Maintaining the Union of Unit Discs Under Insertions with Near-Optimal Overhead

Pankaj K. Agarwal
Department of Computer Science, Duke University, Durham, NC 27708, USA
pankaj@cs.duke.edu

Ravid Cohen
School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel
ravidcohn@gmail.com

Dan Halperin
School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel
danha@post.tau.ac.il

Wolfgang Mulzer
Institut für Informatik, Freie Universität Berlin, 14195 Berlin, Germany
mulzer@inf.fu-berlin.de

Abstract

We present efficient data structures for problems on unit discs and arcs of their boundary in the plane. (i) We give an output-sensitive algorithm for the dynamic maintenance of the union of n unit discs under insertions in $O(k \log^2 n)$ update time and $O(n)$ space, where k is the combinatorial complexity of the structural change in the union due to the insertion of the new disc. (ii) As part of the solution of (i) we devise a fully dynamic data structure for the maintenance of lower envelopes of pseudo-lines, which we believe is of independent interest. The structure has $O(\log^2 n)$ update time and $O(\log n)$ vertical ray shooting query time. To achieve this performance, we devise a new algorithm for finding the intersection between two lower envelopes of pseudo-lines in $O(\log n)$ time, using *tentative* binary search; the lower envelopes are special in that at $x = -\infty$ any pseudo-line contributing to the first envelope lies below every pseudo-line contributing to the second envelope. (iii) We also present a dynamic range searching structure for a set of circular arcs of unit radius (not necessarily on the boundary of the union of the corresponding discs), where the ranges are unit discs, with $O(n \log n)$ preprocessing time, $O(n^{1/2+\varepsilon} + \ell)$ query time and $O(\log^2 n)$ amortized update time, where ℓ is the size of the output and for any $\varepsilon > 0$. The structure requires $O(n)$ storage space.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases lower envelopes, pseudo-lines, unit discs, range search, dynamic algorithms, tentative binary search

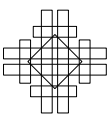
Digital Object Identifier 10.4230/LIPIcs.SoCG.2019.26

Related Version A full version of the paper is available at [3], `http://arxiv.org/abs/1903.10943`.

Funding *Pankaj K. Agarwal*: Work on this paper is supported by NSF under grants CCF-15-13816, CCF-15-46392, and IIS-14-08846, by ARO grant W911NF-15-1-0408, and by grant 2012/229 from the U.S.-Israel Binational Science Foundation.
Ravid Cohen: Work by D.H. and R.C. has been supported in part by the Israel Science Foundation (grant no. 825/15), by the Blavatnik Computer Science Research Fund, by the Blavatnik Interdisciplinary Cyber Research Center at Tel Aviv University, and by grants from Yandex and from Facebook.
Wolfgang Mulzer: Partially supported by ERC STG 757609 and GIF grant 1367/2016.

Acknowledgements We thank Haim Kaplan and Micha Sharir for helpful disucussions.

© Pankaj K. Agarwal, Ravid Cohen, Dan Halperin, and Wolfgang Mulzer;
licensed under Creative Commons License CC-BY
35th International Symposium on Computational Geometry (SoCG 2019).
Editors: Gill Barequet and Yusu Wang; Article No. 26; pp. 26:1–26:15



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Let S be set of n points in $\mathbb{R}^2$, and let U be the union of the unit discs centered at the points of S. We would like to maintain the boundary ∂U of U, as new points are added to S. Even for discs of varying radii, the complexity of ∂U is $O(n)$ [17], and it can be computed in $O(n \log n)$ time using *power diagrams* [6]. An incremental algorithm [20] can maintain ∂U in total of $O(n^2)$ time. This is worst-case optimal, as the overall complexity of the structural changes to ∂U under n insertions may be $\Omega(n^2)$; see an example in [3]. Here, we describe in Section 3 an output-sensitive algorithm that uses $O(n)$ space and updates ∂U in $O(k \log^2 n)$ time per insertion of a disc, where k is the combinatorial complexity of the structural changes to ∂U due to the insertion. Some of our ideas resemble those of de Berg at al. [12], who present a semi-dynamic (insertion only) point-location data structure for U.

The efficient manipulation of collections of unit discs is a widely and frequently studied topic, for example in the context of sensor networks, where every disc represents the area covered by a sensor. Here, we are motivated by multi-agent coverage of a region in search of a target [11], where we investigate the pace of coverage and wish to estimate at each stage the portion of the overall area covered up to a certain point in time. Since the simulation is discretized (i.e., each agents is modeled by a unit disc whose motion is simulated by changing its location at fixed time steps), we can apply the structure above to update the area of the union within the same time bound. We give more details in Section 3.

A set of pseudo-lines in the plane is a set of infinite x-monotone curves each pair of which intersects at exactly one point. Arrangements of pseudo-lines have been intensively studied in discrete and computational geometry; see the recent survey on arrangements [14] for a review of combinatorial bounds and algorithms for arrangements of pseudo-lines. At the heart of our solution to the dynamic maintenance of U lies an efficient data structure for the following problem: Given n pseudo-lines in the plane, dynamically maintain their lower envelope such that one can efficiently answer vertical ray shooting queries from $y = -\infty$. Here, the dynamization allows insertions and deletions. For the case of lines (rather than pseudo-lines), there are several efficient data structures to choose from [8, 9, 18, 7, 16]; these are, however, not directly applicable for pseudo-lines. Also, there are powerful general structures based on shallow cuttings [5, 10, 15]. These structures can handle general families of algebraic curves of bounded description complexity and typically also work in $\mathbb{R}^3$. However, the additional flexibility comes at a cost: the algorithms are quite involved, the performance guarantees are in the expected and amortized sense, and the operations have (comparatively) large polylogarithmic running times. For pseudo-lines, Chan's method [10], with improvements by Kaplan et al. [15], yields $O(\log^3 n)$ amortized expected insertion time, $O(\log^5 n)$ amortized expected deletion time, and $O(\log^2 n)$ worst-case query time. The solution that we propose here is, however, considerably simpler and more efficient: We devise a fully dynamic data structure with $O(\log^2 n)$ worst-case update-time, $O(\log n)$ worst-case ray-shooting query-time, and $O(n)$ space. Additionally, we describe how to find all pseudo-lines below a given query point in $O(\log n + k \log^2 n)$ time, where k is the output size. The structure is an adaptation of the Overmars-van Leeuwen structure [18], matching the performance of the original structure for the case of lines. The key innovation is a new algorithm for finding the intersection between two lower envelopes of planar pseudo-lines in $O(\log n)$ time, using *tentative* binary search (where each pseudo-line in one envelope is "smaller" than every pseudo-line in the other envelope, in a sense to be made precise below). To the best of our knowledge this is the most efficient data structure for the case of pseudo-lines to date.

For our solution to the union-maintenance problem, we need to answer intersection-searching queries of the form: Given the collection $\mathcal{C}$ of unit-radius circular arcs that comprise ∂U and a query unit disc D, report the arcs in $\mathcal{C}$ intersecting D. This problem is a special case of the intersection searching problem in which we wish to preprocess a set of geometric objects into a data structure so that the set of objects intersected by a query object can be reported efficiently. Intersection-searching queries are typically answered using multi-level partition trees; see the recent survey [1] for a comprehensive review. Our final result is a data structure for the intersection-searching problem in which the input objects are arbitrary unit-radius circular arcs rather than arcs forming the boundary of the union of the unit discs, and the query is a unit disc. We present a linear-size data structure with $O(n \log n)$ preprocessing time, $O(n^{1/2+\delta} + \ell)$ query time and $O(\log^2 n)$ amortized update time, where ℓ is the size of the output and $\delta > 0$ is a small constant. Note that because of lack of space, many proofs are omitted from this version and can be found in [3].

2 Dynamic lower envelope for pseudo-lines

We describe a data structure to dynamically maintain the lower envelope of an arrangement of planar pseudo-lines under insertions and deletions. Even though we present our data structure for pseudo-lines, it holds for more general classes of planar curves; see below.

2.1 Preliminaries

Let E be a planar family of pseudo-lines, and let ℓ be a vertical line strictly to the left of the first intersection point in E. The line ℓ defines a total order $\leq$ on the pseudo-lines in E, namely for $e_1, e_2 \in E$, we have $e_1 \leq e_2$ if and only if e_1 intersects ℓ below e_2. Since each pair of pseudo-lines in E crosses exactly once, it follows that if we consider a vertical line ℓ' strictly to the right of the last intersection point in E, the order of the intersection points between ℓ' and E, from bottom to top, is exactly reversed.

The *lower envelope* $\mathcal{L}(E)$ of E is the x-monotone curve obtained by taking the pointwise minimum of the pseudo-lines in E. Combinatorially, the lower envelope $\mathcal{L}(E)$ is a sequence of connected segments of the pseudo-lines in E, where the first and last segment are unbounded. Two properties are crucial for our data structure: (A) every pseudo-line contributes at most one segment to $\mathcal{L}(E)$; and (B) the order of these segments corresponds exactly to the order $\leq$ on E defined above. In fact, our data structure works for every set of planar curves with properties (A) and (B) (with an appropriate order $\leq$), even if they are not pseudo-lines in the strict sense; this fact will prove useful in Section 3 below.

We assume a computational model in which primitive operations on pseudo-lines, such as computing the intersection point of two pseudo-lines or determining the intersection point of a pseudo-line with a vertical line can be performed in constant time.

2.2 Data structure and operations

The tree structure. Our primary data structure is a balanced binary search tree Ξ. Such a tree data structure supports insert and delete, each in $O(\log n)$ time. The leaves of Ξ contain the pseudo-lines, from left to right in the sorted order defined above. An internal node $v \in \Xi$ represents the lower envelope of the pseudo-lines in its subtree. More precisely, every leaf v of Ξ stores a single pseudo-line $e_v \in E$. For an inner node v of Ξ, we write $E(v)$ for the set of pseudo-lines in the subtree rooted at v. We denote the lower envelope of $E(v)$ by $\mathcal{L}(v)$. The inner node v has the following variables:

- f, ℓ, r: a pointer to the parent, left child and right child of v, respectively;
- max: the *last* pseudo-line in E(V) (last in the ordering defined in Section 2.1)
- Λ: a balanced binary search tree that stores the prefix or suffix of $\mathcal{L}(v)$ that is not on the lower envelope $\mathcal{L}(f)$ of the parent (in the root, we store the lower envelope of E). The leaves of Λ store the pseudo-lines that support the segments on the lower envelope, with the endpoints of the segments, sorted from left to right. An inner node of Λ stores the common point of the last segment in the left subtree and the first segment in the right subtree. We will need split and join operations on the binary trees, which can be implemented in $O(\log n)$ time.

Queries. We now describe the query operations available on our data structure. In a *vertical ray-shooting query*, we are given a value $x_0 \in \mathbb{R}$, and we would like to find the pseudo-line $e \in E$ where the vertical line $\ell : x = x_0$ intersects $\mathcal{L}(E)$. Since the root of Ξ explicitly stores $\mathcal{L}(E)$ in a balanced binary search tree, this query can be answered easily in $O(\log n)$ time.

▶ **Lemma 1.** *Let $\ell : x = x_0$ be a vertical ray shooting query. We can find the pseudo-line(s) where ℓ intersects $\mathcal{L}(E)$ in $O(\log n)$ time.*

▶ **Lemma 2.** *Let $q \in \mathbb{R}^2$. We can report all pseudo-lines in E that lie below $q \in \mathbb{R}^2$ in total time $O(\log n + k \log^2 n)$, where k is the output size*

Update. To insert or delete a pseudo-line e in Ξ, we follow the method of Overmars and van Leeuwen [18]. We delete or insert a leaf for e in Ξ using standard binary search tree techniques (the $v.\max$ pointers guide the search in Ξ). As we go down, we construct the lower envelopes for the nodes hanging off the search path, using split and join operations on the $v.\Lambda$ trees. Going back up, we recompute the information $v.\Lambda$ and $v.\max$. To update the $v.\Lambda$ trees, we need the following operation: given two lower envelopes $\mathcal{L}_\ell$ and $\mathcal{L}_r$, such that all pseudo-lines in $\mathcal{L}_\ell$ are smaller than all pseudo-lines in $\mathcal{L}_r$, compute the intersection point q of $\mathcal{L}_\ell$ and $\mathcal{L}_r$. In the next section, we see how to do this in $O(\log n)$ time, where n is the size of E. Since there are $O(\log n)$ nodes in Ξ affected by an update, this procedure takes $O(\log^2 n)$ time. More details can be found in [18, 19].

▶ **Lemma 3.** *It takes $O(\log^2 n)$ to insert or delete a pseudo-line in Ξ.*

2.3 Finding the intersection point of two lower envelopes

Given two lower envelopes $\mathcal{L}_\ell$ and $\mathcal{L}_r$ such that all pseudo-lines in $\mathcal{L}_\ell$ are smaller than all pseudo-lines in $\mathcal{L}_r$, we would like to find the intersection point q between $\mathcal{L}_\ell$ and $\mathcal{L}_r$ in $O(\log n)$ time. We assume that $\mathcal{L}_\ell$ and $\mathcal{L}_r$ are represented as balanced binary search trees. The leaves of $\mathcal{L}_\ell$ and $\mathcal{L}_r$ store the pseudo-line segments on the lower envelopes, sorted from left to right. We assume that the pseudo-line segments in the leaves are half-open, containing their right, but not their left endpoint in $\mathcal{L}_\ell$; and their left, but not their right endpoint in $\mathcal{L}_r$.[1] Thus, it is uniquely determined which leaves of $\mathcal{L}_\ell$ and $\mathcal{L}_r$ contain the intersection point q. A leaf v stores the pseudo-line $\mathcal{L}(v)$ that supports the segment for v, as well as an endpoint $v.p$ of the segment, namely the left endpoint if v is a leaf of $\mathcal{L}_\ell$, and the right

[1] We actually store both endpoints in the trees, but the intersection algorithm uses only one of them, depending on the role the tree plays in the algorithm.

endpoint if v is a leaf of $\mathcal{L}_r$.[2] An inner node v stores the intersection point $v.p$ between the last pseudo-line in the left subtree $v.\ell$ of v and the first pseudo-line in the right subtree $v.r$ of v, together with the lower envelope $\mathcal{L}(v)$ of these two pseudo-lines. These trees can be obtained by appropriate split and join operations from the Λ trees stored in Ξ.

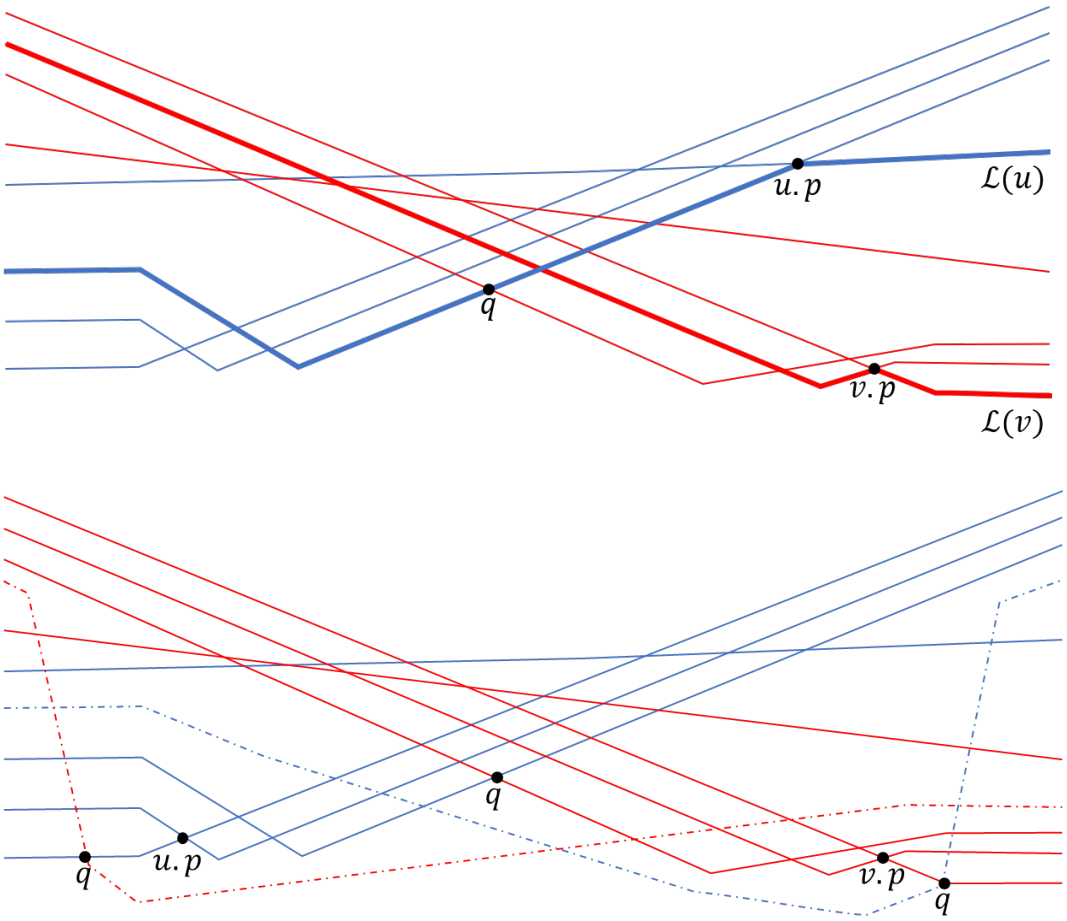


Figure 1 (On the top). An example of case 1. Case 2 is symmetric. (On the bottom). An example of Case 3. $\mathcal{L}_\ell$ is blue; $\mathcal{L}_r$ is red. The solid pseudo-lines are fixed. The dashed pseudo-lines are optional, namely, either none of the dashed pseudo-lines exists or exactly one of them exists. $u.p$ and $v.p$ are the current points; and Case 3 applies. Irrespective of the local situation at u and v, the intersection point can be to the left of $u.p$, between $u.p$ and $v.p$, or to the right of $v.p$, depending on which one of the dashed pseudo-lines exists.

Let $u^* \in \mathcal{L}_\ell$ and $v^* \in \mathcal{L}_r$ be the leaves whose segments contain q. Let π_ℓ be the path in $\mathcal{L}_\ell$ from the root to u^* and π_r the path in $\mathcal{L}_r$ from the root to v^*. Our strategy is as follows: we simultaneously descend into $\mathcal{L}_\ell$ and into $\mathcal{L}_r$. Let u be the current node in $\mathcal{L}_\ell$ and v the current node in $\mathcal{L}_r$. In each step, we perform a local test on u and v to decide how to proceed. There are three possible outcomes:

[2] If the segment is unbounded, the endpoint might not exist. In this case, we use a symbolic endpoint at infinity that lies below every other pseudo-line.

1. $u.p$ is on or above $\mathcal{L}(v)$: the intersection point q is equal to or to the left of $u.p$. If u is an inner node, then u^* cannot lie in $u.r$; if u is a leaf, then u^* lies strictly to the left of u;
2. $v.p$ lies on or above $\mathcal{L}(u)$: the intersection point q is equal to or to the right of $v.p$. If v is an inner node, then v^* cannot lie in $v.\ell$; if v is a leaf, then v^* lies strictly to the right of v;
3. $u.p$ lies below $\mathcal{L}(v)$ and $v.p$ lies below $\mathcal{L}(u)$: then, $u.p$ lies strictly to the left of $v.p$ (since we are dealing with pseudo-lines). It must be the case that $u.p$ is strictly to the left of q or $v.p$ is strictly to the right of q (or both). In the former case, if u is an inner node, u^* lies in or to the right of $u.r$ and if u is a leaf, then u^* is u or a leaf to the right of u. In the latter case, if v is an inner node, v^* lies in or to the left of $v.\ell$ and if v is a leaf, then v^* is v or a leaf to the left of v; see Figure 1.

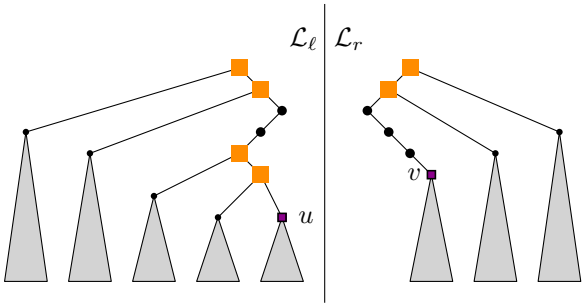


Figure 2 The invariant: the current search nodes are u and v. `uStack` contains all nodes on the path from the root to u where the path goes to a right child (orange squares), `vStack` contains all nodes from the root to v where the path goes to a left child (orange squares). The final leaves u^* and v^* are in one of the gray subtrees; and at least one of them is under u or under v.

Although it is clear how to proceed in the first two cases, it is not immediately obvious how to proceed in the third case, because the correct step might be either to go to $u.r$ or to $v.\ell$. In the case of lines, Overmars and van Leeuwen can solve this ambiguity by comparing the slopes of the relevant lines. For pseudo-lines, however, this does not seem to be possible. For an example, refer to Figure 1, where the local situation at u and v does not determine the position of the intersection point q. Therefore, we present an alternative strategy.

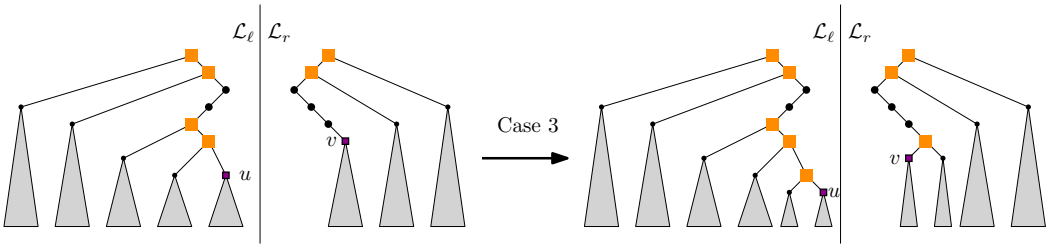


Figure 3 Comparing u to v: in Case 3, we know that u^* is in $u.r$ or v^* is in $v.\ell$; we go to $u.r$ and to $v.\ell$.

We will maintain the invariant that the subtree at u contains u^* or the subtree at v contains v^* (or both). In Case 3, u^* must be in $u.r$, or v^* must be in $v.\ell$; see Figure 3. We move u to $u.r$ and v to $v.\ell$. One of these moves must be correct, but the other move might be mistaken: we might have gone to $u.r$ even though u^* is in $u.\ell$ or to $v.\ell$ even though v^* is in $v.r$. To correct this, we remember the current u in a stack `uStack` and the current v in a stack `vStack`, so that we can revisit $u.\ell$ or $v.r$ if it becomes necessary. This leads to the

general situation shown in Figure 2: u^* is below u or in a left subtree of a node on `uStack`, and v^* is below v or in a right subtree of a node on `vStack`, and at least one of u^* or v^* must be below u or v, respectively. Now, if Case 1 occurs when comparing u to v, we can exclude the possibility that u^* is in $u.r$. Thus, u^* might be in $u.\ell$, or in the left subtree of a node in `uStack`; see Figure 4. To make progress, we now compare u', the top of `uStack`, with v. Again, one of the three cases occurs. In Case 1, we can deduce that going to $u'.r$ was mistaken, and we move u to $u'.\ell$, while v does not move. In the other cases, we cannot rule out that u^* is to the right of u', and we move u to $u.\ell$, keeping the invariant that u^* is either below u or in the left subtree of a node on `uStack`. However, to ensure that the search progresses, we now must also move v. In Case 2, we can rule out $v.\ell$, and we move v to $v.r$. In Case 3, we move v to $v.\ell$. In this way, we keep the invariant and always make progress: in each step, we either discover a new node on the correct search paths, or we pop one erroneous move from one of the two stacks. Since the total length of the correct search paths is $O(\log n)$, and since we push an element onto the stack only when discovering a new correct node, the total search time is $O(\log n)$; see an example run in [3]. For the full pseudocode and the formal proof see [3].

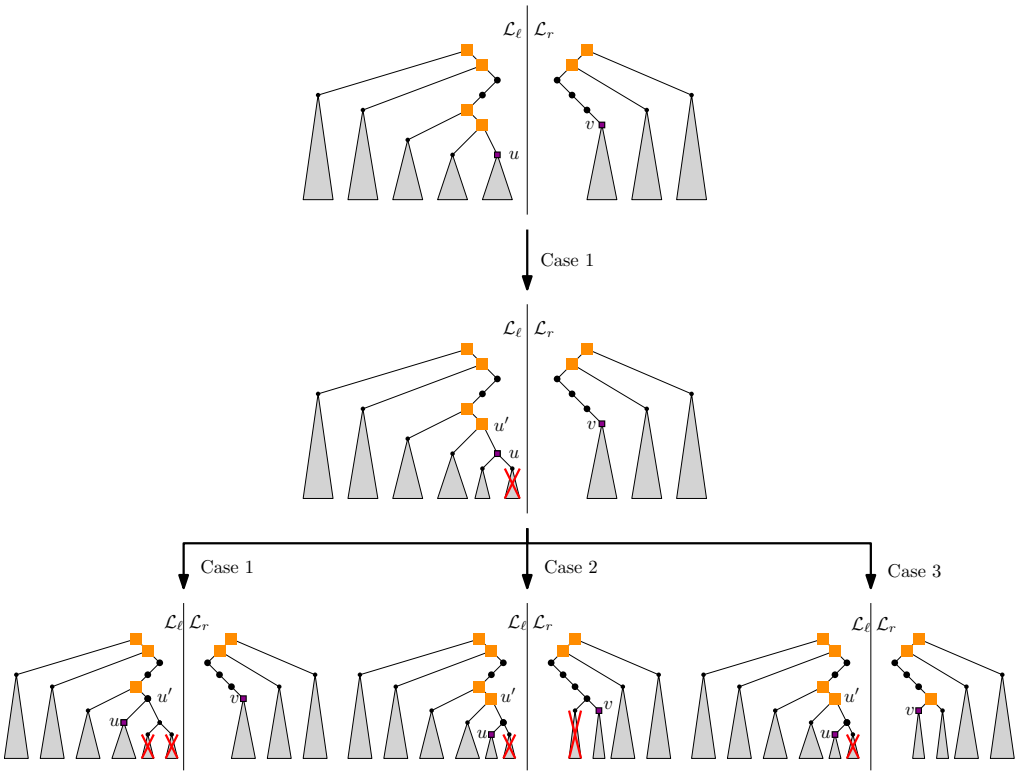


Figure 4 Comparing u to v: in Case 1, we know that u^* cannot be in $u.r$. We compare u' and v to decide how to proceed: in Case 1, we know that u^* cannot be in $u'.r$; we go to $u'.\ell$; in Case 2, we know that u^* cannot be in $u.r$ and that v^* cannot be in $v.\ell$; we go to $u.\ell$ and to $v.r$; in Case 3, we know that u^* is in $u'.r$ (and hence in $u.\ell$) or in $v.\ell$; we go to $u.\ell$ and to $v.\ell$.

3 Maintaining the union of unit discs under insertions

To maintain the union of unit discs under insertions, we maintain dynamic data structures for representing the boundary of the union, for reporting the arcs of the boundary that intersect with the next disc to be inserted, and for updating the boundary representation due to the insertion of the new disc. This section is dedicated to these data structures.

Overview of the algorithm. We denote by $D(x)$ the unit disc centered at x. Let U be the union of n unit discs and let $D(x)$ be the new unit disc, which we wish to insert. In order to report the arcs of ∂U that intersect $D(x)$, we overlay the plane with an implicit grid, where only cells that intersect with U are stored, and where the size of the diagonal of a grid cell is 1. The arcs of ∂U are divided into the cells of the grid – each arc of ∂U is associated with the cell that contains it. Note that if an arc belongs to more than one cell then we split it into (sub)arcs at the boundaries of the cells that it crosses (see an illustration of the structure in [3].We divide the arcs of a given cell into four sets: *top*, *right*, *bottom* and *left*, which we denote by E_t, E_r, E_b and E_l respectively (see Section 3.1). The algorithm consists of the following main steps: (1) Find the cells that $D(x)$ intersects. (2) For each such cell find the arcs of each one of the sets E_t, E_r, E_b and E_l that $D(x)$ intersects. Cells of the union that contain no boundary arcs are treated in a special way. (3) Update ∂U using the arcs we found in the previous step and with $\partial D(x)$.

Step 1 of the algorithm is implemented using a balanced binary tree Ω on the *active cells*, namely cells that have non-empty intersection with the current union U. The key of each active cell is the pair of coordinates of its bottom left corner. The active cells are stored at the leaves of the tree in ascending lexicographic order. Finding the cells intersected by a new disc, inserting or deleting a cell, take $O(\log n)$ time each. For details, see, e.g., [13]. As we will see below, the structure Ω will also be used to decide whether a new disc is fully contained in the current union or lies completely outside the current union (Section 3.3).

Most of this section is dedicated to a description of Steps 2 and 3 of the algorithm for the set E_t. The sets E_r, E_b, and E_l can be handled in a similar manner. The basic property that we use is that $D(x)$ intersects an arc e if and only if x belongs to $e \oplus D_1$, namely the Minkowski sum of e with a unit disc.

We split the boundaries of the Minkowski sums of E_t into upper and lower curves at the x-extremal points; in what follows, we refer to them as upper and lower curves, and denote their respective sets by Γ^+ and Γ^-. (For clarity, we will refer to portions of the boundary of the union as *arcs* and to portions of the boundary of the Minkowski sums as *curves*.) The disc $D(x)$ intersects the arc $e \in E_t$ if and only if x lies above the lower curve induced by e and below the upper curve induce by e. We will store the curves of Γ^+ in a dynamic structure Δ^+ and the curves of Γ^- in a dynamic structure Δ^- (both described in Section 3.2).

Another property that we use is the following (see Lemma 12 below): Let ℓ be a vertical line that passes through x, the center of the new disc. Then the intersection points of curves in Γ^+ with ℓ are all above the intersection points of curves of Γ^- with ℓ.

Assume for the sake of exposition that we are given the point ξ of intersection between ℓ and the upper envelope of the curves in Γ^-. If the center x of our new disc is above ξ then, since x is above all the lower curves that cross ℓ we only need to search the structure Δ^+ for the upper curves that lie above x – these will determine the arcs of E_t that are intersected by $D(x)$. If the point x coincides with or lies below ξ then we only need to search the structure Δ^- for the lower curves that lie below x – now these will determine the arcs of E_t that are intersected by $D(x)$.

However, we cannot easily obtain the point ξ, and hence querying the data structures is a little more involved: We use Δ^+ to iterate over the upper curves that lie above x. For every upper curve we check in $O(1)$ time whether its corresponding arc (of E_t) intersects with $D(x)$. If it intersects then we add this arc to the output list and continue to the next upper curve. If all the upper curves above x turn out to be induced by arcs intersecting $D(x)$ we output this list of arcs and stop.

If all the reported arcs from the query of Δ^+ indeed intersect $D(x)$, then we are guaranteed that x is above ξ and this is the complete answer. Due to Lemma 12, if we detect that the arc induced by a curve reported by Δ^+ to lie above x is not intersecting $D(x)$, then we are guaranteed that x is on or below ξ and we will obtain the full reply by querying Δ^-.

We review the geometric foundations needed by our algorithms and data structures in Section 3.1, then describe the data structures in Section 3.2. Finally, in Section 3.3 we explain how we solve our motivating problem – dynamically reporting the area of the union.

3.1 Preliminaries

Let B be an axis-parallel square, which represents one grid cell with unit-length diagonal, and let ℓ_1 and ℓ_2 be lines that support the diagonals of B. These lines divide the plane into top, right, bottom and left quadrants, which we denote by Q_t, Q_r, Q_b and Q_l, respectively.

Let U be the union of n unit discs. We divide the arcs of ∂U that are contained in B into four sets according to the quadrant which contains their centers. In case that a center lies on one of the lines then it is added either to the top or to the bottom quadrant. Denote these four sets of arcs by E_t, E_r, E_b and E_l. The power of this subdivision into quadrants is that now the projections of the arcs in any one set onto a major axis (the x-axis for E_t or E_b, and the y-axis for E_l or E_r), are pairwise interior disjoint. For example, E_t contains the arcs whose centers are located in Q_t, and the projections of the arcs in E_t onto the x-axis are pairwise interior disjoint, as we show below in Lemma 6.

▶ **Definition 4.** *For two bounded x-monotone arcs e_i and e_j we write $e_i \leq_x e_j$ if and only if the right endpoint of e_i is to the left of or coincides with the left endpoint of e_j.*

▶ **Lemma 5.** *Each arc in E_t is portion of a lower semicircle.*

▶ **Lemma 6.** *The x-projections of the (relative interiors of) arcs in E_t are pairwise disjoint.*

Relying on Lemma 6, henceforth we assume that the arcs in E_t are ordered from left to right: $e_1, \ldots, e_m$. We wish to find which arcs of the set E_t intersect with the new unit disc $D(x)$ to be inserted. For this purpose, we compute the Minkowski sum of each arc e_i of E_t with a unit disc centred at the origin. Then, we divide the boundary of each Minkowski sum into upper and lower curves at the x-extremal points: denote the top curve by γ_i^+ and the bottom curve by γ_i^-. We denote the set of the upper curves, $\{\gamma_i^+ | e_i \in E_t\}$, by Γ^+ and the set of the lower curves, $\{\gamma_i^- | e_i \in E_t\}$, by Γ^-. In the rest of this section we prove some useful properties regrading the curves in Γ^+ and Γ^-:

P1 Every lower curve in Γ^- can appear at most once on the lower envelope of the curves in Γ^-. Furthermore, if γ_i^- and γ_j^- appear on the lower envelope then γ_i^- appears to the left of γ_j^- if and only if $e_i <_x e_j$.

P2 Let e_i, e_{i+1} and e_{i+2} be an ordered sequence of arcs in E_t and q be a point. If q lies below γ_i^+ and γ_{i+2}^+ then q lies also below γ_{i+1}^+.

P3 For every vertical line ℓ. The intersection points of the lower curves with ℓ are below the intersection points of the upper curves with ℓ.

In order to prove Property **P1**, we first need to show that every pair of lower curves intersect at most once.

▶ **Lemma 7.** *Let e_i and e_j be arcs of E_t. Then γ_i^- and γ_j^- intersect in exactly one point.*

For two x-monotone curves ℓ_1, ℓ_2 that intersect exactly once, we say that $\ell_1 < \ell_2$ when ℓ_1 appears on their joint lower envelope immediately to the left of their intersection point and $\ell_2 < \ell_1$ otherwise. The proof of Lemma 7 also implies,

▶ **Corollary 8.** *For any pair of curves $\gamma_i^-, \gamma_j^- \in \Gamma^-$, $\gamma_i^- < \gamma_j^-$ if and only if $e_i <_x e_j$.*

We now turn to discuss the upper curves. To prove Property P2 (Lemma 10), we first consider the structure of the upper envelope of the upper curves.

▶ **Observation 9.** *Let p and q be the endpoints of the arc e_i in E_t. The upper curve γ_i^+ is the upper envelope of the upper boundaries (namely, semicircles) of the discs $D(p)$ and $D(q)$*

▶ **Lemma 10.** *Let e_i, e_{i+1} and e_{i+2} be an ordered sequence of arcs in E_t and q be a point. If q is below γ_i^+ and γ_{i+2}^+ then q is also below γ_{i+1}^+.*

Proof. Let p_1, p_2, p_3 be points on arcs that belong to E_t with $p_1 <_x p_2 <_x p_3$. Let σ_1^+, σ_2^+ and σ_3^+ be the upper semicircles of $\partial D(p_1)$, $\partial D(p_2)$ and $\partial D(p_3)$, respectively. Let p_{12}^+ and p_{23}^+ be the intersection points of $\sigma_1^+ \cap \sigma_2^+$ and $\sigma_2^+ \cap \sigma_3^+$, respectively. These intersection points exist, since the distance between every pair of points in B is at most one. By the assumption, $p_1 <_x p_2$, which means that σ_1^+ appears to the left of σ_2^+ on the upper envelope of σ_1^+ and σ_2^+. Let c be the center of the arc e of E_t on which p_2 lies. The point c is on σ_2^+, since p_2 belongs to a lower semicircle of radius 1. In addition, c is not below σ_1^+ since otherwise $p_1 \in D(c)$ which would contradict that p_1 is a point on an arc in E_t. This means that $p_{12}^+ \leq_x c$. The same argument implies that $p_{23}^+ \geq_x c$ and therefore $p_{12}^+ \leq_x p_{23}^+$. This in turn implies that the intersection point, p_{13}^+, between σ_1^+ and σ_3^+ is below or on σ_2^+ and therefore every point that lies below σ_1^+ and σ_3^+ lies below σ_2^+. The only condition on p_1, p_2 and p_3 is that they will be x-ordered. $e_i \leq_x e_{i+1} \leq_x e_{i+2}$, so the claim holds (see Figure 5). ◀

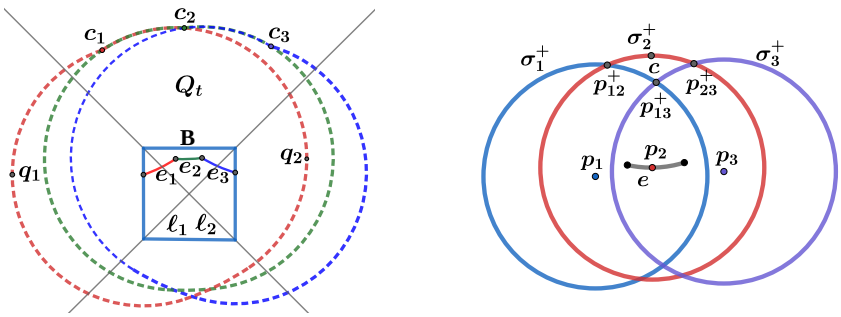


Figure 5 (On the left). An example of $\partial U \cap B$. e_1, e_2 and e_3 are the arcs of E_t whose centers are c_1, c_2 and c_3, respectively. The red, green and blue outer shapes are the boundary of the Minkowski sums of each of e_1, e_2 and e_3 with a disc of radius 1, respectively. γ_1^+ and γ_1^- are denoted by the upper and lower red curves whose endpoints are q_1 and q_2, respectively. (On the right) Illustration of the proof of Lemma 10.

For p an endpoint of $e_i \in E_t$, we call the upper semi-circle of the disc $D(p)$ *the upper curve* of p. We denote the upper envelope of the curves in Γ^+ by $\mathcal{U}(\Gamma^+)$. Note that some of the upper curves may appear on $\mathcal{U}(\Gamma^+)$ as a single point, namely, they coincide with one of the breakpoints of $\mathcal{U}(\Gamma^+)$. The following corollary stems from the proof of Lemma 10.

▶ **Corollary 11.** *(i) The upper curve of each endpoint of every arc of E_t appears on $\mathcal{U}(\Gamma^+)$. (ii) The x-order of the curves on $\mathcal{U}(\Gamma^+)$ corresponds to the x-order of the endpoints of the arcs of E_t.*

Next, we prove that for any pair of arcs $e_i, e_j \in E_t$, γ_i^+ and γ_j^- are disjoint. Furthermore, we show that γ_i^+ is above γ_j^-, and by that prove Property P3.

▶ **Lemma 12.** *Let e_i and e_j be two distinct arcs in E_t and let ℓ be a vertical line. If ℓ intersects with γ_i^+ and γ_j^- at p and q, respectively, then $p >_y q$.*

3.2 Data structures

In this section we describe two data structures. The data structure Δ^+ (resp. Δ^-), dynamically maintains the set Γ^+ of the upper curves (resp. Γ^- of lower curves). The purpose of these structures is to efficiently answer the following queries: given a point x, report on the upper (resp. lower) curves which are above (resp. below) x. For the structure Δ^+ it is required that we get the answer gradually, one curve after the other, since we need to test each curve for being relevant (in addition to being above x), and stop as soon as we detect the first irrelevant curve.

3.2.1 Dynamically maintaining the lower curves

For maintaining the lower curves Γ^- induced by the arcs in E_t, we implement Δ^- using the data structure described in Section 2. Recall that the data structure dynamically maintains a set of curves fulfilling property P1 and supports the following **query**: given a point x report the curves in Γ^- that are below x.

Update. After we insert a new unit disc we may have to delete and insert many lower curves. If a lower curve γ_i^- is split into subcurves, then we delete γ_i^- and create two new subcurves instead. In order for Property P1 to hold at all times, we first delete the old lower curves from Δ^- and then insert the new ones.

3.2.2 Dynamically maintaining the upper curves

Description. Let $p_1, p_2, \ldots, p_r$ be the endpoints of the arcs of E_t sorted in ascending x-order. Recall that $\mathcal{U}(\Gamma^+)$ denotes the upper envelope of Γ^+. Let $s_1, s_2, \ldots, s_r$ be the arcs of $\mathcal{U}(\Gamma^+)$ ordered from left to right. Note that each endpoint of E_t corresponds to an arc in $\mathcal{U}(\Gamma^+)$, i.e., p_i corresponds to the curve s_i. The data structure Δ^+ is a balanced binary search tree. We store the points p_i in the leaves of the tree in their left-to-right order. We also store in each leaf pointers `rn` and `ln` to its right and left neighboring leaves respectively, if exist. Each internal node stores a pointer `lml` to the leftmost leaf of its right subtree. To keep the a structure simple, if two arcs of E_t meet at a single point, we keep only one of the endpoints incident to this point in the list $\{p_i\}$. However, we mark in the leaf of p_i which are the two arcs incident to it. Below, when we traverse the leaves of the tree and test the respective arcs of E_t for intersection with the new disc, in some nodes we may need to test two arcs.

Query. Let q be a query point. By following a path from the root, we first find the leaf v such that the vertical line through p intersects the edge s_v. The search down the tree is carried out as follows. Suppose we reached an internal node u. We use the pointer $\texttt{lml}(u)$ to obtain the leaf w, and use $\texttt{ln}(w)$ to find the point immediately to its left in the sequence

$\{p_i\}$. These two points will determine the breakpoint of $\mathcal{U}(\Gamma^+)$ that separates between the left and right portions of the envelope, which are represented by the subtrees rooted at the left and right children of u.

Recall that the structure Δ^+ plays the extra role of deciding whether the center x of the new disc lies above the point ξ or not (see the overview the algorithm in the beginning of Section 3). Therefore the query process is somewhat more involved than if we used the structure only to determine which curves of Γ^+ pass above x.

Once we find the point p_i whose arc s_i of the envelope intersects the vertical line through the query point q, we will be traversing leaves of Δ^+ starting at v going both rightward and leftward. At each leaf u we test whether q lies below the curve s_j stored at u and if so, we check whether $D(x)$ intersects the relevant arc of E_t. If the answer to both predicates is true then we continue the search in the same direction. If while we search rightwards the first predicate is false then we go leftwards starting from v. If the first predicate is false and we search leftwards then we stop the search and report on the arcs that we found. If the first predicate is true and second predicate is false then we continue with Δ^-.

Update. After we insert a new disc, many arcs may be deleted from E_t and many new arcs may be inserted into E_t. We simply remove the endpoints of the deleted arcs and insert the endpoints of the new arcs into Δ^+.

The correctness of the query procedure follows from Lemma 10. The performance of the structure is summarized in the following lemma whose proof is straightforward.

▶ **Lemma 13.** *The query time of the data structure is $O(\log n + k)$, where k is the number of reported arcs. The update requires $O(\log n)$ time per operation.*

When querying the data structures Δ^+ and Δ^- we obtain the set I of arcs of the existing union-boundary that need to be deleted or partially cut since they are covered by the new disc $D(x)$ to be inserted. However, we also need to update the structures with the arcs that the boundary of the new disc contributes to the union boundary.

To find which portions of $\partial D(x)$ appear on the boundary of the union $U \cup D(x)$, we construct the arrangement $\mathcal{A}(I \cup \partial D(x))$ and look for faces of this arrangement that abut $\partial D(x)$ and are not in the union U. One can show that in a face f of this type the arcs of ∂U appear on it as concave, meaning that any point inside this face is outside the disc bounded by the arcs. Denote the size of I by k. Assume first that $k \geq 1$. We can construct the arrangement in $O(k \log k)$ time (recall that the arcs in $I \cup \partial D(x)$ are pairwise interior disjoint). Finding the arcs of $\partial D(x)$ that should be inserted takes another $O(k)$ time.

If $k = 0$, there are two cases based on whether $D(x) \cap U$ is (i) $D(x)$ or (ii) the empty set. To distinguish between the cases we need to either (i) find a point that belongs to $D(x)$ and U, or (ii) a point that belongs to $D(x)$ but not to U. Recall that in order to find I we overlay the plane with a grid of cells of unit-length diagonal each. This implies that at least one of the cells, denoted by ω, is fully contained in $D(x)$. If ω is an *active cell*, i.e., $\omega \cap U \neq \emptyset$, then ω is fully contained in U (I is an empty set) and therefore $D(x) \cap U = D(x)$; otherwise $D_1(x) \cap U = \emptyset$. To check whether ω is active, we search for it in the structure Ω. In case (i) we do nothing further, and in case (ii) we make all the grid cells covered by $D(x)$ active, and we update the data structures of each grid cell crossed by $\partial D(x)$ by the relevant portions of $\partial D(x)$. To conclude,

▶ **Theorem 14.** *The boundary arcs of the union of a set of n unit discs can be maintained under insertion in a data structure of $O(n)$ size so that a new disc can be inserted in $O(k \log^2 n)$ time, where k is the total number of changes on the boundary of the union.*

3.3 Maintaining the area of the union

We are now ready to solve our motivating problem, namely dynamically reporting the *area* of the union as we insert discs. At a a high level our algorithm proceeds as follows:

1. Find the set I of the arcs on the boundary of the union U that intersect with the new disc $D(x)$ to be inserted.
2. Compute the arrangement $\mathcal{A}(I \cup \partial D(x))$.
3. Calculate the extra area (over the area of the union before inserting $D(x)$) that $D(x)$ covers, using $\mathcal{A}(I \cup \partial D(x))$.

In order to find I we make use of the data strctures described above and summarized in Theorem 14. Let k denote the number of arcs in I and assume that $k \geq 1$. We use a sweep-line algorithm to compute the arrangement $\mathcal{A}(I \cup \partial D(x))$ in time $O(k \log k)$. To calculate the extra area that $D(x)$ covers, we go over the faces of the arrangement and sum up the area of the faces that are contained in $D(x) \setminus U$. If $k = 0$ then either the disc is fully contained in the current union (see above for how to determine this), in which case we do nothing, or it is disjoint from the union before the insertion of the disc, in which case we increase the area by π. To conclude,

▶ **Theorem 15.** *Given a sequence of n unit discs in $\mathbb{R}^2$ to be inserted one after the other, reporting the area of the union of the discs after each insertion can be carried out in $O(k \log^2 n)$ time and $O(n)$ space, where k is the total number of structural changes to the boundary of the union incurred by the insertion of the new disc.*

4 Intersection-searching of unit arcs with unit disc

In this section we address the following intersection-searching problem: Preprocess a collection $\mathcal{C}$ of circular arcs of unit radius into a data structure so that for a query unit disc $D(x)$, centered at the point x, the arcs in $\mathcal{C}$ intersecting $D(x)$ can be reported efficiently. We assume for simplicity that every arc in $\mathcal{C}$ belong to the lower semicircle.

Let $e \in \mathcal{C}$ be a unit-radius circular arc, and let p_1 and p_2 be its endpoints. A unit disc $D(x)$ intersects e if and only if $e \oplus D(0)$, the Minkowski sum of e with a unit disc, contains the center x. Let $z := D(p_1) \cup D(p_2)$, and let $D^+(c)$ be the disk of radius 2 centered at c; z divides $D^+(c)$ into three regions (see Fig. 6): (i) z^+, the portion of $D^+(c) \setminus z$ above z, (ii) z itself, and (iii) z^-, the portion of $D^+(c) \setminus z$ below z. It can be verified that $e \oplus D(0) = z \cup z^-$. We give an alternate characterization of $z \cup z^-$, which will help in developing the data structure.

Let ℓ be a line that passes through the tangents points, p_1' and p_2', of $D(p_1)$ and $D(p_2)$ with $D^+(c)$, respectively, and let ℓ^- be the halfplane lying below ℓ. Set $L(e) = D^+(c) \cap \ell^-$ (see Fig. 6).

▶ **Lemma 16.** *If $\partial D(p_1)$ and $\partial D(p_2)$ intersect at two points (one of which is always c) then ℓ passes through $q := (\partial D(p_1) \cap \partial D(p_2)) \setminus \{c\}$. Otherwise $c \in \ell$.*

Proof. Assume that q exists. The quadrilateral (c, p_1, q, p_2) is a rhombus since all its edges have length 1. Let α be the angle $\angle p_1 q p_2$ and β be the angle $\angle c p_1 q$. The angle $\angle q p_1 p_1'$ is equal to α since the segment (c, p_1') is a diameter of $D(p_1)$. The angle $\angle p_1 q p_1'$ is equal to $\frac{\beta}{2}$ since $\triangle p_1 q p_1'$ is an isosceles triangle. The same arguments apply to the angle $\angle p_2 q p_2'$ implying that the angle $\angle p_1' q p_2'$ is equal to π.

Assume that q does not exists then the segment (p_1, p_2) is a diameter of $D(c)$. The segment (c, p_1') is a diameter of $D(p_1)$. The segment (p_1, p_2) coincide with (c, p_1') at the segment (c, p_1). The same argument applies to the segment (c, p_2'), implying that the angle $\angle p_1' q p_2'$ is equal to π (see Fig 6). ◀

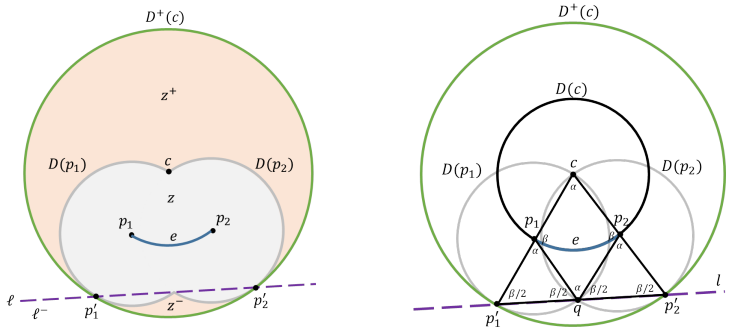


Figure 6 (On the left) Partition of $D_2(c)$ into three regions: z^+, z and z^-. (On the right) Illustration of Lemma 16.

The following corollary summarizes the criteria for the intersection of a unit circular arc with a unit disc.

▶ **Corollary 17.** *Let e be a circular arc in $\mathcal{C}$ with endpoints p_1 and p_2 and center c. Then (i) $z \cup z^- = z \cup L(e)$. (ii) e intersects a unit disc $D(x)$ if and only if at least one of the following conditions is satisfied: (a) $x \in D(p_1)$ (or $p_1 \in D(x)$), (b) $x \in D(p_2)$ (or $p_2 \in D(x)$), and (c) $x \in L(e)$.*

We thus construct three separate data structures. The first data structure preprcesses the left endpoints of arcs in $\mathcal{C}$ for unit-disc range searching, the second data structure preprocesses the right endpoints of arcs in $\mathcal{C}$ for unit-disc range searching, and the third data structure preprocesses $\mathcal{L} = \{L(e) \mid e \in \mathcal{C}\}$ for inverse range searching, i.e., reporting all regions in $\mathcal{L}$ that contain a query point. Using standard circle range searching data structures (see e.g. [4, 2]), we can build these three data structures so that each of them takes $O(n)$ space and answers a query in $O(n^{1/2+\epsilon} + k)$ time, where k is the output size. Furthermore, these data structures can handle insertions/deletions in $O(\log^2 n)$ time. We omit all the details from here and conclude the following:

▶ **Theorem 18.** *Let $\mathcal{C}$ be a set of n unit-circle arcs in $\mathbb{R}^2$. $\mathcal{C}$ can be preprocessed into a data structure of linear size so that for a query unit disk D, all arcs of $\mathcal{C}$ intersecting D can be reported in $O(n^{1/2+\epsilon} + k)$ time, where ϵ is an arbitrarily small constant and k is the output size. Furthermore the data structure can be updated under insertion/deletion of a unit-circle arc in $O(\log^2 n)$ amortized time.*

References

1. Pankaj K. Agarwal. Range searching. In Jacob E. Goodman, Joseph O'Rourke, and Csaba Tóth, editors, *Handbook of Discrete and Computational Geometry*, chapter 40. Chapman & Hall/CRC, 3rd edition, 2017.
2. Pankaj K. Agarwal. *Simplex Range Searching and Its Variants: A Review*, pages 1–30. Springer International Publishing, Cham, 2017.
3. Pankaj K. Agarwal, Ravid Cohen, Dan Halperin, and Wolfgang Mulzer. Maintaining the Union of Unit Discs under Insertions with Near-Optimal Overhead, 2019. `arXiv:1903.10943v1`.
4. Pankaj. K. Agarwal and Jiří Matoušek. On range searching with semialgebraic sets. *Discrete & Computational Geometry*, 11(4):393–418, 1994.
5. Pankaj K. Agarwal and Jiří Matoušek. Dynamic Half-Space Range Reporting and Its Applications. *Algorithmica*, 13(4):325–345, 1995.

6 F. Aurenhammer. Improved algorithms for discs and balls using power diagrams. *Journal of Algorithms*, 9(2):151–161, 1988. doi:10.1016/0196-6774(88)90035-1.

7 Gerth Stølting Brodal and Riko Jacob. Dynamic Planar Convex Hull with Optimal Query Time. In *Algorithm Theory - SWAT 2000, 7th Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 5-7, 2000, Proceedings*, pages 57–70, 2000.

8 Gerth Stølting Brodal and Riko Jacob. Dynamic Planar Convex Hull. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, pages 617–626, 2002.

9 Timothy M. Chan. Dynamic planar convex hull operations in near-logarithmaic amortized time. *J. ACM*, 48(1):1–12, 2001.

10 Timothy M. Chan. A dynamic data structure for 3-D convex hulls and 2-D nearest neighbor queries. *J. ACM*, 57(3):16:1–16:15, 2010.

11 Ravid Cohen, Dan Halperin, and Yossi Yovel. Sensory Regimes of Effective Distributed Searching without Leaders. Manuscript, 2018.

12 Mark de Berg, Kevin Buchin, Bart MP Jansen, and Gerhard Woeginger. Fine-grained complexity analysis of two classic TSP variants. *arXiv preprint*, 2016. arXiv:1607.02725.

13 Dan Halperin and Mark H. Overmars. Spheres, molecules, and hidden surface removal. *Comput. Geom.*, 11(2):83–102, 1998.

14 Dan Halperin and Micha Sharir. Arrangements. In Jacob E. Goodman, Joseph O'Rourke, and Csaba Tóth, editors, *Handbook of Discrete and Computational Geometry*, chapter 28. Chapman & Hall/CRC, 3rd edition, 2017.

15 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic Planar Voronoi Diagrams for General Distance Functions and their Algorithmic Applications. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2495–2504, 2017.

16 Haim Kaplan, Robert Endre Tarjan, and Kostas Tsioutsiouliklis. Faster kinetic heaps and their use in broadcast scheduling. In *Proceedings of the 12th Annual Symposium on Discrete Algorithms*, pages 836–844, 2001.

17 Klara Kedem, Ron Livne, János Pach, and Micha Sharir. On the Union of Jordan Regions and Collision-Free Translational Motion Amidst Polygonal Obstacles. *Discrete & Computational Geometry*, 1:59–70, 1986.

18 Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23(2):166–204, 1981.

19 Franco P. Preparata and Michael Ian Shamos. *Computational Geometry. An Introduction.* Springer-Verlag, New York, 1985.

20 Paul G Spirakis. Very fast algorithms for the area of the union of many circles. *Report no. 98 – Dept. Computer Science, Courant Institute, New York University*, 1983.