_{\lambda}(S)$ *for all* $S \in \mathcal{T}$ *in* $\widetilde{O}(mn/\rho + m\rho^d)$ *total time with high probability.*

**Proof:** For (1), notice that the equivalence classes for $\equiv_{\mathcal{R}}$ are refinements of the equivalence classes for $\equiv_{\mathcal{R}'}$. Let $\lambda''$ be an ordering obtained by taking $\lambda$, and re-ordering the classes for $\equiv_{\mathcal{R}}$ so that classes inside a common class of $\equiv_{\mathcal{R}'}$ appear contiguously, which takes $O(m\rho^d)$ time. This way, $\lambda''$ is both $\mathcal{R}$-respecting and $\mathcal{R}'$-respecting. Now, we can apply Lemma D.2 twice, to convert from $\mathsf{Rep}_{\lambda}(S)$ to $\mathsf{Rep}_{\lambda''}(S)$ and from $\mathsf{Rep}_{\lambda''}(S)$ to $\mathsf{Rep}_{\lambda'}(S)$ for all $S \in \mathcal{T}$. This takes $\widetilde{O}(mn/\rho + m\rho^d)$ time.

Similarly for (2), Let $\lambda''$ be an ordering obtained by taking $\lambda$, and re-ordering the classes for $\equiv_{\mathcal{R}}$ so that classes inside a common class of $\equiv_{\mathcal{R}'}$ appear contiguously. This way, $\lambda''$ is both $\mathcal{R}$-respecting and $\mathcal{R}'$-respecting. Now, we can apply Lemma D.2 twice, to convert from $\mathsf{Rep}_{\lambda'}(S)$ to $\mathsf{Rep}_{\lambda''}(S)$ and from $\mathsf{Rep}_{\lambda''}(S)$ to $\mathsf{Rep}_{\lambda}(S)$. □

# E Handling Small Pieces

## E.1 Patterns

Let $P$ be a piece in some LDD of $G$ with diameter $\Delta$. Recall that the set of boundary vertices of $P$ is denoted by $\partial P$. Fix an arbitrary sequence of vertices $\sigma_P = \langle s_1, s_2, \ldots, s_{|\partial P|}\rangle$. For each vertex $v \in V(G)$, let $d(v,P)$ denote the distance between $v$ and any vertex of $P$. We denote a *pattern* of $v$ with respect to the ordering $\sigma_P$, denoted by $\mathbf{p}_v$ to be the following $|\partial P|$ dimensional vector:

$$\mathbf{p}_v[i] = d(v,s_i) - d(v,P) \qquad \text{for every } 1 \le i \le |\partial P|.$$

We remark that instead of subtracting by an offset of $d(v,P)$, we could have subtracted by any other offset. For example, [LW24] instead use the offset of $d(v,s_1)$.

Le and Wulff-Nilsen [LW24] showed a bound on the total number of patterns with respect to $\sigma_P$ if the distance encoding VC-dimension is bounded. The proof also works for generalized distance VC-dimension.

**Lemma E.1.** *Let $P$ be a piece in a graph $G$ with general distance VC-dimension $d$ and $\sigma_P$ an arbitrary ordering on $\partial P$. Let $\mathbf{P} = \{\mathbf{p}_v \mid v \in V(G)\}$ be the set of patterns with respect to $\sigma_P$. Then $|\mathbf{P}| = O(|\partial P|^d \Delta^d)$.*

**Proof:** Consider the set system $(V_G \times \mathbb{Z}, \mathcal{GB})$ of generalized neighborhood balls, and the set system where we restrict the ground set $(\partial P \times [\Delta], \mathcal{GB})$. This restriction of the ground set does not increase the VC-dimension of the set system. There is a clear bijection between $\mathbf{p}_v \in \mathbf{P}$ and the generalized neighborhood ball: $\tilde{N}^{d(v,P)}[v] \cap (\partial P \times [\Delta]) = \{(u,r) : u \in \partial P, r \in [\Delta], d(u,v) \le d(u,P) + r\}$. So the number of patterns is bounded by the number of unique sets of $(\partial P \times [\Delta], \mathcal{GB})$. By the Sauer-Shelah Lemma (see Lemma 2.4), this is at most $O(|\partial P|^d |\Delta|^d)$. □

## E.2 Diameter and Eccentricities using Patterns

The following algorithm computes the eccentricities of all vertices in a piece $P$ of the graph $G$.

1. Compute the pairwise distance between pairs of vertices in $P$. Let $d'_v$ denote the distance to the farthest vertex from $v$ that is within $P$.
2. Compute all patterns $\mathbf{P}$ for $P$, and for each pattern $\mathbf{p} \in \mathbf{P}$ find the farthest vertex $u \in V(G)$ that attains that pattern. Let $d_{\mathbf{p}}$ be $d(u,s_0)$, the base distance for the pattern.

3. For each pattern $\mathbf{p} \in \mathbf{P}$, compute the distance $d(\mathbf{p}, v)$ from the pattern to each $v \in P$ by doing a *boundary weighted BFS*, i.e., a BFS where the boundary vertex distances are initialized to the values of the pattern $\mathbf{p}$, and for each vertex $v \in P$ compute $d_v = \max_{\mathbf{p} \in \mathbf{P}} d(\mathbf{p}, v) + d_{\mathbf{p}}$.
4. Return $\max\{d_v, d'_v\}$.

Step 1 can be implemented by running a BFS within $P$ from each vertex. By Lemma E.1 the number of patterns computed in step 2 is at most $O(|\partial P|^d \Delta^d)$, and it takes $O(n|\partial P|)$ time to consider all distances to compute the pattern. Running a BFS for each pattern in step 3 takes time $T(P)$ per pattern where $T(P)$ is the time it takes to run a boundary weighted BFS in $P$.

**Lemma 2.15.** *Let $G$ be a graph on $n$ vertices with distance encoding VC-dimension $d$. Let $P$ be a piece in $G$ with boundary $\partial P$ and diameter $\Delta$. If distances from $\partial P$ to all vertices of $G$ are known, the eccentricity of all vertices in $P$ can be computed in $O\left(n \cdot |\partial P| + (|P| + |\partial P|^d \Delta^d) \cdot T(P)\right)$ where $T(P)$ is the time it takes to run boundary weighted BFS on $P$ with weights at most $\Delta$.*

## E.3 Boundary Weighted BFS in Geometric Intersection Graphs

One approach to compute a shortest path tree in a unit disk graph of $n$ disks uses a semi-dynamic data structure, developed in [EIK01], that in $O(\log n)$ amortized time finds a disk containing a query point and deletes it from the set. Thus one can repeatedly apply the data structure to find the disks at the $(i+1)$-hop frontier in a BFS tree from the $i$th hop frontier– for each disk at $i$-hop away from the root, repeated query the center of the disk to look for disks that intersect with it until such disks are exhausted. This gives a running time of $O(n \log n)$ to compute a BFS tree, since each disk is only deleted once. The semi-dynamic data structure uses a grid of side length $1/2$. For each cell $Q$ of the grid, maintain the set of disks whose center lies in $Q$. Furthermore, maintain the upper envelope $S_1$ of the disks that intersect $Q$ with centers below the line through the lower boundary of $Q$, and similarly maintain the envelopes $S_2, S_3, S_4$ for the other three boundaries. Therefore, if a query point $q$ lies in a cell $Q$, all the disks that are centered inside $Q$ would contain $q$ and can be returned. Further, query $q$ against the upper envelope $S_1$ (check if $q$ is below $S_1$) to look for additional candidates. And repeat the same procedure for the other three envelopes. The upper envelope is maintained by a binary tree similar to a segment tree.

The boundary weighted BFS problem in a unit disk graph can be solved by a slight modification of this procedure: vertices on the boundary appear as query points when the shortest path tree has reached a sufficient depth. Therefore we have the following observation.

**Observation E.2.** *The boundary weighted BFS problem in a unit disk graph can be implemented in $O(|P| \log |P|)$ time.*

For boundary weighted BFS in the intersection graph of axis aligned squares (of varying sizes), we can use the same idea above. We need the following semi-dynamic data structure for a set of axis-parallel squares: given a query square $q$ return a square $r$ that intersects $q$, and then delete $r$. If $q$ and $r$ intersect, either some corners of $q$ is inside $r$ or some corners of $r$ are inside $q$. Thus, the above query can be implemented by running an orthogonal range query of $q$ on the set of corner points of current set of squares, as well as a point enclosure query [Cha86] (also called a rectangle stabbing query [SJ05]) of each of the corners of $q$ against the set of current squares. These queries can be answered by 2D orthogonal range trees or 2D segment trees. By using dynamic fractional cascading with deletion only, both query and deletion can be handled in $O(\log s)$ amortized time if we have $s$ squares [CG86b, CG86a]. Therefore, we have the following lemma.

**Lemma E.3.** *The boundary weighted BFS problem in the intersection graph of axis-aligned squares can be implemented in $O(|P| \log |P|)$ time.*

## E.4 Exact Distance Oracles

The lemmas in this section are implicit in the distance oracles of [LW24], but we present their proofs in full to keep our exposition self-contained.

**Lemma E.4 (Section 3.2.3 of [LW24]).** *Let $G$ be a graph on $n$ vertices with bounded generalized distance VC-dimension $d$ and $P$ be a piece in $G$ with boundary $\partial P$ and diameter $\Delta$. There exists an exact distance oracle for queries in which at least one end point lies within $P$ with $O(n + |\partial P|^d \Delta^d |P| + |P|^2)$ space and $O(1)$ query time.*

*Furthermore if distances from $\partial P$ to all vertices of $G$ are known, the distance oracle can be computed in $O(n|\partial P| + (|\partial P|^d \Delta^d + |P|) \cdot T(P))$ precomputation time, where $T(P)$ is the time it takes to run vertex weighted BFS on $P$ with weights at most $\Delta$.*

**Proof:** For each vertex $v \in P$, store the distances to all other vertices in $P$. Every other vertex of the graph $u \in G \setminus P$ stores a pointer to their respective pattern $\mathbf{p}_u$, and the distance $d(u, P)$. Also store the distance $d(\mathbf{p}, v)$ for each pattern $\mathbf{p} \in \mathbf{P}$ to each $v \in P$.

To handle a query between two vertices of $P$, we can look up the distance between the vertices in constant time. For one vertex $v \in P$, and another vertex $u \in G \setminus P$, we know that:

$$d(u, v) = d(u, P) + d(\mathbf{p}_u, v).$$

and we can look up $d(u, P)$, $\mathbf{p}_u$, and $d(\mathbf{p}_u, v)$ in constant time.

The total space needed for the oracle is $O(|P|^2)$ for the distances between pairs of vertices in $P$, $O(n)$ for the pointers from vertices $u \in G \setminus P$ to their respective patterns, and $O(|\partial P|^d \Delta^d |P|)$ to store the pattern to $P$ distances.

The precomputation time is the same as in Lemma 2.15 for eccentricities. □

**Lemma 2.16 (Section 4.3.1 of [LW24]).** *Let $G = (V_G, E_G)$ be a graph with bounded distance VC-dimension $d$, and $P$ be an induced subgraph of $G$ with boundary $\partial P$ and diameter $\Delta$. There exists a distance oracle that answers distances from any vertex $s \in P$ and any vertex $t \in V_G$ with $O(n \cdot |\partial P| + |P|^d)$ space and $O(\log |\partial P|)$ query time.*

*Furthermore, if $G$ also has bounded generalized distance VC-dimension $d$ and distances from $\partial P$ to all vertices of $G$, the distance oracle can be computed in $O\left(n \cdot |\partial P| + (|\partial P|^d \Delta^d + |P|) \cdot T(P)\right)$ time, where $T(P)$ is the time it takes to run vertex weighted BFS on $P$ with weights at most $\Delta$.*

**Proof:** Store the distances between pairs of vertices in $P$. For every other vertex $u \in G \setminus P$, consider the sequence of balls $B(u, r_1), \ldots, B(u, r_k)$ such that $B(u, r_1)$ is the smallest ball that contains at least one vertex of $\partial P$, and $B(u, r_i)$ is the smallest ball containing at least one vertex of $\partial P \setminus B(u, r_i)$ (note that $k \leq |\partial P|$). Store a pointer to each of these balls, and the set of vertices $Y_i = B(u, r_i) \cap P$ (and $Y_0 = \emptyset$) in a data structure that allows for $O(1)$ time membership lookup. For each relevant $Y_i$, store the distance $d(Y_i, v) := \min_{s \in Y_i} d(s, v)$.

If two vertices $u$ and $v$ are within $P$, we can look up their distance in $O(1)$ time. Otherwise, if $v \in P$ and $u \in G \setminus P$, then we can binary search over $Y_0, Y_1 \ldots, Y_k$ to find the first $Y_i$ where $v \notin Y_i$ and $v \in Y_{i+1}$ in $O(\log k) = O(\log |\partial P|)$ time. Then, we can look up the distance $d(Y_i, v)$ in constant time and return the distance:

$$d(u, v) = d(u, r_i) + d(Y_i, v).$$

The space required to store distances between pairs of vertices in $P$ is at most $O(|P|^2)$. The space required is $O(n|\partial P|)$ to store the pointers between $u \in G \setminus P$ and their respective $Y_0, Y_1, \ldots, Y_k$ since $k \leq |\partial P|$. The total number of balls is at most $O(|P|^d)$ by Lemma 2.4 (Sauer's lemma).

To compute this distance oracle, we need to compute $Y_1, \ldots, Y_k$ for each vertex $u \in G \setminus P$. To do so, we can cluster these vertices into vertices with the same pattern $\mathbf{p}_u$, and consider $Y_1, \ldots, Y_k$ with respect to each pattern. This can be done as the BFS to compute $d(\mathbf{p}_u, v)$ for every vertex $v \in P$ also implicitly computes the balls $Y_1, \ldots Y_k$, as well as the distances $d(Y_i, v)$. To compute a pointer from $u$ to $Y_i$, we can look up the balls we computed by storing all $Y_i$s in a data structure that supports $O(1)$ lookup for sets (e.g. a hashing based data structure). The precomputation time analysis is the same as in Lemma 2.15 for eccentricities. □