Dynamic Unit-Disk Range Reporting

Haitao Wang
Kahlert School of Computing, University of Utah, Salt Lake City, UT, USA

Yiming Zhao
Department of Computer Sciences, Metropolitan State University of Denver, CO, USA

Abstract

For a set P of n points in the plane and a value $r > 0$, the *unit-disk range reporting* problem is to construct a data structure so that given any query disk of radius r, all points of P in the disk can be reported efficiently. We consider the dynamic version of the problem where point insertions and deletions of P are allowed. The previous best method provides a data structure of $O(n \log n)$ space that supports $O(\log^{3+\epsilon} n)$ amortized insertion time, $O(\log^{5+\epsilon} n)$ amortized deletion time, and $O(\log^2 n / \log \log n + k)$ query time, where ϵ is an arbitrarily small positive constant and k is the output size. In this paper, we improve the query time to $O(\log n + k)$ while keeping other complexities the same as before. A key ingredient of our approach is a shallow cutting algorithm for circular arcs, which may be interesting in its own right. A related problem that can also be solved by our techniques is the dynamic unit-disk range emptiness queries: Given a query unit disk, we wish to determine whether the disk contains a point of P. The best previous work can maintain P in a data structure of $O(n)$ space that supports $O(\log^2 n)$ amortized insertion time, $O(\log^4 n)$ amortized deletion time, and $O(\log^2 n)$ query time. Our new data structure also uses $O(n)$ space but can support each update in $O(\log^{1+\epsilon} n)$ amortized time and support each query in $O(\log n)$ time.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Design and analysis of algorithms

Keywords and phrases Unit disks, range reporting, range emptiness, alpha-hulls, dynamic data structures, shallow cuttings

Digital Object Identifier 10.4230/LIPIcs.STACS.2025.76

Related Version *Full Version*: `https://arxiv.org/pdf/2501.00120`

Funding This research was supported in part by NSF under Grant CCF-2300356.

1 Introduction

Range searching is a fundamental problem and has been studied extensively in computational geometry [2,3,26]. In this paper, we consider a dynamic range reporting problem regarding disks of fixed radius, called *unit disks*.

Given a set P of n points in the plane, the *unit-disk range reporting* problem (or UDRR for short) is to construct a data structure to report all points of P in any query unit disk. The problem is also known as the *fixed-radius neighbor problem* in the literature [4,14,16,17]. Chazelle and Edelsbrunner [17] constructed a data structure of $O(n)$ space that can answer each query in $O(\log n + k)$ time, where k is the output size; their data structure can be constructed in $O(n^2)$ time. By a standard lifting transformation [5], the problem can be reduced to the half-space range reporting queries in 3D; this reduction also works if the radius of the query disk is arbitrary. Using Afshani and Chan's 3D half-space range reporting data structure [1], one can construct a data structure of $O(n)$ space with $O(\log n + k)$ query time, while the preprocessing takes $O(n \log n)$ expected time since it invokes Ramos' algorithm [27] to construct shallow cuttings for a set of planes in 3D. Chan and Tsakalidis [13] later presented an $O(n \log n)$-time deterministic shallow cutting algorithm. Combining the framework in [1] with the shallow cutting algorithm [13], one can build a data structure of $O(n)$ space in $O(n \log n)$ deterministic time that can answer each UDRR query in $O(\log n + k)$ time.

We consider the dynamic UDRR problem in which point insertions and deletions of P are allowed. By the lifting transformation, the problem can be reduced to dynamic halfspace range reporting in 3D [7, 10, 11], which also works for query disks of arbitrary radii. Using the currently best result of dynamic halfspace range reporting [6], one can obtain a data structure of $O(n \log n)$ space that supports $O(\log^{3+\epsilon} n)$ amortized insertion time, $O(\log^{5+\epsilon} n)$ amortized deletion time, and $O(\log^2 n / \log \log n + k)$ query time, where ϵ is an arbitrarily small positive constant and k is the output size.

Our result. In this paper, we achieve the optimal $O(\log n + k)$ query time, while the space of the data structure and the update time complexities are the same as above.

A byproduct of our techniques is a static data structure of $O(n)$ space that can be built in $O(n \log n)$ time and support $O(\log n + k)$ query time. This matches the above result of [1, 13]. But our method is much simpler. Indeed, the algorithm of [1, 13] involves relatively advanced geometric techniques like computing shallow cuttings for the planes in 3D, planar graph separators, etc. Our algorithm, on the contrary, relies only on elementary techniques (the most complicated one might be a fractional cascading data structure [18, 19]). One may consider our algorithm a generalization of the classical 2D half-plane range reporting algorithm of Chazelle, Guibas, and Lee [20].

Our techniques may also be useful for solving other problems related to unit disks. In particular, we can obtain an efficient algorithm for the dynamic *unit-disk range emptiness queries*. For a dynamic set P of points in the plane, we wish to determine whether a query unit disk contains any point of P (and if so, return such a point as an "evidence"). The previous best solution is to use a dynamic nearest neighbor search data structure [11]. Specifically, we can have a data structure of $O(n)$ space that supports $O(\log^2 n)$ amortized insertion time, $O(\log^4 n)$ amortized deletion time, and $O(\log^2 n)$ query time. Using our techniques, we obtain an improved data structure of $O(n)$ space that supports both insertions and deletions in $O(\log^{1+\epsilon} n)$ amortized time and supports queries in $O(\log n)$ time.

Our approach. We first discuss our static data structure. We use a set of $O(n)$ grid cells (each of which is an axis-parallel rectangle) to capture the proximity information for the points of P, such that the distance between any two points in the same cell is at most 1. For a query unit disk D_q whose center is q, points of P in the cell C that contains q can be reported immediately. The critical part is handling other cells that contain points of $P \cap D_q$. The number of such cells is constant and each of them is separated from C (and thus from q) by an axis-parallel line. The problem thus boils down to the following subproblem: Given a set Q of points in a grid cell C' above a horizontal line ℓ, report the points of Q in any query unit disk whose center is below ℓ. A point $p \in Q$ is in D_q if and only if q lies in the unit disk D_p centered at p, or equivalently, q is above the arc of the boundary of D_p below ℓ. Let Γ denote the set of all such arcs for all points $p \in Q$. To find the points of Q in D_q, it suffices to report the arcs of Γ below q. To tackle this problem, we follow the same framework as that for the 2D half-plane range reporting algorithm [20], by constructing lower envelope layers of Γ and building a fractional cascading data structure on them [18, 19].

To make the data structure dynamic, we first derive a data structure to maintain the grid cells dynamically so that each update (point insertions/deletions) can be handled in $O(\log n)$ amortized time. This dynamic data structure could be of independent interest. Next, we develop a data structure to dynamically maintain arcs of Γ to support the arc-reporting queries (i.e., given a query point, report all arcs of Γ below it). To this end, we cannot use the fractional cascading data structure anymore because it is not amenable to dynamic changes. Instead, we adapt the techniques for dynamically maintaining a set of lines to answer line-reporting queries (i.e., given a query point, report all lines below the query point; its dual problem is the halfplane range reporting queries) [6, 9–11, 23]. To make these techniques work for the arcs of Γ in our problem, we need an efficient shallow cutting algorithm for Γ. For this, we adapt the algorithm in [13] for lines and derive an $O(|\Gamma|\log|\Gamma|)$ time shallow cutting algorithm for Γ. As shallow cuttings have many applications, our algorithm may be interesting in its own right.

Outline. The rest of the paper is organized as follows. In Section 2, we introduce notation and a *conforming coverage set* of grid cells to capture the proximity information for points of P. Section 3 discusses our dynamic UDRR data structure. A main subproblem of it is solved in Section 4. A key ingredient of our method is an efficient algorithm for computing shallow cuttings for arcs; this algorithm is presented in Section 5. Section 6 demonstrates that our techniques may be used to solve other related problems, such as dynamic unit-disk range emptiness queries. Due to the space limit, some details and proofs are omitted but can be found in the full paper.

2 Preliminaries

We define some notation that will be used throughout the paper. A *unit disk* refers to a disk of radius 1. A *unit circle* is defined similarly. Unless otherwise stated, a circular arc or an arc refers to a circular arc of radius 1. For a circular arc γ, we call the circle that contains γ the *underlying circle* of γ and call the disk whose boundary contains γ the *underlying disk*.

For any point q, let D_q denote the unit disk centered at q. For any region R and any set P of points in the plane, let $P(R)$ denote the subset of points of P inside R, i.e., $P(R) = P \cap R$. For any region R in the plane, we use ∂R to denote its boundary, e.g., if R is a disk, then ∂R is its bounding circle.

Unless otherwise stated, ϵ refers to an arbitrarily small positive constant. Depending on the context, we often use k to denote the output size of a reporting query. For any point p in the plane, we use $x(p)$ and $y(p)$ to denote the x and y-coordinates of p, respectively.

2.1 Conforming coverage of P

Let P be a set of n points in the plane. We wish to have a data structure for P to answer the following *unit-disk range reporting queries*: Given a query unit disk D, report $P(D)$, i.e., the points of P in D.

As discussed in Section 1, our method (for both static and dynamic problems) relies on a set of grid cells to capture the proximity information for the points of P. The technique of using grids has been widely used in various algorithms for solving problems in unit-disk graphs [12, 28–32]. However, the difference here is that we need to handle updates to P and therefore our grid cells will be dynamically changed as well. To resolve the issue, our definition of grid cells is slightly different from the previous work. Specifically, we define a *conforming coverage* set of cells for P in the following.

▶ **Definition 1** (Conforming Coverage). *A set $\mathcal{C}$ of cells in the plane is called a* conforming coverage *for P if the following conditions hold.*

1. *Each cell of $\mathcal{C}$ is an axis-parallel rectangle of side lengths at most $1/2$. This implies that the distance of every two points in each cell is at most 1.*
2. *The union of all cells of $\mathcal{C}$ covers all the points of P.*

3. *Every two cells are separated by an axis-parallel line.*
4. *Each cell $C \in \mathcal{C}$ is associated with a subset $N(C) \subseteq \mathcal{C}$ of $O(1)$ cells (called* neighboring cells *of C) such that for any point $q \in C$, $P(D_q) \subseteq \bigcup_{C' \in N(C)} P(C')$.*
5. *For any point q, if q is not in any cell of $\mathcal{C}$, then $P \cap D_q = \emptyset$.*

To solve the static problem, after computing a conforming coverage set of cells for P, we never need to change it in the future. As such, the following lemma from [28] suffices.

▶ **Lemma 2** ([28]).
1. *A conforming coverage set $\mathcal{C}$ of size $O(n)$, along with $P(C)$ and $N(C)$ for all cells $C \in \mathcal{C}$, can be computed in $O(n \log n)$ time and $O(n)$ space.*
2. *With $O(n \log n)$ time and $O(n)$ space preprocessing, given any point q, we can do the following in $O(\log n)$ time: Determine whether q is in a cell C of $\mathcal{C}$, and if so, return C and $N(C)$.*

However, for the dynamic problem, due to the updates of P, the conforming coverage set also needs to be maintained dynamically. For this, we have the following lemma.

▶ **Lemma 3.** *A conforming coverage set $\mathcal{C}$ of $O(n)$ cells for P can be maintained in $O(n)$ space (where n is the size of the current set P) such that each point insertion of P can be handled in $O(\log n)$ worst-case time, each point deletion can be handled in $O(\log n)$ amortized time, and the following query can be answered in $O(\log n)$ time: Given a query point q, determine whether q is in a cell C of $\mathcal{C}$, and if so, return C and $N(C)$.*

The proof of Lemma 3, which is lengthy and technical, is in the full paper. Roughly speaking, if a point p is inserted to P, then at most $O(1)$ cells will be added to $\mathcal{C}$ and p will eventually be inserted into $P(C)$ for the cell $C \in \mathcal{C}$ containing p. If a point p is deleted from P, the deletion boils down to the deletion of p from $P(C)$ for the cell $C \in \mathcal{C}$ containing p. We do not remove cells from $\mathcal{C}$. Instead, we reconstruct the entire data structure after $n/2$ deletions; this guarantees that the size of $\mathcal{C}$ is always $O(n)$. See the full paper for the details.

3 Dynamic range reporting

Let P be a set of points in the plane. We wish to maintain a data structure for P to answer unit-disk range reporting queries subject to point insertions and deletions of P. Let n denote the size of the current set P.

Using Lemma 3, we maintain a conforming coverage set $\mathcal{C}$ of $O(n)$ cells for P. To insert a point p to P, our insertion algorithm for Lemma 3 boils down to inserting p to $P(C)$ for a cell $C \in \mathcal{C}$ that contains p. To delete a point p from P, our deletion algorithm for Lemma 3 boils down to deleting p from $P(C)$ for a cell $C \in \mathcal{C}$ that contains p.

Consider a query unit disk D_q whose center is q. If q is not in a cell of $\mathcal{C}$, then by Definition 1(5), $P(D_q) = \emptyset$ and thus we simply return null. Otherwise, to report $P(D_q)$, it suffices to report $P(C') \cap D_q$ for all cells $C' \in N(C)$. In the case of $C' = C$, since the distance between two points in C is at most 1 by Definition 1(1), we can simply report all points of $P(C)$. If $C' \neq C$, C and C' are separated by an axis-parallel line by Definition 1(3). Without loss of generality, we assume that C and C' are separated by a horizontal line ℓ with C' above ℓ and C below ℓ. As $q \in C$, q is below ℓ, i.e., q is separated from C' by ℓ. Our goal is to report points of $P(C') \cap D_q$. Due to the point updates of $P(C')$, our problem is reduced to the following subproblem, called *dynamic line-separable UDRR problem.*

▶ **Problem 1** (Dynamic line-separable UDRR)**.** *For a set Q of m points above a horizontal line ℓ, maintain Q in a data structure to support the following operations. (1) Insertion: insert a point to Q; (2) deletion: delete a point from Q; (3) unit-disk range reporting query: given a point q below ℓ, report the points of Q in the unit disk D_q.*

We have the following Lemma 4 for the dynamic line-separable UDRR problem.

▶ **Lemma 4.** *For the dynamic line-separable UDRR problem, we can have a data structure of $O(m \log m)$ space to maintain Q to support insertions in $O(\log^{3+\epsilon} m)$ amortized time, deletions in $O(\log^{5+\epsilon} m)$ amortized time, and unit-disk range reporting queries in $O(k+\log m)$ time, where k is the output size, ϵ is an arbitrarily small positive constant, and m is the size of the current set Q.*

We will prove Lemma 4 in Section 4. With Lemmas 3 and 4, we can obtain the following main result for our original dynamic UDRR problem.

▶ **Theorem 5.** *We can maintain a set P of points in the plane in a data structure of $O(n \log n)$ space to support insertions in $O(\log^{3+\epsilon} n)$ amortized time, deletions in $O(\log^{5+\epsilon} n)$ amortized time, and unit-disk range reporting queries in $O(k + \log n)$ time, where k is the output size, ϵ is an arbitrarily small positive constant, and n is the size of the current set P.*

Proof. We build the data structure $\mathcal{D}$ in Lemma 3 to maintain a conforming coverage set $\mathcal{C}$ of $O(n)$ cells for P. For each cell $C \in \mathcal{C}$ that contains at least one point of P, we maintain a data structure $\mathcal{D}_e(C)$ for $P(C)$ with respect to the supporting line of each edge e of C. Since the space of each $\mathcal{D}_e(C)$ is $O(|P(C)| \log |P(C)|)$, each cell of $\mathcal{C}$ has four edges, and $\sum_{C\in\mathcal{C}} |P(C)| = n$, the total space of the overall data structure is $O(n \log n)$.

Insertions. To insert a point p to P, we first update $\mathcal{D}$ by Lemma 3, which takes $O(\log n)$ worst-case time. The insertion algorithm of Lemma 3 eventually inserts p to $P(C)$ for a cell $C \in \mathcal{C}$ that contains p. We insert p to $\mathcal{D}_e(C)$ for each edge e of C, which takes $O(\log^{3+\epsilon} n)$ amortized time by Lemma 4. As such, each insertion takes $O(\log^{3+\epsilon} n)$ amortized time.

Deletions. To delete a point p from P, we first update $\mathcal{D}$ by Lemma 3, which takes $O(\log n)$ amortized time. The deletion algorithm of Lemma 3 eventually deletes p from $P(C)$ for a cell $C \in \mathcal{C}$ that contains p. We delete p from $\mathcal{D}_e(C)$ for each edge e of C, which takes $O(\log^{5+\epsilon} n)$ amortized time by Lemma 4. As such, each deletion takes $O(\log^{5+\epsilon} n)$ amortized time.

Queries. Given a query unit disk D_q with center q, we first check whether q is in a cell of $\mathcal{C}$, and if so, find such a cell; this takes $O(\log n)$ time by Lemma 3. If no cell of $\mathcal{C}$ contains q, then $P \cap D_q = \emptyset$ and we simply return null. Otherwise, let C be the cell of $\mathcal{C}$ that contains q. We first report all points of $P(C)$. Next, for each $C' \in N(C)$, by Definition 1(3), C and C' are separated by an axis-parallel line ℓ. Since each edge of C and C' is axis-parallel, C' must have an edge e whose supporting line is parallel to ℓ and separates C and C'. Using $\mathcal{D}_e(C')$, we report all points of $P(C')$ inside D_q. As $|N(C)| = O(1)$, the total query time is $O(\log n + k)$ by Lemma 4. ◀

4 Proving Lemma 4: Dynamic line-separable UDRR

We now prove Lemma 4. For notational convenience, instead of m, we use n to denote the size of Q.

Consider a query unit disk D_q with center q below ℓ. The goal is to report $Q(D_q)$. Observe that a point $p \in Q$ is in D_q if and only if q is in the unit disk D_p. The portion of ∂D_p below ℓ is a circular arc, denoted by γ_p. Since p is above ℓ, γ_p is on the lower half circle of ∂D_p and thus is x-monotone. As such, p is in D_q if and only if q is above the arc γ_p. Define Γ to be the set of arcs γ_p for all points $p \in Q$. Therefore, reporting the points of Q in D_q becomes reporting the arcs of Γ that are below q, which we call *arcs reporting queries.*

In what follows, an arc of Γ always refers to the portion below ℓ of a unit circle with center above ℓ. Our problem thus becomes dynamically maintaining a set Γ of arcs to report the arcs of Γ below a query point q. The arcs reporting queries can be reduced to the following *k-lowest-arcs queries*: Given a query vertical line ℓ^* and a number $k \geq 1$, report the k lowest arcs of Γ intersecting ℓ^*. We have the following observation, which follows the proof of Chan [7] for lines.

▶ **Observation 6** ([7]). *Suppose that we can answer each k-lowest-arcs query in $O(\log n + k)$ time. Then, the arcs of Γ below a query point q can be reported in $O(\log n + k)$ time, where k is the output size.*

In light of the above observation, we now focus on the k-lowest-arcs queries. We adapt a technique for a similar problem on lines (which is the dual problem of the dynamic halfplane range reporting problem): Dynamically maintain a set of lines (subject to insertions and deletions) to report the k-lowest lines at a query vertical line. For this problem, Chan [10] gave a data structure of $O(n \log n)$ space that supports $O(\log^{6+\epsilon} n)$ amortized update time and $O(k + \log n)$ query time. De Berg and Staals [6] improved the result of [10] for dynamically maintaining a set of planes in 3D. They gave a data structure of $O(n \log n)$ space that supports $O(\log^{3+\epsilon} n)$ amortized insertion time, $O(\log^{5+\epsilon} n)$ amortized deletion time, and $O(\log^2 n / \log\log n + k)$ query time. Their approach is based on the techniques for dynamically maintaining planes for answering lowest point queries [9, 11, 23] and these techniques in turn replies on computing shallow cuttings on the planes in 3D [13]. In the following, we will extend these techniques to the arcs of Γ and prove the following result.

▶ **Lemma 7.** *For the set Γ of arcs, we can have a data structure of $O(n \log n)$ space to support insertions in $O(\log^{3+\epsilon} n)$ amortized time, deletions in $O(\log^{5+\epsilon} n)$ amortized time, and k-lowest-arcs queries in $O(k + \log n)$ time, where n is the size of the current set Γ.*

Combining Lemma 7 and Observation 6 immediately leads to Lemma 4.

In what follows, we first develop a shallow cutting algorithm for arcs of Γ in Section 4.1 and then using the algorithm to prove Lemma 7 in Section 4.2.

4.1 Shallow cuttings

Without loss of generality, we assume that ℓ is the x-axis. Let $\mathbb{R}^-$ (resp., $\mathbb{R}^+$) be the half-plane below (resp., above) ℓ. Note that each arc of Γ is x-monotone, every arc has both endpoints on ℓ, and every two arcs cross each other at most once.

We use *$\mathbb{R}^+$-constrained unit disk* to refer to a unit disk with center in $\mathbb{R}^+$ and use *$\mathbb{R}^+$-constrained arc* to refer to a portion of the arc $C \cap \mathbb{R}^-$ for a unit circle C with center in $\mathbb{R}^+$. For any point $q \in \mathbb{R}^-$, let $\rho(q)$ to denote the vertical downward ray from q. We say that an arc γ of Γ is *below* q if it intersects $\rho(q)$. As the center of γ is in $\mathbb{R}^+$ and $\rho_q \in \mathbb{R}^-$, γ intersects ρ_q at most once.

For a parameter $r \leq n$ and a region R of the plane, a *$(1/r)$-cutting covering* R for the arcs of Γ is a set of interior-disjoint cells such that the union of all cells covers R and each cell intersects at most n/r arcs of Γ. For each cell Δ, its *conflict list* Γ_Δ is the set of arcs of Γ that intersect Δ. The size of the cutting is the number of its cells.

For a point $p \in \mathbb{R}^-$, the *level* of p in Γ is the number of arcs of Γ below p. For any integer $k \in [1, n]$, the ($\leq k$)*-level* of Γ, denoted by $L_{\leq k}(\Gamma)$, is defined as the region consisting of all points of $\mathbb{R}^-$ with level at most k. Given parameters $r, k \in [1, n]$, a *k-shallow* $(1/r)$*-cutting* is a $(1/r)$-cutting for Γ that covers $L_{\leq k}(\Gamma)$.

We use *pseudo-trapezoid* to refer to a region that has two vertical line segments as left and right edges, an $\mathbb{R}^+$-constrained arc or a line segment on ℓ as a top edge, and an $\mathbb{R}^+$-constrained arc as a bottom edge. In particular, if a pseudo-trapezoid does not have a bottom edge, i.e., the bottom side is unbounded, then we call it a *bottom-open* pseudo-trapezoid.

We say that a shallow cutting is in the *bottom-open pseudo-trapezoid form* if every cell of it is a bottom-open pseudo-trapezoid. Our main result about the shallow cuttings for Γ is given in the following theorem.

▶ **Theorem 8.** *There exist constants B, C, and C', such that for a parameter $k \in [1, n]$, we can compute a $(B^i k)$-shallow $(CB^i k/n)$-cutting of size at most $C' \frac{n}{B^i k}$ in the bottom-open pseudo-trapezoid form, along with conflict lists of all its cells, for all $i = 0, 1, \ldots, \log_B \frac{n}{k}$, in $O(n \log \frac{n}{k})$ total time. In particular, we can compute a k-shallow (Ck/n)-cutting of size $O(n/k)$, along with its conflict lists, in $O(n \log \frac{n}{k})$ time.*

Since the proof of Theorem 8 is technical and lengthy (and is one of our main results in this paper), we devote the entire Section 5 to it.

4.2 Proving Lemma 7

We now prove Lemma 7. With the shallow cutting algorithm in Theorem 8, we generalize the techniques of [6, 10] for lines to the arcs of Γ. We first give two deletion-only data structures, which will be needed in our fully dynamic data structure for Lemma 7.

4.2.1 Deletion-only data structure

Our first deletion-only data structure is given in Lemma 9 (see the full paper for the proof).

▶ **Lemma 9.** *There is a data structure of $O(n)$ size to maintain a set Γ of n arcs to support $O(\log n)$ amortized time deletions and $O(\sqrt{n} \log^{O(1)} n + k)$ time k-lowest-arcs queries. If a set Γ of n arcs is given initially, the data structure can be constructed in $O(n \log n)$ time.*

We have the following lemma for another deletion-only data structure, obtained by following the same algorithmic scheme as [6, Lemma 6] and replacing their shallow cutting algorithm for lines with our shallow cutting algorithm for arcs of Γ in Theorem 8.

▶ **Lemma 10.** [6, Lemma 6] *For any fixed r, there is a data structure of $O(n \log r)$ size to maintain a set Γ of n arcs to support $O(r \log n)$ amortized time deletions and $O(\log r + n/r + k)$ time k-lowest-arcs queries. If a set Γ of n arcs is given initially, the data structure can be constructed in $O(n \log n)$ time.*

4.2.2 Fully-dynamic data structure for Lemma 7

With the two deletion-only data structures in Lemmas 9 and 10, we are now in a position to describe our fully dynamic data structure for Lemma 7.

Overview. To achieve our result in Lemma 7, roughly speaking, we can simply plug our shallow cutting algorithm for Γ in Theorem 8 and Lemma 9 into the algorithmic scheme of [6] or [10]. The algorithms of [6] and [10] are similar. For the method of [10], we can just replace their shallow cutting algorithm for lines [13] with our shallow cutting algorithm for Γ in Theorem 8 and replace their deletion-only data structure [24] with a combination of Lemmas 9 and 10. In addition, a general technique of querying multiple structures simultaneously from [6, Theorem 1] is also needed. For the method of [6], it was described for the plane problem in 3D with a query time $O(\log^2 n/\log\log n + k)$. We can follow the same algorithmic scheme but using our shallow cutting algorithm for Γ in Theorem 8 and Lemma 9 in the corresponding places. In addition, since our problem is a 2D problem, the technique of dynamic interval trees in [13] can be used to reduce the query time component from $O(\log^2 n/\log\log n)$ to $O(\log n)$. In the following, we adapt the method from Chan [10].

The data structure is an adaptation of the one for dynamic 3D convex hulls [9,11,23] (the idea was originally given in [9] and subsequent improvements were made in [11,23]). We first give the following lemma. The lemma is similar to [10, Theorem 3.1], which is based on the result in [9], but Lemma 11 provides slightly better complexities than [10, Theorem 3.1] by using the recently improved result of [11] (see the full paper for more details).

▶ **Lemma 11** ([9–11,23]). *Let Γ be a set of arcs, which initially is $\emptyset$ and undergoes n updates (insertions and deletions). For any $b \geq 2$, we can maintain a collection of shallow cuttings T_i^j in the bottom-open pseudo-trapezoid form, $i = 1, 2, \ldots, \lceil \log n \rceil$, $j = 1, 2, \ldots, O(\log_b n)$, in $b^{O(1)} \log^5 n$ amortized time per update such that the following properties hold.*

1. *Each cutting T_i^j is of size $O(2^i)$ and never changes until it is replaced by a new one created from scratch. The total size of all cuttings created over time is $b^{O(1)} \log^3 n$.*
2. *Each cell $\Delta \in T_i^j$ is associated with a list L_Δ of $O(n/2^i)$ arcs of Γ. Each list L_Δ undergoes deletions only after its creation. The total size of all such lists created over time is $b^{O(1)} n \log^4 n$.*
3. *For any $k \geq 1$, let $i_k = \lceil \log(n/Ck) \rceil$ for a sufficiently large constant C. For any vertical line ℓ^*, if an arc $\gamma \in \Gamma$ is among the k lowest arcs at ℓ^*, then there exists some j such that γ is in the list L_{Δ^j} of the cell $\Delta^j \in T_{i_k}^j$ intersecting ℓ^*.*
4. *At any moment, for each i, the number of cells of the current cuttings T_i^j for all j is $O(2^i)$. This implies that the total size of the lists L_Δ of all cells Δ of all current cuttings T_i^j at any moment is $O(n \log n)$.*

With Lemma 11, we can answer a k-lowest-arcs query as follows. Consider a query vertical line ℓ^*. By Lemma 11(3), for each j, we compute the cell Δ^j of $T_{i_k}^j$ intersecting ℓ^*, which takes $O(\log n)$ time by binary search as the x-projections of $T_{i_k}^j$ partition the x-axis into intervals. Then, we use "brute-force" to find the k lowest arcs among all arcs in L_{Δ^j} in $O(k)$ time as $|L_{\Delta^j}| = O(k)$. Finally, among all arcs found above, we return the k lowest arcs, which takes $O(k \log_b n)$ time. As such, the total query time is $O((\log n + k) \log_b n)$.

An improved query algorithm. We now improve the query time. We store each list L_Δ by a deletion-only data structure that supports k-lowest-arcs queries. Suppose that such a deletion-only data structure is of space $S_0(|L_\Delta|)$, supports each k-lowest-arcs query in $O(Q_0(|L_\Delta|) + k)$ time and $D_0(|L_\Delta|)$ deletion time, and can be built in $O(P_0(|L_\Delta|))$ time. Then, by Lemma 11(2), each update causes at most $b^{O(1)} \log^4 n$ amortized number of deletions to the lists L_Δ, and thus the amortized update time is

$$U(n) = b^{O(1)} \log^5 n + \max_{\Delta \in T_i^j} D_0(|L_\Delta|) \cdot b^{O(1)} \log^4 n + P_0(b^{O(1)} n \log^4 n)/n. \qquad (1)$$

Note that the last term is obtained due to the following. After every n updates, we reconstruct the entire data structure and thus the reconstruction time is on the order of $\sum_{\Delta\in T_i^j} P_0(|L_\Delta|)$, which is bounded by $P_0(b^{O(1)} n \log^4 n)$ since $\sum_{\Delta\in T_i^j} |L_\Delta| = b^{O(1)} n \log^4 n$ by Lemma 11(2) (assuming that $P_0(n) = \Omega(n)$).

By Lemma 11(4), the total space is

$$S(n) = O\left(\sum_{i=1}^{\log n} 2^i \cdot S_0(n/2^i)\right). \tag{2}$$

For each query, there are two tasks: (1) Compute the cell Δ^j for every j; (2) find the k lowest arcs of all lists L_{Δ^j} for all j. In the following, we solve the first task in $O(\log n + \log_b n)$ time and solve the second task in $O(\log n + k)$ time.

For the first task, following the method in [10], for each i, we use a dynamic interval tree [5] to store the intervals of the x-projections of the cuttings T_i^j for all j in an interval tree T_i. Using T_i, all t intervals intersecting ℓ^* can be computed in $O(\log n + t)$ time. In our problem, $t = O(\log_b n)$. Insertions and deletions of intervals on T_i can be supported in $O(\log n)$ amortized time. We can thus maintain all interval trees T_i in additional amortized $\log n \cdot b^{O(1)} \log^3 n$ time per update as the total size of all cuttings over time is $b^{O(1)} \log^3 n$ by Lemma 11(1). This additional update time is subsumed by the first term in (1). In this way, the first task can be solved in $O(\log n + \log_b n)$ time.

For the second task, instead of brute-force, we use the deletion-only data structures for the lists L_{Δ^j} and resort to a technique of querying multiple k-lowest-arcs data structures simultaneously in [6, Theorem 1], which is based on an adaption of the heap selection algorithm of Frederickson [22]. Applying [6, Theorem 1], the second task can be accomplished in $O(k + \log_b n \cdot \max_j Q_0(|L_{\Delta^j}|))$ time, which is $O(k + \log_b n \cdot Q_0(O(k)))$ since the size of each $|L_{\Delta^j}|$ is $O(k)$.

Combining the complexities of the first and second tasks, the overall query time is

$$Q(n) = O(\log n + \log_b n + k) + Q_0(O(k)) \cdot \log_b n. \tag{3}$$

Let $m = |L_\Delta|$. Depending on m, we use different deletion-only data structures for L_Δ.

1. If $m \geq \log^3 n$, then we use Lemma 9 to handle L_Δ with $P_0(m) = O(m \log m)$, $S_0(m) = O(m)$, $D_0(m) = O(\log m)$, $Q_0(m) = m^{1/2} \log^{O(1)} m$. Plugging them into (1), (2), and (3) and setting $b = \log^\epsilon n$, we obtain $U(n) = O(\log^{5+\epsilon} n)$, $S(n) = O(n \log n)$, and $Q(n) = O(\log n + k + k^{1/2} \log^{O(1)} k \cdot \log n / \log\log n)$. Since $m = O(k)$, we have $k = \Omega(m)$. As $m \geq \log^3 n$, we have $k = \Omega(\log^3 n)$. Therefore, $Q(n) = O(\log n + k)$.
2. If $m < \log^3 n$, then we use Lemma 10 to handle L_Δ by setting $r = \log n / \log\log n$. This results in $P_0(m) = O(m \log m)$, $S_0(m) = O(m \log\log n)$, $D_0(m) = O(\log n \log m / \log\log n)$, $Q_0(m) = O(\log\log n + m \log\log n / \log n)$. Since $m < \log^3 m$, we have $D_0(m) = O(\log n)$. Plugging them into (1) and (3) and setting $b = \log^\epsilon n$, we obtain $U(n) = O(\log^{5+\epsilon} n)$ and $Q(n) = O(\log n + k + (\log\log n + k \log\log n / \log n) \cdot \log n / \log\log n)$, which is $O(\log n + k)$. For the space, since $m < \log^3 n$, if we plug $S_0(m) = O(m \log\log n)$ into (2), we only need to consider those i's such that $n/2^i < \log^3 n$. There are only $O(\log\log n)$ such i's. Therefore, we obtain $S(n) = O(n \log^2 \log n)$ for all such m's in this case.

Combining the above two cases leads to $U(n) = O(\log^{5+\epsilon} n)$, $S(n) = O(n \log n)$, and $Q(n) = O(\log n + k)$. We can actually obtain a better bound for the insertion time. If $P'(n)$ is the preprocessing time for constructing the data structure for a set of n arcs, then the amortized insertion time $I(n)$ is bounded by $I(n) = O(b \log_b n \cdot P'(n)/n)$ [6, 11]. According

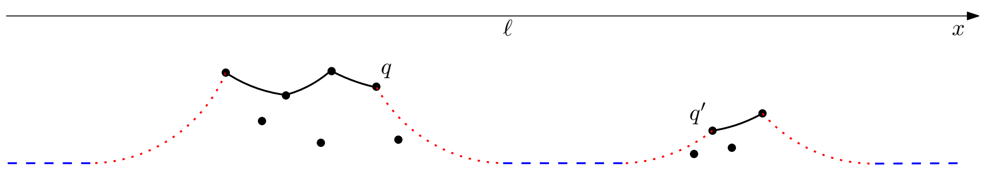

Figure 1 Illustration the boundary of $H_\ell(Q)$, where Q is the set of points below the x-axis ℓ. It consists of three (blue) dashed horizontal line segments of y-coordinates -1, four (red) dotted $\mathbb{R}^+$-constrained arcs with centers on ℓ, and four other solid $\mathbb{R}^+$-constrained arcs. The region below the boundary is $H_\ell(Q)$.

to our above discussion and Lemma 11(4), constructing the deletion-only data structures for all lists L_Δ is $O(n \log^2 n)$. The shallow cuttings in Lemma 11 can be built in $O(n \log^2 n)$ following the method in [11] and using our shallow cutting algorithm in Theorem 8. In this way, we can bound the amortized insertion time by $O(b \log_b n \log^2 n)$, which is $O(\log^{3+\epsilon} n)$ with $b = \log^\epsilon n$. This proves Lemma 7.

5 Algorithm for shallow cuttings

In this section, we prove Theorem 8. We follow the notation in Section 4.1.

As in [13], we use parameter $K = n/r$ instead of r. A k-shallow $(1/r)$-cutting becomes a k-shallow (K/n)-cutting and we use a (k, K)-shallow cutting to represent it. It has the following properties: (1) $|\Gamma_\Delta| \leq K$ for each cell Δ; (2) the union of all cells covers $L_{\leq k}(\Gamma)$. Theorem 8 says that a $(k, O(k))$-shallow cutting of size $O(n/k)$ in the bottom-open pseudo-trapezoid form can be computed in $O(n \log \frac{n}{k})$ time.

Following the analysis of Matoušek [25], we start with the following lemma (see the full paper for the proof), which shows the existence of the shallow cutting in *the pseudo-trapezoid form*, i.e., each cell is a pseudo-trapezoid but not necessarily bottom-open.

▶ **Lemma 12.** *For any $k \in [1, n]$, there exists a $(k, O(k))$-shallow cutting of size $O(n/k)$ in the pseudo-trapezoid form.*

Chan and Tsakalidis [13] introduced a vertex form of the shallow cutting for lines. Here for arcs of Γ, using vertices is not sufficient. We will introduce a vertex-segment form in Section 5.2. The definition requires a concept, which we call *line-separated α-hulls* and is discussed in Section 5.1. In Section 5.3, we present our algorithm to compute shallow cuttings in the vertex-segment form.

5.1 Line-separated α-hulls

The line-separated α-hull is an extension of the α-hull introduced in [21]. In [21], α-hull is considered for all values $\alpha \in (-\infty, \infty)$. For our problem, we only consider the value $\alpha = -1$.

Let Q be a set of points in $\mathbb{R}^-$. We define the *line-separated α-hull $H_\ell(Q)$* of Q with respect to the x-axis ℓ as the complement of the union of all unit disks with centers in $\mathbb{R}^+$ that do not contain any point of Q (so the disk centers and the points of Q are separated by ℓ, which is why we use "line-separated"; see Fig. 1).

Many of the properties of the α-hulls [21] can be extended to the line-separated case. We list some of these in the following observation; the proof is a straightforward extension of that in [21] by adding the "line-separated" constraint.

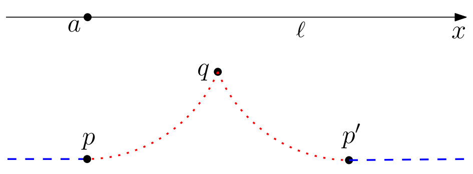

Figure 2 Illustration the wings of the a point q. The two (red) dotted curves are wing arcs and the two (blue) dashed segments are wing half-lines. p and p' are the left and right wing vertices, respectively.

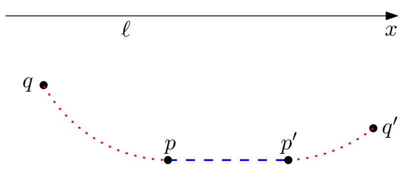

Figure 3 Illustration two points q and q' that are in far-away position. The two (red) dotted arcs and the (blue) dashed segments in between constitute $\beta(q, q')$.

▶ **Observation 13.**

1. *$Q \subseteq H_\ell(Q)$, and for any subset $Q' \subseteq Q$, $H_\ell(Q') \subseteq H_\ell(Q)$.*
2. *A point $q \in Q$ is a vertex of $H_\ell(Q)$ if and only if there exists a unit disk with center in $\mathbb{R}^+$ and its boundary containing q such that the interior of the disk does not contain any point of Q.*
3. *If there is an $\mathbb{R}^+$-constrained arc connecting two points of Q such that the interior of the underlying disk of the arc does not contain any point of Q, then the arc is an edge of $H_\ell(Q)$.*

For any two points $q, q' \in \mathbb{R}^-$ that can be covered by a $\mathbb{R}^+$-constrained unit disk, there exists a unique $\mathbb{R}^+$-constrained arc that connects q and q'; we use $\gamma(q, q')$ to denote that arc.

5.1.1 Algorithm for computing $H_\ell(Q)$

By slightly modifying the algorithm of [21], $H_\ell(Q)$ can be computed in $O(m \log m)$ time, where $m = |Q|$. The algorithm also suggests that $\partial H_\ell(Q)$ is x-monotone. In the following, assuming that the points of Q are already sorted from left to right as $q_1, q_2, \ldots, q_m$, we give a linear time algorithm to compute $H_\ell(Q)$, which is similar in spirit to Graham's scan for computing convex hulls.

Irrelevant points. Note that if a point $q \in Q$ whose y-coordinate is smaller than or equal to -1, then q must be in $H_\ell(Q)$ because every $\mathbb{R}^+$-constrained disk does not contain q in the interior. Hence, in that case q is *irrelevant* for computing $H_\ell(Q)$ and thus can be ignored. If all points of Q are irrelevant, then $H_\ell(Q)$ is simply the region below the horizontal line whose y-coordinate is -1. In the following, we assume that every point of Q is relevant.

Wings. Consider a point $q \in Q$. Let a be a point on the x-axis ℓ with $x(a) < x(q)$ such that q is on the unit circle C_a centered at a. Let p be the lowest point of C_a. We define the *left wing* of q to be the concatenation of the following two parts (see Fig. 2): (1) the arc of $C_a \cap R^-$ between q and p, called the *left wing arc*, and the horizontal half-line with p as the right endpoint, called the *left wing half-line*. The point p is called the *left wing vertex* of q. We define the *right wing* of q and the corresponding concepts symmetrically. The left and right wings together actually form the boundary of the line-separated α-hull of $\{q\}$. In Fig. 1, the four (red) dotted arcs are wing arcs and the three (blue) dashed segments are on wing half-lines.

Far-away position. Consider two points $q, q' \in \mathbb{R}^-$ such that $x(q) < x(q')$. We say that (q, q') are in *far-away* position if $x(p) < x(p')$ holds, where p is the right wing vertex of q and p' is the left wing vertex of q' (see Fig. 3). In this case, q' is above the right wing of q

and there is no $\mathbb{R}^+$-constrained unit disk covering both q and q'. We use $\beta(q, q')$ to denote the concatenation of the right wing arc of q, the segment $\overline{pp'}$, and the left wing arc of q'. In fact, the left wing of q, $\beta(q, q')$, and the right wing of q' together form the boundary of the line-separated α-hull of $\{q, q'\}$. In Fig. 1, q and q' are also in far-away position.

The algorithm. Define $Q_i = \{q_1, \ldots, q_i\}$ for each $1 \leq i \leq m$. Our algorithm handles the points of Q incrementally from q_1 to q_m. For each q_i, the algorithm computes $H_\ell(Q_i)$ by updating $H_\ell(Q_{i-1})$ with q_i. Suppose that $q_{i_1}, q_{i_2}, \ldots, q_{i_t}$ are the points of Q_i that are the vertices of $H_\ell(Q_i)$ sorted from left to right. Then, our algorithm maintains the following invariant: the boundary $\partial H_\ell(Q_i)$ is x-monotone and consists of the following parts from left to right: the left wing of q_{i_1}, $\gamma(q_{i_j}, q_{i_{j+1}})$ or $\beta(q_{i_j}, q_{i_{j+1}})$, for each $j = 1, 2, \ldots, t-1$ in order, and the right wing of q_{i_t}. $H_\ell(Q_i)$ is the region below $\partial H_\ell(Q_i)$.

Initially, for q_1, we set $\partial H_\ell(Q_1)$ to the concatenation of the left wing and the right wing of q_1. In general, suppose we already have $\partial H_\ell(Q_{i-1})$. We compute $\partial H_\ell(Q_i)$ as follows. We process the points of $q_{i_1}, q_{i_2}, \ldots, q_{i_t}$ in the backward order. For ease of exposition, we assume that $t > 1$; the special case $t = 1$ can be easily handled.

We first process the point q_{i_t}. If q_{i_t} and q_i are in the far-away position, then we delete the right wing of q_{i_t} from $H_\ell(Q_{i-1})$ and add $\beta(q_{i_t}, q_i)$ and the right wing of q_i. This finishes computing $H_\ell(Q_i)$. Below, we assume that q_{i_t} and q_i are not in the far-away position.

If q_i is below the right wing of q_{i_t}, then q_i is inside $H_\ell(Q_{i-1})$. In this case, $H_\ell(Q_i)$ is $H_\ell(Q_{i-1})$ and we are done. If q_i is above the right wing of q_{i_t}, then we further check whether the arc $\gamma(q_{i_t}, q_i)$ exists (which is true if and only if there exists an $\mathbb{R}^+$-constrained unit disk covering both q_{i_t} and q_i).

- If $\gamma(q_{i_t}, q_i)$ does not exist (in this case q_{i_t} must be below the left wing of q_i and thus q_{i_t} does not contribute to $H_\ell(Q_i)$ because it is "dominated" by q_i), then we "prune" q_{i_t} from $\partial H_\ell(Q_{i-1})$, i.e., delete the right wing of q_{i_t} and also delete $\gamma(q_{i_{t-1}}, q_{i_t})$ or $\beta(q_{i_{t-1}}, q_{i_t})$ whichever exists in $H_\ell(Q_{i-1})$. Next, we process $q_{i_{t-1}}$ following the same algorithm.
- If $\gamma(q_{i_t}, q_i)$ exists, then we further check whether D contains $q_{i_{t-1}}$, where D is the underlying disk of $\gamma(q_{i_t}, q_i)$. If $q_{i_{t-1}} \in D$, then q_{i_t} must be in $H_\ell(\{q_{i_{t-1}}, q_i\})$ and thus does not contribute to $H_\ell(Q_i)$. In this case, we prune q_{i_t} as above and continue processing $q_{i_{t-1}}$. If $q_{i_{t-1}} \notin D$, we delete the right wing of q_{i_t} from $\partial H_\ell(Q_{i-1})$ and add the arc $\gamma(q_{i_t}, q_i)$ and the right wing of q_i; this finishes computing $H_\ell(Q_i)$.

Clearly, the runtime for computing $H_\ell(Q_i)$ is $O(1 + t')$, where t' is the number of points of $q_{i_1}, q_{i_2}, \ldots, q_{i_t}$ pruned from $H_\ell(Q_{i-1})$. The overall algorithm for computing $H_\ell(Q)$ takes $O(m)$ time since once a point is pruned it will never appear on the hull again, which resembles Graham's scan for computing convex hulls.

5.1.2 Vertical decompositions

According to the above discussion, $H_\ell(Q)$ has at most $5m$ vertices with $m = |Q|$, including all wing vertices. The vertical downward rays from all vertices partition $H_\ell(Q)$ into at most $5m$ bottom-open pseudo-trapezoids and rectangles. We call this partition the *vertical decomposition* of $H_\ell(Q)$, denoted by $\mathrm{VD}(Q)$.

In our later discussion, we need to combine Q with a set S of pairwise disjoint segments on ℓ whose endpoints are all in Q. For each segment $s \in S$, we draw a vertical downward ray from each endpoint of s; let $R(s)$ denote the bottom-open rectangular region bounded by the two rays and s. The regions $R(s)$ for all segments $s \in S$ form the *vertical decomposition* of S, denoted by $\mathrm{VD}(S)$.

We combine VD(Q) and VD(S) to form a vertical decomposition of (Q, S), denoted by VD($Q \cup S$) as follows. Let U be the upper envelope of $H_\ell(Q)$ and S. We draw a vertical downward ray from each vertex of v of U. These rays divide the region below U into cells, each of which is bounded by two vertical rays from left and right, and bounded from above by a line segment or an $\mathbb{R}^+$-constrained arc. These cells together form the decomposition VD($Q \cup S$). In the following, depending on the context, VD(Q) may refer to the region covered by all cells of it; the same applies to VD(S) and VD($Q \cup S$). As such, we have VD($Q \cup S$) = VD(Q) ∪ VD(S). Note that since the endpoints of all segments of S are in Q and on ℓ, the boundary ∂VD($Q \cup S$) is x-monotone.

5.2 Shallow cuttings in the vertex-segment form

We introduce a vertex-segment form of the shallow cutting. Given parameters $k, K \in [1, n]$ with $k \leq K$, a (k, K)-shallow cutting for the arcs of Γ in the *vertex-segment* form is a set Q of points in $\mathbb{R}^-$ along with a set S of interior pairwise-disjoint segments on ℓ such that the following conditions hold:
1. The endpoints of all segments of S are in Q.
2. Every point of Q has level at most K in Γ.
3. Every segment of S intersects at most K arcs of Γ.
4. VD($Q \cup S$) covers $L_{\leq k}(\Gamma)$.

The *conflict list* of a point $q \in Q$, denoted by Γ_q, is the set of arcs of Γ below q. Note that $|\Gamma_q| \leq K$ as the level of q is at most K. The *conflict list* of a segment $s \in S$, denoted by Γ_s, is the set of arcs intersecting s. The conflict lists of (Q, S) refer to the conflict lists of all points of Q and all segments of S. The *size* of the cutting is defined to be $|Q|$. Observe that since the endpoints of all segments of S are in Q and the segments of S are interior pairwise-disjoint, we have $|S| < |Q|$. Therefore, VD($Q \cup S$) has $O(|Q|)$ cells, and more specifically, at most $5|Q|$ cells. Further, we have following observation.

▶ **Observation 14.** *Suppose that (Q, S) is a (k, K)-shallow cutting for Γ in the vertex-segment form. Then every cell of VD($Q \cup S$) intersects at most $3K$ arcs of Γ.*

Proof. Consider a cell Δ of VD($Q \cup S$). Let e be the top edge of Δ. By the definition of VD($Q \cup S$), e is one of the following: a segment of S, an arc $\gamma(q, q')$ for two points $q, q' \in Q$, a wing arc of a point $q \in Q$, and a segment of a wing half-line of a point $q \in Q$. Below, we argue $|\Gamma_\Delta| \leq 3K$ for each of these cases, where Γ_Δ is the set of arcs of Γ intersecting Δ.

1. If e a segment of S, then recall that $|\Gamma_e| \leq K$. Also, the size of the conflict list of each endpoint of e is at most K. For any arc $\gamma \in \Gamma$ intersecting Δ, since the center of γ is in $\mathbb{R}^+$, γ must either intersect e or in the conflict list of at least one endpoint of e. Therefore, $|\Gamma_\Delta| \leq 3K$ holds.
2. If e is an arc $\gamma(q, q')$ for two points $q, q' \in Q$, then any arc $\gamma \in \Gamma$ intersecting Δ must be in the conflict list of one of q and q'. Hence, we have $|\Gamma_\Delta| \leq 2K$.
3. If e is a wing arc γ of a point $q \in Q$, then q is an endpoint of γ. Let p be the other endpoint of γ. By definition, the y-coordinate of p is -1. Thus, no arc of Γ is below p. By definition, the radius of γ is 1 and the center of γ is on ℓ. Hence, any arc of Γ intersecting Δ must be in the conflict list of q and thus $|\Gamma_\Delta| \leq |\Gamma_q| \leq K$.
4. If e is a segment s of a wing half-line of a point $q \in Q$, then by definition e is horizontal and has y-coordinate equal to -1. As centers of all arcs of Γ are in $\mathbb{R}^+$, no arc of Γ can intersect Δ. Hence, $|\Gamma_\Delta| = 0$.

Combining all the above cases leads to $|\Gamma_\Delta| \leq 3K$. ◀

In the next two lemmas, we show that shallow cuttings in the vertex-segment form and in the pseudo-trapezoid form can be transformed to each other.

▶ **Lemma 15.** *A (k, K)-shallow cutting of size t in the pseudo-trapezoid form can be transformed into a $(k, k+K)$-shallow cutting in the vertex-segment form of size $O(t)$.*

Proof. Let Ξ be a (k, K)-shallow cutting of size t in the pseudo-trapezoid form. Without loss of generality, we assume that all cells of Ξ intersect $L_{\leq k}(\Gamma)$. Define Q to be the set of vertices of all cells of Ξ. Define S to be the top edges of all cells of Ξ that are segments of ℓ. Since the interiors of cells of Ξ are pairwise disjoint, the segments of S are also interior pairwise-disjoint. As Ξ has t cells, we have $|Q| = O(t)$. In the following, we argue that (Q, S) is a $(k, k+K)$-shallow cutting in the vertex-segment form.

First of all, by definition, endpoints of all segments of S are in Q. Consider a point $q \in Q$, which is a vertex of a cell $\Delta \in \Xi$. As Δ intersects $L_{\leq k}(\Gamma)$ and $|\Gamma_\Delta| \leq K$, there are at most $k+K$ arcs of Γ below q. Hence, q has level at most $k+K$ in Γ. For each segment $s \in S$, since it is a top edge of a cell $\Delta \in \Xi$ and $|\Gamma_\Delta| \leq K$, we obtain $|\Gamma_s| \leq K$.

It remains to argue that $\mathrm{VD}(Q \cup S)$ covers $L_{\leq k}(\Gamma)$. By definition, the union of all cells of Ξ covers $L_{\leq k}(\Gamma)$. Consider a cell $\Delta \in \Xi$, which is a pseudo-trapezoid. We show that $\Delta \subseteq \mathrm{VD}(Q \cup S)$, which will prove that $\mathrm{VD}(Q \cup S)$ covers $L_{\leq k}(\Gamma)$.

Let e be the top edge of Δ. As Δ is a pseudo-trapezoid, e is either a segment on ℓ or an $\mathbb{R}^+$-constrained arc. If e is a segment of ℓ, then $e \in S$ and thus Δ is a cell of $\mathrm{VD}(S)$. Hence, $\Delta \subseteq \mathrm{VD}(S) \subseteq \mathrm{VD}(Q \cup S)$. If e is an $\mathbb{R}^+$-constrained arc, then let q_1 and q_2 be its two endpoints; thus e is the arc $\gamma(q_1, q_2)$. Since Δ is a pseudo-trapezoid with $\gamma(q_1, q_2)$ as the top edge, Δ must be contained in the line-separated α-hull $H_\ell(\{q_1, q_2\})$, which is a subset of $H_\ell(Q)$ by Observation 13(1) as $q_1, q_2 \in Q$. Recall that $\mathrm{VD}(Q)$ is the vertical decomposition of $H_\ell(Q)$. Hence, we have $\Delta \subseteq \mathrm{VD}(Q) \subseteq \mathrm{VD}(Q \cup S)$.

This proves $\Delta \subseteq \mathrm{VD}(Q \cup S)$ and therefore $\mathrm{VD}(Q \cup S)$ covers $L_{\leq k}(\Gamma)$. ◀

▶ **Lemma 16.** *A (k, K)-shallow cutting of size t in the vertex-segment form can be transformed into a $(k, 3K)$-shallow cutting of size $O(t)$ in the bottom-open pseudo-trapezoid form.*

Proof. Let (Q, S) be a (k, K)-shallow cutting of size t in the vertex-segment form. We intend to take the vertical decomposition $\mathrm{VD}(Q, S)$ as the $(k, 3K)$-shallow cutting Ξ of size $O(t)$ in the bottom-open pseudo-trapezoid form. However, a subtle issue is that some cells of $\mathrm{VD}(Q, S)$ might not be pseudo-trapezoids. More specifically, consider a cell $\Delta \in \mathrm{VD}(Q, S)$. Let e be the top edge of Δ. According to the definition of $\mathrm{VD}(Q, S)$, e belongs to one of the three cases: (1) e is an $\mathbb{R}^+$-constrained arc; (2) e is a segment of ℓ; (3) e is a segment of a wing half-line of a point of Q. In the first two cases, Δ is a bottom-open pseudo-trapezoid and we include Δ in Ξ. In the third case, Δ is not a pseudo-trapezoid by our definition since e is a line segment but not on ℓ. In this case, we extend Δ by moving e upwards until ℓ to obtain an extended cell Δ', which is a bottom-open pseudo-trapezoid; we add Δ' to Ξ. We call Δ' a *special cell* of Ξ.

We claim that $\Gamma_{\Delta'} = \emptyset$, i.e., Δ' does not intersect any arcs of Γ. Indeed, let e' be the top edge of Δ'. Let R be the rectangular region of Δ' between e and e' (see Fig. 4). Then, $\Delta' = \Delta \cup R$. Consider any point $p \in R$. We argue that no arc of Γ is below p, which will prove the claim. Assume to the contradiction that there is an arc $\gamma \in \Gamma$ below p. Without loss of generality, we assume that γ is the lowest arc of Γ intersecting the vertical downward ray $\rho(p)$. Let p' be the intersection of γ and $\rho(p)$. By definition, $p' \in L_{\leq 0}(\Gamma)$. Since (Q, S) is a (k, K)-shallow cutting, $\mathrm{VD}(Q, S)$ covers $L_{\leq k}(\Gamma)$ and thus covers $L_{\leq 0}(\Gamma)$ as $k \geq 0$. Therefore, $\mathrm{VD}(Q, S)$ covers p'. On the other hand, since e is on a wing half-line of a point of Q, the

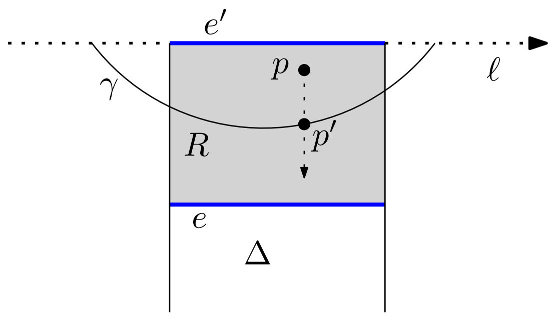

Figure 4 Illustrating the proof of $\Gamma_{\Delta'} = \emptyset$, with $\Delta' = R \cup \Delta$, where R is the gray rectangle and Δ is the region below e.

y-coordinate of e is -1 and thus no arcs of Γ intersect e. Hence, p' must be above e. But since e is the top edge of the cell $\Delta \in \mathrm{VD}(Q,S)$, p' cannot be covered by $\mathrm{VD}(Q,S)$, a contradiction. This proves that $\Gamma_{\Delta'} = \emptyset$.

Since $|Q| = t$, $\mathrm{VD}(Q \cup S)$ has $O(t)$ cells. By definition, the size of Ξ is $O(t)$. We next show that Ξ is a $(k, 3K)$-shallow cutting in the bottom-open pseudo-trapezoid form. First of all, by definition, each cell of $\mathrm{VD}(Q,S)$ is a bottom-open pseudo-trapezoid. Also, since $\mathrm{VD}(Q,S)$ covers $L_{\leq k}(\Gamma)$ and each cell of $\mathrm{VD}(Q,S)$ is either in Ξ or contained in a cell of Ξ, $\mathrm{VD}(Q,S)$ is a subset of Ξ. Therefore, Ξ covers $L_{\leq k}(\Gamma)$. For each Δ of Ξ, if it is a special cell, then $|\Gamma_\Delta| = 0$ as proved above; otherwise Δ is also a cell in $\mathrm{VD}(Q,S)$ and we have $|\Gamma_\Delta| \leq 3K$ by Observation 14. Therefore, Ξ is a $(k, 3K)$-shallow cutting in the bottom-open pseudo-trapezoid form. ◀

Combining Lemmas 12 and 15 leads to the following.

▶ **Corollary 17.** *Given $k \in [1,n]$, there exists a $(k, O(k))$-shallow cutting of size $O(n/k)$ in the vertex-segment form.*

5.3 Computing shallow cuttings in the vertex-segment form

In what follows, by extending the algorithm of [13], we present an algorithm to compute shallow cuttings for Γ in the vertex-segment form.

We say that a (standard) cutting is in the *pseudo-trapezoid form* if every cell of the cutting is a pseudo-trapezoid. The following result was known previously [15, 28].

▶ **Lemma 18** ([15,28]). *Given any constant $\epsilon > 0$, an ϵ-cutting for Γ in the pseudo-trapezoid form of $O(1)$ size covering the plane, along with its conflict lists, can be computed in $O(n)$ time.*

We say that a shallow cutting (Q,S) in the vertex-segment form is *sorted* if points of Q are sorted by their x-coordinates. The following is the main theorem about our algorithm.

▶ **Theorem 19.** *There exist constants B, C, C', such that for any parameter $k \in [1,n]$, given a (Bk, CBk)-shallow cutting (Q_{IN}, S_{IN}) in the sorted vertex-segment form for Γ of size at most $C' \frac{n}{Bk}$ along with its conflict lists, we can compute a (k, Ck)-shallow cutting (Q_{OUT}, S_{OUT}) in the sorted vertex-segment form for Γ of size at most $C' \frac{n}{k}$ along with its conflict lists in $O(n)$ time.*

Proof. Let ϵ be a constant to be set later. We begin by computing the decomposition $\mathrm{VD}(Q_{\mathrm{IN}}, S_{\mathrm{IN}})$. Since Q_{IN} is sorted, $H_\ell(Q_{\mathrm{IN}})$ can be computed in $O(|Q_{\mathrm{IN}}|)$ time using the algorithm from Section 5.1. As the endpoints of all segments of S_{IN} are in Q_{IN} and segments of S_{IN} are interior pairwise-disjoint on ℓ, the segments of S_{IN} can also be sorted from left to right in $O(|Q_{\mathrm{IN}}|)$ time. As such, computing $\mathrm{VD}(Q_{\mathrm{IN}}, S_{\mathrm{IN}})$ can be done in $O(|Q_{\mathrm{IN}}|)$ time, which is $O(n/k)$ as $|Q_{\mathrm{IN}}| \le C' \frac{n}{Bk}$.

Next, for each cell $\Delta \in \mathrm{VD}(Q_{\mathrm{IN}}, S_{\mathrm{IN}})$, we perform the following two steps.

1. Compute an ϵ-cutting Ξ_Δ of size $O(1)$ for Γ_Δ. We clip the cells of Ξ_Δ to lie within Δ (and redecompose each new cell into pseudo-trapezoids if needed). Let Q_Δ denote the set of vertices of all cells of Ξ_Δ and S_Δ the set of top edges of the cells of Ξ_Δ that are segments of ℓ.
   Since $\epsilon = O(1)$, Ξ_Δ has $O(1)$ cells and computing Ξ_Δ takes $O(|\Gamma_\Delta|)$ time by Lemma 18. Hence, both $|Q_\Delta|$ and $|S_\Delta|$ are $O(1)$. As $\sum_{\Delta \in \mathrm{VD}(Q_{\mathrm{IN}}, S_{\mathrm{IN}})} |\Gamma_\Delta| = O(n)$, the total time of this step for all cells $\Delta \in \mathrm{VD}(Q_{\mathrm{IN}}, S_{\mathrm{IN}})$ is $O(n)$.
2. Compute by brute force a smallest subset $Q'_\Delta \subseteq Q_\Delta$, along with a subset $S'_\Delta \subseteq S_\Delta$, such that the following conditions are satisfied.
   a. The endpoints of all segments of S'_Δ are in Q'_Δ.
   b. Every vertex in Q'_Δ has level in Γ_Δ at most Ck.
   c. Every segment in S'_Δ intersects at most Ck arcs of Γ_Δ.
   d. For each cell $\sigma \in \Xi_\Delta$ whose vertices are all in $L_{\le 2k}(\Gamma_\Delta)$, σ is covered by $\mathrm{VD}(Q'_\Delta, S'_\Delta)$.
   As both $|Q_\Delta|$ and $|S_\Delta|$ are $O(1)$, there are $O(1)$ different pairs of Q'_Δ and S'_Δ. For each such pair (Q'_Δ, S'_Δ), we can check whether the four conditions are satisfied in $O(|\Gamma_\Delta|)$ time, because $|Q'_\Delta|$, $|S'_\Delta|$, and the size of Ξ_Δ are all $O(1)$. Hence, finding a smallest subset Q'_Δ with S'_Δ takes $O(|\Gamma_\Delta|)$ time. After that, for each point $q \in Q'_\Delta$, its conflict list in Γ_Δ, which is also its conflict list in Γ, can be found in $O(|\Gamma_\Delta|)$ time. Similarly, for each segment of S'_Δ, its conflict list can be found in $O(|\Gamma_\Delta|)$ time. As both $|Q'_\Delta|$ and $|S'_\Delta|$ are $O(1)$, finding the conflict lists of (Q'_Δ, S'_Δ) takes $O(|\Gamma_\Delta|)$ time. Since $\sum_{\Delta \in \mathrm{VD}(Q_{\mathrm{IN}}, S_{\mathrm{IN}})} |\Gamma_\Delta| = O(n)$, the runtime of this step for all $\Delta \in \mathrm{VD}(Q_{\mathrm{IN}}, S_{\mathrm{IN}})$ is $O(n)$.

We define $Q_{\mathrm{OUT}} = \bigcup_{\Delta \in \mathrm{VD}(Q_{\mathrm{IN}}, S_{\mathrm{IN}})} Q'_\Delta$ and $S_{\mathrm{OUT}} = \bigcup_{\Delta \in \mathrm{VD}(Q_{\mathrm{IN}}, S_{\mathrm{IN}})} S'_\Delta$. We can sort the points of Q_{OUT} in $O(n/k)$ time as follows. For each $\Delta \in \mathrm{VD}(Q_{\mathrm{IN}}, S_{\mathrm{IN}})$, we sort the points of Q'_Δ, which takes $O(1)$ time as $|Q'_\Delta| = O(1)$. Then, for all cells $\Delta \in \mathrm{VD}(Q_{\mathrm{IN}}, S_{\mathrm{IN}})$ in order from left to right, we concatenate the sorted lists of Q'_Δ, and the resulting list is a sorted list of Q_{OUT}. This takes $O(n/k)$ time in total as $\mathrm{VD}(Q_{\mathrm{IN}}, S_{\mathrm{IN}})$ has $O(n/k)$ cells.

The total runtime of the above algorithm is $O(n)$.

Correctness. In the following, we argue the correctness, i.e., prove that $(Q_{\mathrm{OUT}}, S_{\mathrm{OUT}})$ is a (k, Ck)-shallow cutting for Γ of size at most $C' \frac{n}{k}$. We first show that $(Q_{\mathrm{OUT}}, S_{\mathrm{OUT}})$ is a (k, Ck)-shallow cutting and then bound its size.

Consider a cell $\Delta \in \mathrm{VD}(Q_{\mathrm{IN}}, S_{\mathrm{IN}})$. For each point $q \in Q'_\Delta$, according to our algorithm, q has level in Γ_Δ at most Ck. In light of the definition of $\mathrm{VD}(Q_{\mathrm{IN}}, S_{\mathrm{IN}})$, Δ is a bottom-open cell bounded by two vertical rays. Hence, the level of q in Γ_Δ is also its level in Γ. Therefore, q has level in Γ at most Ck.

For each segment $s \in S'_\Delta$, by definition, s is a top edge of a cell $\Delta \in \mathrm{VD}(Q_{\mathrm{IN}}, S_{\mathrm{IN}})$. According to our algorithm, s intersects at most Ck arcs of Γ_Δ. Since $s \subseteq \Delta$, any arc of Γ intersecting s must intersect Δ and thus is in Γ_Δ. Therefore, s intersects at most Ck arcs of Γ. In addition, according to our algorithm, the endpoints of s are in Q'_Δ.

To show that $(Q_{\mathrm{OUT}}, S_{\mathrm{OUT}})$ is a (k, Ck)-shallow cutting, it remains to prove that $\mathrm{VD}(Q_{\mathrm{OUT}}, S_{\mathrm{OUT}})$ covers $L_{\le k}(\Gamma)$. Consider a point $p \in L_{\le k}(\Gamma)$. By definition, $\mathrm{VD}(Q_{\mathrm{IN}}, S_{\mathrm{IN}})$ covers $L_{\le Bk}(\Gamma)$. By setting $B > 1$, $\mathrm{VD}(Q_{\mathrm{IN}}, S_{\mathrm{IN}})$ covers $L_{\le k}(\Gamma)$ and thus p must be in a cell Δ of $\mathrm{VD}(Q_{\mathrm{IN}}, S_{\mathrm{IN}})$. In the following, we argue that p is covered by $\mathrm{VD}(Q'_\Delta \cup S'_\Delta)$, which will prove that $\mathrm{VD}(Q_{\mathrm{OUT}}, S_{\mathrm{OUT}})$ covers $L_{\le k}(\Gamma)$.

Let σ be the cell of the cutting Ξ_Δ that contains p. Since p has level in Γ_Δ at most k and the number of arcs of Γ_Δ intersecting σ is at most $\epsilon \cdot |\Gamma_\Delta|$, every vertex of σ has level at most $k + \epsilon \cdot |\Gamma_\Delta|$. Because the size of the conflict list of each point of Q_{IN} is at most CBk, $|\Gamma_\Delta| \leq 3CBk$ by Observation 14. Therefore, $k + \epsilon \cdot |\Gamma_\Delta| \leq 2k$ by setting the constant $\epsilon = 1/(3CB)$. Hence, all vertices of σ are in $L_{\leq 2k}(\Gamma_\Delta)$ and thus σ is covered by $\mathrm{VD}(Q'_\Delta \cup S'_\Delta)$ according to our algorithm. As $p \in \sigma$, we obtain that p is covered by $\mathrm{VD}(Q'_\Delta \cup S'_\Delta)$.

Bounding the size of VD($Q_{\mathrm{OUT}}, S_{\mathrm{OUT}}$), i.e., $|Q_{\mathrm{OUT}}|$. We now prove $|Q_{\mathrm{OUT}}| \leq C'n/k$. To this end, we compare it against a $(5k, 5c_0k)$-shallow cutting (Q^*, S^*) of size at most $c'_0 n/(5k)$ for some constant c'_0, whose existence is guaranteed by Corollary 17. The details can be found in the full paper. This proves the theorem. ◀

▶ **Corollary 20.** *There exist constants B, C, and C', such that for any parameter $k \in [1, n]$, we can compute a $(B^i k, CB^i k)$-shallow cutting in the sorted vertex-segment form of size at most $C' \frac{n}{B^i k}$, along with its conflict lists, for all $i = 0, 1, \ldots, \log_B \frac{n}{k}$ in $O(n \log \frac{n}{k})$ total time. In particular, we can compute a (k, Ck)-shallow cutting of size $O(n/k)$ in the sorted vertex-segment form, along with its conflict lists, in $O(n \log \frac{n}{k})$ time.*

Proof. By Theorem 19, the runtime $T(n, k)$ satisfies the recurrence $T(n, k) = T(n, Bk) + O(n)$ with the trivial base case $T(n, n) = O(n)$. The recurrence solves to $T(n, k) = O(n \log \frac{n}{k})$. ◀

Proving Theorem 8. We first apply Corollary 20 to compute the shallow cuttings in the sorted vertex-segment form. Then, we transform them to shallow cuttings in the bottom-open pseudo-trapezoid form by Lemma 16, which can be done in additional $O(n \log \frac{n}{k})$ time (i.e., linear time for each cutting). This proves Theorem 8.

6 Dynamic unit-disk range emptiness queries

Our techniques may be extended to solve other related problems about unit disks. In the following, we demonstrate one exemplary problem: the dynamic unit-disk range emptiness queries (see the full paper for a simple solution to the static problem using our techniques).

For a set P of points in the plane, we wish to maintain P in a dynamic data structure for points insertions and deletions to answer *unit-disk range emptiness queries*: Given a unit disk D, determine whether D contains a point of P, and if so, return such a point. One can solve the problem by using a dynamic nearest neighbor search data structure (i.e., given a query disk D, using a nearest neighbor query we find a point $p \in P$ nearest to the center of D; D contains a point of P if and only if $p \in D$). The current best dynamic nearest neighbor search data structure is given by Chan [11]; with that, we can obtain a data structure of $O(n)$ space in $O(n \log n)$ time that supports $O(\log^2 n)$ amortized insertion time, $O(\log^4 n)$ amortized deletion time, and $O(\log^2 n)$ time for unit-disk range emptiness queries. In the following, using our techniques, we propose a better result.

As in Section 3 for the dynamic reporting problem, we can use Lemma 3 to reduce the problem to the following line-separated problem.

▶ **Problem 2** (Dynamic line-separable unit-disk range emptiness queries). *Given a set Q of m points above a horizontal line ℓ, build a data structure to maintain Q to support the following operations. (1) Insertion: insert a point to Q; (2) deletion: delete a point from Q; (3) unit-disk range emptiness query: given a unit disk D whose center is below ℓ, determine whether D contains a point of Q, and if so, return such a point.*

To solve the line-separable problem, we define the set Γ of arcs using Q in the same way as before. Let D_q be a unit disk with center q below ℓ. Note that $D_q \cap Q \neq \emptyset$ if and only if q is above the lower envelope of Γ. Further, q is above the lower envelope of Γ if and only if the lowest arc of Γ intersecting ℓ_q is below q, where ℓ_q is the vertical line through q. Therefore, our problem reduces to the following *vertical line queries* subject to arcs insertions and deletions for Γ: Given a vertical line ℓ^*, find the lowest arc of Γ that intersects ℓ^*.

To solve the dynamic vertical line query problem among arcs of Γ, we apply Chan's framework [8] for the dynamic vertical line query problem among lines (in the dual plane, a vertical line query is dual to: Finding an extreme point on the convex hull of all dual points along a query direction). To this end, we need the following two components: (1) a dynamic data structure of $O(m)$ space with $O(\log m)$ query time and $m^{O(1)}$ update time; (2) a deletion-only data structure of $O(m)$ space that can be built in $O(m \log m)$ time, supporting $O(\log m)$ query time and $O(\log m)$ amortized deletion time. For (1), we can use our static data structure as discussed above, i.e., whenever there is an update, we simply rebuild the data structure. For (2), Wang and Zhao [30] already provided such a data structure. Using these two components, we can apply exactly the same framework of Chan [8]. Indeed, the framework still works for the arcs of Γ because every arc is x-monotone. With the framework and the above two components, we can obtain a data structure of $O(m)$ space that allows insertions and deletions of arcs of Γ in $O(\log^{1+\epsilon} m)$ amortized update time and answers a vertical line query in $O(\log m)$ time, where m is the size of the current set Γ. Consequently, we can solve Problem 2 with the same time complexities. Finally, with Lemma 3 and our problem reduction, we can have a data structure of $O(n)$ space that allows insertions and deletions of points of P in $O(\log^{1+\epsilon} n)$ amortized time and answers a unit-disk range emptiness query in $O(\log n)$ time, where n is the size of the current set P.

References

1 Peyman Afshani and Timothy M. Chan. Optimal halfspace range reporting in three dimensions. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 180–186, 2009. doi:10.1137/1.9781611973068.21.

2 Pankaj K. Agarwal. Range searching, in *Handbook of Discrete and Computational Geometry,* C.D. Tóth, J. O'Rourke, and J.E. Goodman (eds.), pages 1057–1092. CRC Press, 3rd edition, 2017.

3 Pankaj K. Agarwal. Simplex range searching and its variants: a review. In *A Journey Through Discrete Mathematics*, pages 1–30. Springer, 2017. doi:10.1007/978-3-319-44479-6_1.

4 Jon L. Bentley and Hermann A. Maurer. A note on Euclidean near neighbor searching in the plane. *Information Processing Letters*, 8:133–136, 1979.

5 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry — Algorithms and Applications.* Springer-Verlag, Berlin, 3rd edition, 2008.

6 Sarita de Berg and Frank Staals. Dynamic data structures for k-nearest neighbor queries. *Computational Geometry: Theory and Applications*, 111(101976), 2023. doi:10.1016/j.comgeo.2022.101976.

7 Timothy M. Chan. Random sampling, halfspace range reporting, and construction of $(\leq k)$-levels in three dimensions. *SIAM Journal on Computing*, 20:561–575, 2000. doi:10.1137/S0097539798349188.

8 Timothy M. Chan. Dynamic planar convex hull operations in near-logarithmaic amortized time. *Journal of the ACM*, 48:1–12, 2001. doi:10.1145/363647.363652.

9 Timothy M. Chan. A dynamic data structure for 3-D convex hulls and 2-D nearest neighbor queries. *Journal of the ACM*, 57:16:1–16:15, 2010. doi:10.1145/1706591.1706596.

10 Timothy M. Chan. Three problems about dynamic convex hulls. *International Journal of Computational Geometry and Applications*, 22:341–364, 2012. doi:10.1142/S0218195912600096.

11 Timothy M. Chan. Dynamic geometric data structures via shallow cuttings. *Discrete and Computational Geometry*, 64:1235–1252, 2020. doi:10.1007/s00454-020-00229-5.

12 Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In *Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC)*, pages 24:1–24:13, 2016. doi:10.4230/LIPIcs.ISAAC.2016.24.

13 Timothy M. Chan and Konstantinos Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. *Discrete and Computational Geometry*, 56:866–881, 2016. doi:10.1007/s00454-016-9784-4.

14 Bernard Chazelle. An improved algorithm for the fixed-radius neighbor problem. *Information Processing Letters*, 16:193–198, 1983. doi:10.1016/0020-0190(83)90123-0.

15 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete and Computational Geometry*, 9(2):145–158, 1993. doi:10.1007/BF02189314.

16 Bernard Chazelle, Richard Cole, Franco P. Preparata, and Chee-Keng Yap. New upper bounds for neighbor searching. *Information and Control*, 68:105–124, 1986. doi:10.1016/S0019-9958(86)80030-4.

17 Bernard Chazelle and Herbert Edelsbrunner. Optimal solutions for a class of point retrieval problems. *Journal of Symbolic Computation*, 1:47–56, 1985. doi:10.1016/S0747-7171(85)80028-6.

18 Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1:133–162, 1986. doi:10.1007/BF01840440.

19 Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: II. Applications. *Algorithmica*, 1:163–191, 1986. doi:10.1007/BF01840441.

20 Bernard Chazelle, Leonidas J. Guibas, and D.T. Lee. The power of geometric duality. *BIT*, 25:76–90, 1985. doi:10.1007/BF01934990.

21 Herbert Edelsbrunner, David G. Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29:551–559, 1983. doi:10.1109/TIT.1983.1056714.

22 G.N. Frederickson. An optimal algorithm for selection in a min-heap. *Information and Computation*, 104:197–214, 1993. doi:10.1006/inco.1993.1030.

23 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. *Discrete and Computational Geometry*, 64:838–904, 2020. doi:10.1007/s00454-020-00243-7.

24 Jiří Matoušek. Efficient partition trees. *Discrete and Computational Geometry*, 8(3):315–334, 1992. doi:10.1007/BF02293051.

25 Jiří Matoušek. Reporting points in halfspaces. *Computational Geometry: Theory and Applications*, 2:169–186, 1992. doi:10.1016/0925-7721(92)90006-E.

26 Jiří Matoušek. Geometric range searching. *ACM Computing Survey*, 26:421–461, 1994. doi:10.1145/197405.197408.

27 Edgar A. Ramos. On range reporting, ray shooting and k-level construction. In *Proceedings of the 15th Annual Symposium on Computational Geometry (SoCG)*, pages 390–399, 1999. doi:10.1145/304893.304993.

28 Haitao Wang. Unit-disk range searching and applications. *Journal of Computational Geometry*, 14:343–394, 2023. doi:10.20382/jocg.v14i1a13.

29 Haitao Wang and Jie Xue. Near-optimal algorithms for shortest paths in weighted unit-disk graphs. *Discrete and Computational Geometry*, 64:1141–1166, 2020. doi:10.1007/s00454-020-00219-7.

30 Haitao Wang and Yiming Zhao. Computing the minimum bottleneck moving spanning tree. In *Proceedings of the 47th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 82:1–82:15, 2022. doi:10.4230/LIPIcs.MFCS.2022.82.

31 Haitao Wang and Yiming Zhao. An optimal algorithm for L_1 shortest paths in unit-disk graphs. *Computational Geometry: Theory and Applications*, 110:101960: 1–9, 2023. doi:10.1016/j.comgeo.2022.101960.

32 Haitao Wang and Yiming Zhao. Reverse shortest path problem for unit-disk graphs. *Journal of Computational Geometry*, 14:14–47, 2023. doi:10.20382/jocg.v14i1a2.