A Dynamic $(1+\varepsilon)$-Spanner for Disk Intersection Graphs

Sarita de Berg
IT University of Copenhagen, Denmark

Ivor van der Hoog
IT University of Copenhagen, Denmark

Eva Rotenberg
IT University of Copenhagen, Denmark

Johanne Müller Vistisen
IT University of Copenhagen, Denmark

Sampson Wong
University of Copenhagen, Denmark

Abstract

We maintain a $(1+\varepsilon)$-spanner over the disk intersection graph of a dynamic set of disks. We restrict all disks to have their diameter in $[4, \Psi]$ for some fixed and known Ψ. The resulting $(1+\varepsilon)$-spanner has size $O(n\varepsilon^{-2} \log \Psi \log(\varepsilon^{-1}))$, where n is the present number of disks.

We develop a novel use of persistent data structures to dynamically maintain our $(1+\varepsilon)$-spanner. Our approach requires $O(\varepsilon^{-2} n \log^4 n \log \Psi)$ space and has an $O(\left(\frac{\Psi}{\varepsilon}\right)^2 \log^4 n \log^2 \Psi \log^2(\varepsilon^{-1}))$ expected amortised update time. For constant ε and Ψ, this spanner has near-linear size, uses near-linear space and has polylogarithmic update time. Furthermore, we observe that for any $\varepsilon < 1$, our spanner also serves as a connectivity data structure. With a slight adaptation of our techniques, this leads to better bounds for dynamically supporting connectivity queries in a disk intersection graph. In particular, we improve the space usage when compared to the dynamic data structure of (Baumann et al., DCG'24), replacing the linear dependency on Ψ by a polylogarithmic dependency. Finally, we generalise our results to d-dimensional hypercubes.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases intersection graphs, dynamic data structures, spanners

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.52

Related Version *Full Version*: `https://arxiv.org/abs/2604.25397` [29]

Funding This research was supported by Danmarks Frie Forskningsfond Case no. 10.46540/3160-00020B, the VILLUM Foundation grant (VIL37507) "Efficient Recomputations for Changeful Problems", and the European Union's Marie Skłodowska-Curie Actions Postdoctoral Fellowship Project number No. 101146276.

1 Introduction

Given a set of n geometric shapes, e.g. disks, the corresponding intersection graph has the shapes as its vertex set, and an edge between two shapes whenever they intersect (see Figure 1). Disk intersection graphs are often used as a model for wireless ad-hoc networks [13, 28, 33], in particular, the radius of a disk equals the transmission range of the device. One of the central algorithmic challenges of disk intersection graphs is that, with n vertices, the graph may have $\Omega(n^2)$ edges. Nonetheless, there are many problems that can be solved in subquadratic time in disk graphs [4, 8, 24, 26, 42, 43, 47]. Both weighted [4, 22, 32, 34] and unweighted [8, 10, 12, 47] disk intersection graphs have been studied; we will consider weighted disk intersection graphs

where the weight of an edge is the Euclidean distance between the centres of the disks. We study two dynamic problems: maintaining a spanner for the disk intersection graph or a data structure for connectivity queries (for the latter, the edge weights are irrelevant).

Geometric spanners. Given a graph, a spanner is a subgraph that approximately preserves the distances of the original graph. Formally, a $(1+\varepsilon)$-spanner for a graph G is a subgraph G' such that for every pair of vertices $u, v \in G$, we have that $d_{G'}(u,v) \leq (1+\varepsilon) \cdot d_G(u,v)$, where $d_X(\cdot,\cdot)$ denotes the shortest path distance in X. Throughout the paper, we assume that ε is an arbitrary fixed constant with $0 < \varepsilon < 1$. In computational geometry, there are many works that consider spanners of *point sets*, where the graph G is the complete graph over the point set and edge weights correspond to Euclidean distances. Results for such Euclidean spanners of point sets include a variety of $(1+\varepsilon)$-spanners of size $O(n/\varepsilon^{d-1})$ [3,5,14,25,37,48,55]. For a thorough treatment of spanners for Euclidean point sets, we reference the textbook [50]. Spanners are also well studied in other settings, such as in doubling spaces [16,39], hyperbolic spaces [45,46], general graphs [3,51], minor-free graphs [11,36] and polygonal domains [1,2,9]. Many of these spanners have also been studied in a dynamic setting, where the goal is to maintain a spanner when vertices are inserted or deleted [6,19,23,35,46].

Spanners in disk graphs. There exists a variety of static spanners for unweighted intersection graphs [15,20,27,31,54]. Fürer and Kasiviswanathan [32] were the first to consider spanners of weighted disk intersection graphs. They constructed a $(1+\varepsilon)$-spanner of size $O(n\varepsilon^{-2})$ in $O(n^{4/3+\delta}\varepsilon^{-4/3}\log^{2/3}\Psi)$ time, with $\delta \geq 0$, for when all disk diameters lie in $[1,\Psi]$. Their results generalise to higher dimensions and other disk-like objects. Kaplan, Mulzer, Roditty, Seiferth, and Sharir [44] provided an improved $(1+\varepsilon)$-spanner construction that runs in $O(n\varepsilon^{-2}\log^9 n\lambda_6(\log n))$ time, where $\lambda_s(t)$ is the maximum length of a Davenport–Schinzel sequence on t symbols of order s [52]. New techniques by Liu [49] improve their construction time to $O(n\varepsilon^{-2}\log^4 n)$ expected time.

A natural extension is to dynamically maintain a spanner as disks get inserted and deleted. Surprisingly, to the best of our knowledge, this paper is the first to study the dynamic maintenance of a disk intersection graph spanner. This includes weighted, unweighted, and even unit disk intersection graphs. However, we note that some closely related problems have been studied, such as dynamic connectivity.

Dynamic connectivity in disk graphs. In the connectivity problem, the goal is to maintain a data structure on the graph such that connectivity queries can be answered efficiently. A *connectivity query* takes two disks σ, ρ as input and returns whether σ and ρ are in the

Figure 1 A set of disks $\mathcal{S}$ (left) and the corresponding disk graph $\mathcal{D}(\mathcal{S})$ (right) that includes both the thick black and the thin grey edges. The thick black edges form a $(1+\varepsilon)$-spanner for $\varepsilon = 3/4$.

same connected component of the disk graph. The most common approach for dynamic connectivity in disk graphs is to construct a proxy graph (a sparser subgraph of the disk intersection graph) and then maintain this sparser graph in the dynamic graph connectivity data structure introduced by Holm, de Lichtenberg, and Thorup [40].

Kaplan, Mulzer, Roditty, Seiferth, and Sharir [44] study a dynamic set of disks $\mathcal{S}$ whose radii lie in $[1, \Psi]$. Using the above approach, their data structure supports connectivity queries in $O(\log n / \log\log n)$ and has an expected amortised update time of $O(\Psi^2 \log^9 n \lambda_6(\log n))$, which can be improved to $O(\Psi^2 \log^4 n)$ using [49]. The overall space is $O(\Psi^2 n \log n)$. Baumann, Kaplan, Klost, Knorr, Mulzer, Roditty, and Seiferth [7] further improve the expected amortised update time to $O(\Psi \log^4 n)$ and the space to $O(\Psi n \log n)$. If the input consists of axis-aligned squares, van der Hoog, Nusser, Rotenberg, and Staals [41] maintain connectivity using $O(\Psi \log^4 n + \log^6 n)$ amortised update time and $O(n \log^3 n \log \Psi)$ space.

Finally, Chan and Huang [21] consider the case where there are no restrictions on the disk radii. They use a completely different approach to obtain a data structure with constant query time and an amortised update time of $O(n^{7/8+\varepsilon})$ for an arbitrarily small $\varepsilon > 0$. Polylogarithmic results for the insertion-only and deletion-only cases have been obtained by Baumann, Kaplan, Klost, Knorr, Mulzer, Roditty, and Seiferth [7] using techniques very similar to ours. Nevertheless, this approach has serious robustness issues when studying a fully dynamic set of disks with unbounded radii. One of the reasons for this is that there exist no fully dynamic compressed quadtrees on the real RAM [38, 53].

Our contributions. We introduce the first dynamic $(1+\varepsilon)$-spanner for disk intersection graphs. Our model for dynamic disks follows [7, 41, 44], whereby each disk has its diameter in $[4, \Psi]$ for a fixed and known Ψ. We present a $(1+\varepsilon)$-spanner of $O(n\varepsilon^{-2} \log \Psi \log(\varepsilon^{-1}))$ size. Our expected amortised update time is $O((\frac{\Psi}{\varepsilon})^2 \log^4 n \log^2 \Psi \log^2(\varepsilon^{-1}))$, and our expected total space is $O(n\varepsilon^{-2} \log^4 n \log \Psi)$.

By using similar techniques, we obtain a dynamic connectivity data structure with $O(\Psi \log^4 n \log \Psi)$ expected amortised update time and $O(n \log^4 n \log \Psi)$ expected space. Compared to the best known result of [7], we match their update time up to logarithmic factors and improve their space from linear in Ψ to logarithmic in Ψ.

Finally, we note that all our results generalise to d-dimensional hypercubes by replacing a disk intersection data structure with a hypercube intersection data structure. Table 1 provides an overview of our results and the most relevant related work.

Organisation. After covering the preliminaries in Section 2, we present an overview of our results and techniques in Section 3. In Section 4, we present an easier variant of our dynamic $(1+\varepsilon)$-spanner for disks within a fixed bounding box that uses $\tilde{O}((\frac{\Psi}{\varepsilon})^2 n)$ space. In Appendix A, we reduce the space usage by a factor of $O(\Psi^2)$ (omitting some details which can be found in the full version [29]) and in the full version of this paper [29], we remove the bounding box assumption. In the full version of the paper [29], we present our results on dynamic connectivity. Finally, we generalise all of our results to d-dimensional hypercubes (see the full version of this paper [29]).

2 Preliminaries

The input is a dynamic set $\mathcal{S} = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ of n disks in the plane, or a set of n d-dimensional cubes in $\mathbb{R}^d$ for some constant dimension d. These preliminaries focus on disks in the plane, but the concepts are defined analogously for d-dimensional cubes in $\mathbb{R}^d$. To

Table 1 Table of our results. The size of the spanners is $O(n\varepsilon^{-d}\log\Psi\log\varepsilon^{-1})$. All connectivity structures have $O(\log n/\log\log n)$ query time.

Dynamic $(1+\varepsilon)$-spanners, fixed aspect ratio Ψ

	Amortised update time	Space	Reference
2D disks	$O((\frac{\Psi}{\varepsilon})^2\log^4 n\log^2\Psi\log^2(\varepsilon^{-1}))$ exp.	$O(n\,\varepsilon^{-2}\log^4 n\log\Psi)$ exp.	Thm. 1
d-dim cubes	$O((\frac{\Psi}{\varepsilon})^d\log^d n\log^2\Psi\log^2(\varepsilon^{-1}))$	$O(n\,\varepsilon^{-d}\log^d n\log\Psi)$	Thm. 3

Dynamic connectivity, fixed aspect ratio Ψ

	Amortised update time	Space	Reference
2D squares	$O(\Psi\log^4 n+\log^6 n)$	$O(n\log^3 n\log\Psi)$	[41, Thm. 10]
2D disks	$O(\Psi^2\log^4 n)$ exp.	$O(\Psi^2 n\log n)$	[44, 49]
2D disks	$O(\Psi\log^4 n)$ exp.	$O(\Psi n\log n)$	[7, Thm. 4.6]
2D disks	$O(\Psi\log^4 n\log\Psi)$ exp.	$O(n\log^4 n\log\Psi)$ exp.	Thm. 2
d-dim cubes	$O(\Psi^{d-1}\log^d n\log\Psi)$	$O(n\log^d n\log\Psi)$	Thm. 4

minimise numerical dependencies on $\sqrt{2}$, we define for any convex shape C the diameter $|C|$ of C as the length of the largest horizontal segment contained in C. We assume that, at all times, $\forall\sigma\in\mathcal{S}$, the diameter is at least 4 and at most Ψ, where Ψ is some fixed and known value. In other words, $|\sigma|\in[4,\Psi]$. For now, we assume that at all times $\mathcal{S}$ lies contained within a bounding box $B=[0,\Psi^*]^2$, where Ψ^* is the smallest power of two that is bigger than Ψ. In the full version of this paper [29], we drop this bounding box assumption.

The *disk intersection graph* $\mathcal{D}(\mathcal{S})$ is an edge-weighted graph where vertices are the disks in $\mathcal{S}$, and there is an edge (σ_1,σ_2) whenever the two disks intersect. The weight $d(\sigma_1,\sigma_2)$ of (σ_1,σ_2) is the Euclidean distance between the disk centres. For any edge-weighted graph G over $\mathcal{S}$, we let $d_G(\sigma_1,\sigma_2)$ denote the length of the shortest path between σ_1 and σ_2 in G.

Problem statement. Since disk intersection graphs can be very dense, one may be interested in maintaining a sparse *spanner* of this graph. A $(1+\varepsilon)$-spanner of a disk intersection graph $\mathcal{D}(\mathcal{S})$ is an edge-weighted subgraph G of $\mathcal{D}(\mathcal{S})$ such that for any $\sigma_1,\sigma_2\in S$, we have $d_G(\sigma_1,\sigma_2)\le(1+\varepsilon)\cdot d_{\mathcal{D}(\mathcal{S})}(\sigma_1,\sigma_2)$. The *size* of a spanner is its number of edges. We wish to maintain a $(1+\varepsilon)$-spanner of near-linear size subject to disk insertions and deletions in $\mathcal{S}$. The running time, space usage, and size of the spanner depend on n, ε, and Ψ. Our results apply to both the word RAM, real RAM, and pointer machine model of computation.

Dynamic Euclidean spanner. We will use the following dynamic spanner in our construction.

▶ **Lemma 1** ([19, 35]). *For any d-dimensional point set P of size n there exists a dynamic $(1+\varepsilon)$-spanner of their pairwise Euclidean distances that has size $O(n\varepsilon^{-d}\log(\varepsilon^{-1}))$ that can be maintained using $O(n\varepsilon^{-d}\log(\varepsilon^{-1}))$ space and $O(\varepsilon^{-d}\log n\log^2(\varepsilon^{-1}))$ update time.*

Quadtree. We will also use the following quadtree in our construction.

▶ **Definition 2.** *Let B be an axis-aligned d-dimensional hypercube whose side length is a positive power of two. A* split *operation selects a hypercube and divides it into 2^d equal hypercubes. A* quadtree *is any tree obtained by recursively applying the split operation on B. Each node of the tree corresponds to a hypercube in $\mathbb{R}^d$, called a* cell.

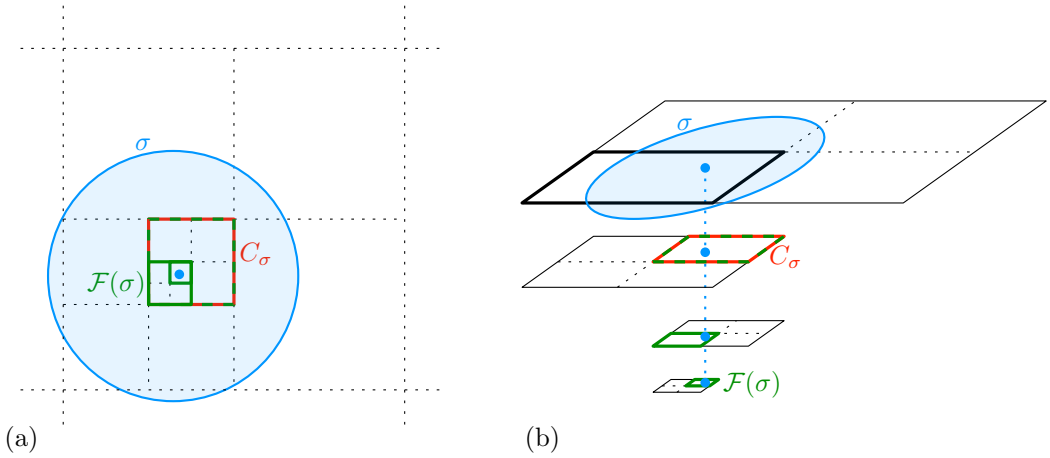

Figure 2 For disk σ, the storing cell is C_σ and its storing family $\mathcal{F}(\sigma)$ in (a) 2D and (b) 3D view.

For a cell C and any odd integer k, we denote by $k * C$ the region $[0, k|C|]^2$ translated to have C as its centre. For any disk σ, we define its *storing cell* C_σ (see Figure 2) as the largest cell, obtained by recursively splitting B, that: (1) contains the centre of σ and (2) is contained in σ. The *storing family* $\mathcal{F}(\sigma)$ of σ is the set of all cells, down to diameter 1, that can be obtained by recursively splitting C_σ and contain the centre of σ. Typically, one uses the minimum-size quadtree that contains all storing cells C_σ for $\sigma \in \mathcal{S}$ [7, 32]. We deviate this standard by considering the minimum-size quadtree that contains all storing families:

▶ **Definition 3.** *Given a fixed bounding box B and $\mathcal{S}$, we define $T(\mathcal{S})$ as the minimum-size quadtree that contains the storing families $\mathcal{F}(\sigma)$ for all $\sigma \in \mathcal{S}$. For a cell $C \in T(\mathcal{S})$, the set $\pi(C)$ consists of all disks in $\mathcal{S}$ that have C in their storing families.*

In other words, the set $\pi(C)$ of a cell C consists of all disks that have their centre in C and also fully contain C. Since B is fixed as $[0, \Psi^*]^2$ we observe:

▶ **Observation 4.** *The quadtree $T(\mathcal{S})$ has $O(n \log \Psi)$ cells. Moreover, for all cells $C, C' \in T(\mathcal{S})$, where $|C| = |C'|$ and $C' \subset 3 * C$, all pairs $(\sigma, \sigma') \in \pi(C) \times \pi(C')$ intersect.*

Next, we subdivide each cell C into a finer grid of what we call ε-cells (see Figure 3):

▶ **Definition 5.** *For a fixed $0 < \varepsilon < 1$, we partition each cell $C \in T(\mathcal{S})$ into a (d-dimensional) grid of cells of side length $\varepsilon|C|$, called* ε-cells. *Let E be an ε-cell of a cell C, then the* subpopulation $\Gamma_\varepsilon(E)$ *denotes all disks that have C as their storing cell and their centre in E. Analogously, the* population $\pi_\varepsilon(E)$ *denotes all disks in $\pi(C)$ that have their centre in E.*

Disk intersection queries. Let $\mathcal{S}$ be a set of n disks. Given a query disk ρ, the *disk intersection query* structure returns a disk in $\mathcal{S}$ that intersects ρ, if such a disk exists.

▶ **Lemma 6** ([44, 49])**.** *We can maintain a set of n disks $\mathcal{S} \subset \mathbb{R}^2$ in a data structure of $O(n \log n)$ size, that can answer intersection queries in $O(\log^2 n)$-time, supporting disk updates (insertions and deletions) in $O(\log^4 n)$ amortised expected time.*

Proof. As observed in [7], the results of Kaplan, Mulzer, Rodity Seiferth, and Sharir [44] and Liu [49] for dynamic additively weighted nearest neighbour queries can be used to obtain a fully dynamic data structure for disk intersection queries with corresponding space and

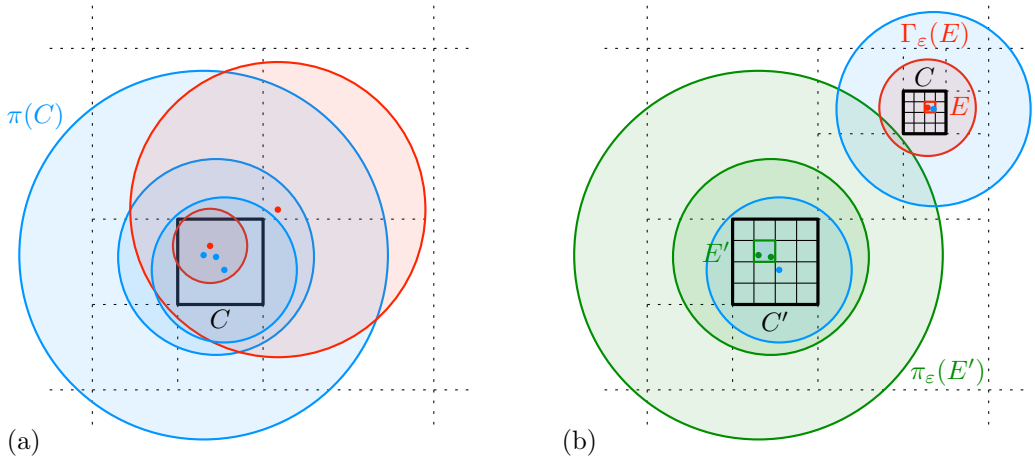

Figure 3 (a) The set $\pi(C)$ consists of all blue disks, the red disks do not belong to $\pi(C)$ because either their centre is not in C, or they do not contain C. (b) The ε-cells E and E' of cells C and C' and their corresponding subpopulation $\Gamma_\varepsilon(E)$ (red) and population $\pi_\varepsilon(E')$ (green).

time bounds as follows. We insert all centres of $\mathcal{S}$ into the data structure, where the weight for each disk is the negative radius of the disk. For an intersection query with a disk ρ with centre point r, we query this data structure with r. Let the point corresponding to a disk σ be closest to r. If the distance is negative, then r is contained in σ. Otherwise, the returned distance d is the distance from r to the perimeter of $\mathcal{S}$ and ρ intersects a disk in $\mathcal{S}$ if and only if d is less than the radius of ρ. ◀

Hypercubes. Hypercube intersection queries are defined analogously, and we obtain all our d-dimensional hypercube results by using the following lemma instead of Lemma 6:

▶ **Lemma 7** (Proof in full version [29]). *We can maintain a set of n d-dimensional hypercubes $\mathcal{S} \subset \mathbb{R}^d$ in a data structure of $O(n \log^{d-1} n)$ size, that can answer intersection queries in $O(\log^d n)$ time, and that can handle cube updates (insertion or deletion) in $O(\log^d n)$ time.*

Pointer machine data structures. In our paper, we will make use of data structure *persistence* as described by Driscoll, Sarnak, Sleator, and Tarjan [30]. To this end, we require that the data structures that we use (Lemmas 6 and 7) are implementable on a *pointer machine*. We can then either assume a pointer machine as our model of computation, or simulate a pointer machine on the RAM model of our choice. A pointer machine consists of *nodes* and *pointers*. A *data node* stores any type of data (integers, logical values, strings, real numbers, etc), and the dynamic input (our dynamic set of disks) is a set of data nodes.

A *pointer node* stores a number of constantly many bits. Each node has constantly many outgoing pointers, where a pointer is a directed edge from one node to another. An *update* adds or removes a data node to or from the input. A *dynamic data structure* maintains an arbitrarily large set of pointer nodes and, for each data node, a pointer node that points to it. One pointer node is called the *root*, and we maintain the invariant that any node can be reached by a directed path from the root. The *size* of the data structure is the total number of nodes it has. Any update or query runs a *program* which, starting from the root node, can execute instructions such as *pointer traversals*, *comparisons* between nodes that share a

pointer, and *atomic operations* (such as addition between nodes that share a pointer). The *running time* of a program is the number of instructions it executes before termination. The following can be observed from [18, 44, 49] and has been confirmed by [17]:

▶ **Observation 8.** *Both data structures of Lemmas 6 and 7 can be implemented on a pointer machine. However, pointer machine nodes will not necessarily have constant in-degree.*

3 Overview of techniques

The main focus of this paper is to dynamically maintain a $(1+\varepsilon)$-spanner of $\mathcal{D}(\mathcal{S})$. In the main body, we assume that $\mathcal{S}$ remains contained within a bounding box $[0, \Psi^*]^2$, where Ψ is a known upper bound on the diameter of the disks in $\mathcal{S}$ and $\Psi^* = 2^{\lceil \log_2 \Psi \rceil}$. This assumption allows us to ignore dynamic quadtree misalignment issues (see full version [29]). The basis of our spanner is a quadtree $T(\mathcal{S})$ that stores $\mathcal{S}$ (see Definition 3). We then construct a spanner G by using two types of edges: type i edges connect two disks in quadtree cells that are "close", whereas type ii edges connect two disks that lie in ε-cells that are "far away".

We obtain type i edges by considering pairs of "close" quadtree cells (C_1, C_2), where close is defined such that all disks assigned to C_1 intersect all disks assigned to C_2. We then dynamically maintain a Euclidean $(1+\varepsilon)$-spanner (Lemma 1) on the centre points of disks assigned to C_1 or C_2, and for each edge in this Euclidean spanner, we add a type i edge to G.

Type ii edges are instead constructed for pairs of ε-cells (Definition 5). For any pair (E, E') that lies within a specified distance, we first test whether any two disks in $\Gamma_\varepsilon(E) \times \pi_\varepsilon(E')$ intersect. One can intuitively imagine this as testing whether any two disks across their populations intersect, but the definition requires slightly more care so as to avoid an additional $O(\log \Psi)$ factor across the paper. If there exist any intersecting disks in the bichromatic intersection graph of $\Gamma_\varepsilon(E) \times \pi_\varepsilon(E')$, we add a single edge in the spanner between an arbitrary intersecting pair. To dynamically maintain type ii edges under disk insertions and deletions, we need to be able to maintain whether there is *any* intersection in $\Gamma_\varepsilon(E) \times \pi_\varepsilon(E')$. To this end, we maintain a maximal bichromatic matching in this graph. To do this, we first show a straightforward approach based on [7, 44]: for any pair (E, E') we maintain two separate disk intersection data structures (Lemma 6) storing the disks in $\Gamma_\varepsilon(E)$ and $\pi_\varepsilon(E')$ that do not appear in the matching. In this way, we can immediately query whether a new disk σ can be matched to an existing but unmatched disk. This leads to the following result:

▶ **Lemma 9.** *Let $\mathcal{S}$ be a fully dynamic set of disks in the plane with diameters in $[4, \Psi]$ for a fixed and known Ψ. For any fixed $\varepsilon \in (0, 1)$, we can maintain a $(1+\varepsilon)$-spanner of $\mathcal{D}(\mathcal{S})$ of size $O(n\varepsilon^{-2} \log \Psi \log(\varepsilon^{-1}))$ using $O(\left(\frac{\Psi}{\varepsilon}\right)^2 \log^4 n \log \Psi \log^2(\varepsilon^{-1}))$ expected amortised update time and $O(\left(\frac{\Psi}{\varepsilon}\right)^2 n \log n \log \Psi \log(\varepsilon^{-1}))$ space.*

The dynamic spanner in Lemma 9 has an $\Theta(\Psi^2)$ dependence in both its update time and space. We observe that the $\Omega(\Psi^2)$ dependence in update time is unavoidable:

▶ **Observation 10** (Figure 4)**.** *Any fully dynamic data structure that maintains a $O(1)$-spanner of $\mathcal{D}(\mathcal{S})$ has $\Omega(\Psi^2)$ update time.*

The $\Omega(\Psi^2)$ dependence in space comes from our naive strategy of storing a separate disk intersection data structure for every pair of ε-cells (E, E') at a specified distance. To reduce this space dependence, our main insight is to introduce a new form of data structure persistence: *branch persistence* (Definition 16). This type of persistence allows access and updates to the main (root) version of the data structure and additionally allows access and

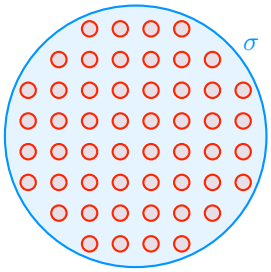

Figure 4 If σ is deleted or inserted in $\mathcal{S}$, $\Theta(\Psi^2)$ edges have to be added or removed in the spanner.

updates on several *branches*. An update to the root also performs the corresponding update on all branches. A branch persistent data structure supports branch creation, branch queries, branch-updates and complete rebuilds. Instead of maintaining a disk intersection query data structure (Lemma 6) for every pair of suitable ε-cells, which requires $\Omega(\Psi^2)$ space, we instead maintain a branch persistent disk intersection data structure (Lemma 18) for each ε-cell. The corresponding disk intersection data structure can then be accessed either via a branch or via the root version of the branch persistent data structure. By bounding the total symmetric differences between the branches, we obtain the following dynamic spanner:

▶ **Theorem 1.** *Let $\mathcal{S}$ be a fully dynamic set of disks in the plane with diameters in $[4, \Psi]$ for a fixed and known Ψ. For any fixed $0 < \varepsilon < 1$, we can maintain a $(1+\varepsilon)$-spanner of $\mathcal{D}(\mathcal{S})$ of size $O(n\,\varepsilon^{-2}\log\Psi\log(\varepsilon^{-1}))$ using $O((\frac{\Psi}{\varepsilon})^2\log^4 n\log^2\Psi\log^2(\varepsilon^{-1}))$ expected amortised update time and $O(n\,\varepsilon^{-2}\log^4 n\log\Psi)$ expected space.*

Removing the bounding box assumption. The dynamic spanners in Lemma 9 and Theorem 1 require that the disks in $\mathcal{S}$ remain in the bounding box $B = [0, \Psi^*]^2$. Given a disk σ, our quadtree recursively splits B until we obtain a unit cell that contains the centre of σ. Unfortunately, if B is unbounded, this procedure takes unbounded time.

In the static setting, this issue can be avoided by using a compressed quadtree. However, despite papers frequently using dynamic compressed quadtrees, there exists no dynamic compressed quadtree due *dynamic quadtree misalignment*. As the formal definition of dynamic quadtree misalignment is complex, we defer the definition to the full version of this paper [29]. The issue of dynamic quadtree misalignment was also identified in the journal version of [7, Appendix A]. There, they describe a complex approach to maintaining connectivity despite misalignment, which relies on the fact that the disks have a diameter in $[1, \Psi]$.

In the full version [29], we give a simple and general approach that can be applied to [7, 41, 44]. Since Ψ is fixed and we only require amortised bounds, it suffices to maintain an insertion-only disjoint collection of boxes $B_i \subset \mathbb{R}^2$ satisfying: (i) each B_i defines an uncompressed quadtree storing $\{\sigma \in \mathcal{S} \mid B_i \cap \sigma \neq \emptyset\}$, (ii) all $\sigma \in \mathcal{S}$ intersect at least one box B_i, and (iii) the union over all uncompressed quadtree spanners is a spanner of $\mathcal{S}$. After $N \in \Theta(n)$ updates, we simply rebuild our data structure from scratch – deleting all B_i that are not intersected by a disk in $\mathcal{S}$. As a consequence, we dynamically maintain our disk intersection graph representation with no misalignment issues and no asymptotic slowdown.

Connectivity. A $(1+\varepsilon)$-spanner preserves connectivity. By storing our spanner G in the dynamic connectivity data structure of Holm, de Lichtenberg and Thorup [40], we can also answer connectivity queries in time $O(\log n/\log\log n)$ at no overhead. This immediately yields a connectivity data structure that uses less space than the state-of-the-art [7]. However,

its update time depends quadratically on Ψ, versus linearly in [7]. In the full version of this paper [29], we combine the techniques in our dynamic spanner with the dynamic connectivity pipeline of [7, 41] to obtain a linear dependence on Ψ whilst maintaining the same space usage. Incorporating our techniques into the pipeline is relatively straightforward, as it simply involves reconstructing their lemmas using our branch persistent data structure. This yields the following theorem:

▶ **Theorem 2** (Proof in full version [29]). *Let $\mathcal{S}$ be a fully dynamic set of disks in the plane with diameters in $[4, \Psi]$ for a fixed and known Ψ. We can dynamically maintain $\mathcal{S}$ supporting* connectivity queries *in time $O(\log n / \log\log n)$ using $O(\Psi \log^4 n \log \Psi)$ expected amortised update time and $O(n \log^4 n \log \Psi)$ expected space.*

Hypercube intersection graphs. In the full version of this paper [29], we show that our techniques extend to intersection graphs of d-dimensional hypercubes. We maintain a $(1+\varepsilon)$-spanner and a fully dynamic connectivity data structure for a dynamic set of n d-dimensional hypercubes by replacing the disk intersection data structure (Lemma 6) with a hypercube intersection data structure (Lemma 7). For our spanner, the update time depends on a factor $\Theta(\Psi^d)$ as there are $\Theta(\Psi^d)$ unit hypercubes contained in a hypercube of diameter Ψ.

▶ **Theorem 3** (Proof in full version [29]). *Let $\mathcal{S}$ be a fully dynamic set of d-dimensional axis-aligned hypercubes in $\mathbb{R}^d$ with side lengths in $[4, \Psi]$ for a fixed and known Ψ and constant dimension d. For any fixed $0 < \varepsilon < 1$, we can maintain a $(1+\varepsilon)$-spanner of $\mathcal{D}(\mathcal{S})$ of size $O(n\,\varepsilon^{-d} \log \Psi \log(\varepsilon^{-1}))$ using $O((\frac{\Psi}{\varepsilon})^d \log^d n \log^2 \Psi \log^2(\varepsilon^{-1}))$ amortised update time and $O(n\,\varepsilon^{-d} \log^d n \log \Psi)$ space.*

For connectivity, the pipeline from [7, 41] shaves a single Ψ factor from the update time.

▶ **Theorem 4** (Proof in full version [29]). *Let $\mathcal{S}$ be a fully dynamic set of d-dimensional axis-aligned hypercubes in $\mathbb{R}^d$ with diameter in $[4, \Psi]$ for a fixed and known Ψ and constant dimension d. We can dynamically maintain $\mathcal{S}$ supporting* connectivity queries *in time $O(\log n / \log\log n)$ using $O(\Psi^{d-1} \log^d n \log \Psi)$ amortised update time and $O(n \log^d n \log \Psi)$ total space.*

4 Dynamic spanner using $\tilde{O}(\Psi^2 n)$ space within a fixed bounding box

The input is a dynamic set $\mathcal{S} = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ of n disks in the plane, where $\mathcal{S}$ remains contained within the bounding box $[0, \Psi^*]^2$ and $\forall \sigma \in \mathcal{S}$, $|\sigma| \in [4, \Psi]$. We denote by $T(\mathcal{S})$ the dynamic quadtree (Definition 2) storing $\mathcal{S}$ which has $O(n \log \Psi)$ space and can trivially be maintained using $O(\log \Psi)$ update time. We define a graph $G \subseteq \mathcal{D}(\mathcal{S})$ that is a $(1+9\varepsilon)$-spanner of size $O(n\varepsilon^{-2} \log \Psi \log(\varepsilon^{-1}))$. By choosing $\varepsilon' = \varepsilon/9$, we obtain a $(1+\varepsilon)$-spanner of the same asymptotic size. Our spanner G has two types of edges (see Figure 5):

i. For each cell C in the quadtree $T(\mathcal{S})$, consider all other cells $C' \in T(\mathcal{S})$ of size $|C|$ in the region $3 * C$. For each pair of cells (C, C'), use Lemma 1 to construct a Euclidean $(1+\varepsilon)$-spanner $Span(C, C')$ on the centre points of disks in $\pi(C) \cup \pi(C')$. For every edge in the Euclidean $(1+\varepsilon)$-spanner, add an edge between the pair of corresponding disks to our graph G. By Observation 4, this edge lies in the disk intersection graph $\mathcal{D}(\mathcal{S})$.
ii. For each cell C in the quadtree and $i \in [1, \lceil \log \Psi \rceil]$, consider the cells C' such that $|C'| = 2^{i-1}|C|$ and C' intersects $(2^{i+5}+1) * C$. For every pair of ε-cells E of C and E' of C', if there exists a pair $(\sigma, \sigma') \in \Gamma_\varepsilon(E) \times \pi_\varepsilon(E')$ that intersect, then we select an arbitrary pair of intersecting disks $(\sigma_1, \sigma_2) \in \Gamma_\varepsilon(E) \times \pi_\varepsilon(E')$ and add (σ_1, σ_2) to G.

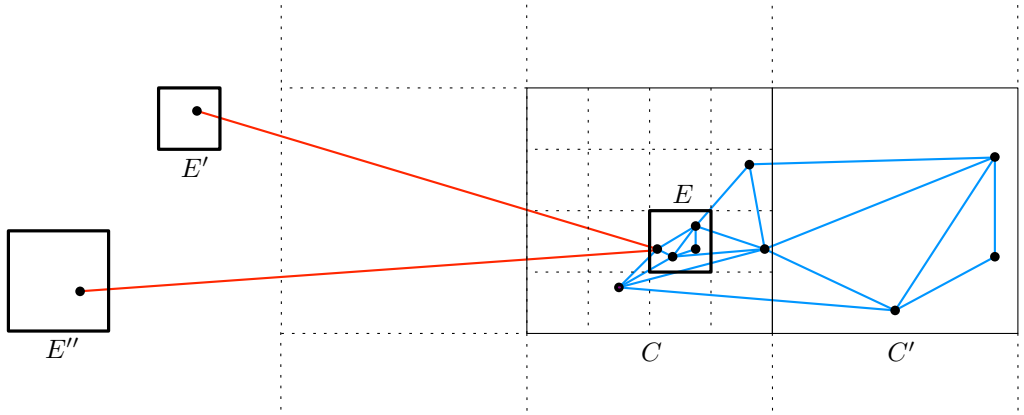

Figure 5 The two types of edges where blue edges are type i and red edges are type ii.

For type ii edges, it will be useful to consider (σ_1, σ_2) to be an ordered pair. In other words, if we write that (σ_1, σ_2) is a type ii edge, then we have that $\sigma_1 \in \Gamma_\varepsilon(E)$ and $\sigma_2 \in \pi_\varepsilon(E')$, where E is an ε-cell of C, E' is an ε-cell of C', $|C'| = 2^{i-1}|C|$ and C' intersects $(2^{i+5}+1) * C$.

We show that G is a $(1+9\varepsilon)$-spanner of $\mathcal{D}(\mathcal{S})$. Specifically, we will show that for every edge $(\sigma_1, \sigma_2) \in \mathcal{D}(\mathcal{S})$ there exists a path in G from σ_1 to σ_2 of length at most $(1+9\varepsilon) \cdot d(\sigma_1, \sigma_2)$. To this end, consider the storing cells C_1 of σ_1, and C_2 of σ_2. Assume without loss of generality that $|C_1| \leq |C_2|$. We have two cases: either the centre of σ_2 lies inside $3 * C_1$ or not.

▶ **Lemma 11.** *If the centre of σ_2 lies in $3 * C_1$ then there exists a path in G from σ_1 to σ_2 of length at most $(1+\varepsilon) \cdot d(\sigma_1, \sigma_2)$.*

Proof. Since the centre of σ_2 lies in $3 * C_1$, the cell C_2 must have a descendent C_2' of size $|C_1|$ that is contained in $3 * C_1$ where $\sigma_2 \in \pi(C_2')$. By construction of the type i edges, G contains an Euclidean $(1+\varepsilon)$-spanner over the centre points of all the disks in $\pi(C_1) \cup \pi(C_2')$, so there is a path of length $\leq (1+\varepsilon) \cdot d(\sigma_1, \sigma_2)$ from σ_1 to σ_2 using only type i edges. ◀

▶ **Lemma 12.** *If the centre of σ_2 does not lie in $3 * C_1$ then there exists a path in G from σ_1 to σ_2 of length at most $(1+9\varepsilon) \cdot d(\sigma_1, \sigma_2)$.*

Proof. Intuitively, since the centre of σ_2 is "far away" relative to σ_1, we will require both type i and type ii edges to construct a spanning path from σ_1 to σ_2. Recall that C_1 is the storing cell of σ_1 and consider the storing family $\mathcal{F}(\sigma_2)$ of σ_2. The first step is to find a cell C' in the storing family $\mathcal{F}(\sigma_2)$ such that there exists a type ii edge between C_1 and C'. We will consider two cases, depending on the relative sizes of $|C_1|$ and $d(\sigma_1, \sigma_2)$. Let $\Delta = d(\sigma_1, \sigma_2)$. Since the centre of σ_2 is not in $3 * C_1$, we have $\Delta \geq |C_1|$, so there exists an integer $j \geq 0$ such that $\Delta \in [2^j|C_1|, 2^{j+1}|C_1|)$. Our two cases are $0 \leq j \leq 3$ and $j > 3$.

- If $0 \leq j \leq 3$ then $\Delta \in [|C_1|, 2^4|C_1|)$. As $|C_2| \geq |C_1|$, there exists a cell $C' \in \mathcal{F}(\sigma_2)$ for which $|C'| = |C_1|$. We also know that C' lies in $(2^5+1) * C_1$ because $\Delta < 2^4|C_1|$. Therefore, the definition of type ii edges applies to the cells C_1 and C' (by setting $i = 1$ in the definition), and there exists a type ii edge between C_1 and C'.
- Now for the case of $j > 3$, we consider the positive integer $i := j - 3$, i.e. $\Delta \in [2^{i+3}|C_1|, 2^{i+4}|C_1|)$. The first observation is then that Δ is at most $\frac{1}{2}(|\sigma_1| + |\sigma_2|)$. Moreover, since $|C_2| \geq |C_1|$, $|\sigma_2| \geq \frac{1}{4}|\sigma_1|$ and so $\Delta \leq \frac{5}{2}|\sigma_2|$. Because $|C_2| \geq \frac{|\sigma_2|}{4\sqrt{2}}$ (see Figure 6) it follows that $|C_2| \geq \frac{2}{20\sqrt{2}}\Delta \geq \frac{2}{20\sqrt{2}}2^{i+3}|C_1| \geq 2^{i-1}|C_1|$. This means that there

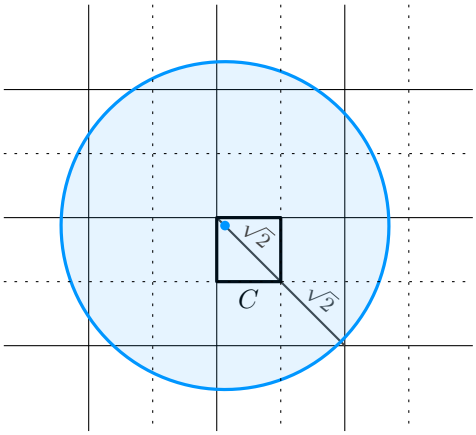

Figure 6 A disk has diameter at most $4\sqrt{2}$ times the diameter of its storing cell.

exists a $C' \in \mathcal{F}(\sigma_2)$ with $|C'| = 2^{i-1}|C_1|$. Because σ_1 and σ_2 intersect and their centres are contained in C_1 and C', the distance between C_1 and C' is at most Δ, which is upper bounded by $2^{i+4}|C_1|$. Thus, we have that C' intersects $(2^{i+5}+1) * C_1$ and $|C'| = 2^{i-1}|C_1|$, and therefore the definition of type ii edges applies to the cells C_1 and C'.

In both cases, we have found a cell C' such that the definition of type ii edges applies to C_1 and C'. We notice that, when applying the definition of type ii edges, it provides us with a type ii edge between every pair of ε-cells that contain a pair of intersecting disks. Specifically, let E_1 be the ε-cell of C_1 that contains the centre of σ_1, and let E_2 be the ε-cell of C' that contains the centre of σ_2. The above analysis provides a type ii edge between an arbitrary pair of intersecting disks $(\sigma_1^*, \sigma_2^*) \in \Gamma_\varepsilon(E_1) \times \pi_\varepsilon(E_2)$ that is in G. Per construction, the centres of σ_1, σ_1^* lie in E_1, and the centres of σ_2, σ_2^* lie in E_2, see Figure 7.

We upper bound the length of the path $\sigma_1 \rightsquigarrow \sigma_1^* \to \sigma_2^* \rightsquigarrow \sigma_2$. Since σ_1 and σ_1^* are both in E_1, we can apply Lemma 11 to observe that there exists a path in G from σ_1 to σ_1^* of length at most $(1+\varepsilon)d(\sigma_1, \sigma_1^*) \leq (1+\varepsilon)\sqrt{2}\varepsilon|C_1| \leq 2\sqrt{2}\varepsilon\Delta \leq 3\varepsilon\Delta$. Similarly, there exists a path in G from σ_2 to σ_2^* of length at most $(1+\varepsilon)d(\sigma_2, \sigma_2^*) \leq (1+\varepsilon)\sqrt{2}\varepsilon|C'| \leq 3\varepsilon\Delta$. Since $d(\sigma_1, \sigma_2) = \Delta$, $d(\sigma_1, \sigma_1^*) \leq \sqrt{2}\varepsilon\Delta$, and $d(\sigma_2, \sigma_2^*) \leq \sqrt{2}\varepsilon\Delta$ then, by the triangle inequality, the length of the type ii edge (σ_1^*, σ_2^*) is at most $(1+3\varepsilon)\Delta$.

Therefore, the length of a spanning path from $\sigma_1 \rightsquigarrow \sigma_1^* \to \sigma_2^* \rightsquigarrow \sigma_2$ is at most $3\varepsilon\Delta + (1+3\varepsilon)\Delta + 3\varepsilon\Delta \leq (1+9\varepsilon)\Delta$. ◀

Next, we bound the size of the spanner.

▶ **Lemma 13.** *The graph G is a $(1+\varepsilon)$-spanner of size $O(n\varepsilon^{-2}\log\Psi\log(\varepsilon^{-1}))$.*

Proof. Lemmas 11 and 12 imply that G is a $(1+9\varepsilon)$-spanner. By computing a $(1+\varepsilon')$ spanner with $\varepsilon' = \frac{\varepsilon}{9}$, we thus obtain a $(1+\varepsilon)$-spanner at no asymptotic size overhead. It remains to bound the size of G, i.e. the number of type i and type ii edges in G. We first bound the number of type i edges. Every cell $C \in T(\mathcal{S})$ has at most $O(1)$ cells C' of size $|C|$ such that $C' \in 3 * C$ or $C \in 3 * C'$. For each pair (C, C'), by Lemma 1, we add $O((|\pi(C)| + |\pi(C')|)\,\varepsilon^{-2}\log(\varepsilon^{-1}))$ type i edges to the subgraph $\pi(C) \cup \pi(C')$. If we uniformly charge the edges in the subgraph, then each disk $\sigma \in \pi(C) \cup \pi(C')$ is charged $O(\varepsilon^{-2}\log(\varepsilon^{-1}))$ times. Each σ can participate in $O(\log\Psi)$ cells C in the quadtree, so each disk σ can be charged at most $O(\varepsilon^{-2}\log\Psi\log(\varepsilon^{-1}))$ times. Therefore, the total number of type i edges is $O(n\varepsilon^{-2}\log\Psi\log(\varepsilon^{-1}))$.

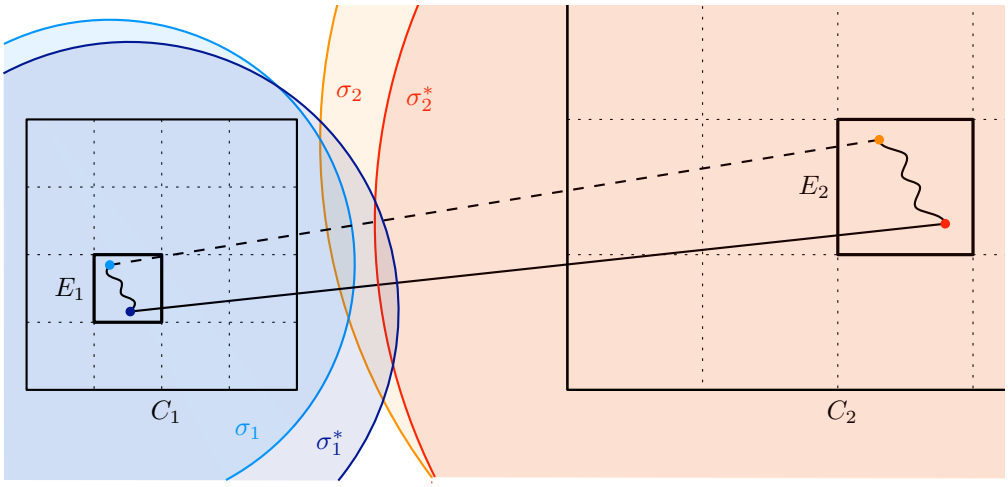

Figure 7 The path $\sigma_1 \rightsquigarrow \sigma_1^* \to \sigma_2^* \rightsquigarrow \sigma_2$ from the proof of Lemma 12.

We also bound for each disk $\sigma \in \mathcal{S}$ the number of type ii edges (σ, σ') that σ participates in, where we recall that type ii edges (σ, σ') are an ordered pair. For each disk σ with storing cell C, there is a unique ε-cell E of C that contains the centre point of σ. For each index $i \in [1, \lceil \log \Psi \rceil]$, there are constantly many cells C' of size $2^{i-1}|C|$ that intersect $(2^{i+5}+1) * C$. For every such cell C', there are $O(\varepsilon^{-2})$ ε-cells E' of C'. For (E, E') there exists at most one type ii edge $(\sigma_1, \sigma_2) \in \Gamma_\varepsilon(E) \times \pi_\varepsilon(E')$ (where σ_1 *may* be σ). Therefore, there can be at most $O(\varepsilon^{-2} \log \Psi)$ edges (σ, σ') of type ii for fixed σ. ◀

▶ **Lemma 14.** *Given a set of disks $\mathcal{S}$ contained in the bounding box $[0, \Psi^*]^2$, we can construct the spanner G in $O(n\varepsilon^{-2} \log^4 n \log \Psi \log^2(\varepsilon^{-1}))$ time.*

Proof. We can construct the quadtree and compute for each cell C the set $\pi(C)$ in $O(n \log \Psi)$ total time. There are at most $O(n \log \Psi)$ ε-cells E with non-empty population $\pi_\varepsilon(E)$. In $O(n \log \Psi)$ time, we can compute the population $\pi_\varepsilon(E)$ and the subpopulation $\Gamma_\varepsilon(E)$ for all of these cells. Next, we consider all $O(n \log \Psi)$ cells C for which $\pi(C)$ is non-empty and the $O(1)$ cells C' for which there are type i edges between disks in C and C'. Constructing the Euclidean spanner between C and C' takes $O((|\pi(C)| + |\pi(C')|)\, \varepsilon^{-2} \log n \log^2(\varepsilon^{-1}))$ time. Since each $\sigma \in S$ appears in $O(\log \Psi)$ sets $\pi(C^*)$ for $C^* \in T(\mathcal{S})$, doing this for all such pairs (C, C') takes $O(n\varepsilon^{-2} \log n \log \Psi \log^2(\varepsilon^{-1}))$ total time.

Next, we consider all ε-cells E for which the population $\pi_\varepsilon(E)$ is non-empty, and we store $\pi_\varepsilon(E)$ in the data structure of Lemma 6, which takes $O(n \log^4 n \log \Psi)$ expected total time. Let E be an ε-cell of a quadtree cell C. We consider for all $i \in [1, \lceil \log \Psi \rceil]$ all $O(1)$ cells C' of size $2^{i-1}|C|$ that intersect $(2^{i+5}+1) * C$. For each of the $O(\varepsilon^{-2})$ ε-cells E' of C', we query for each disk $\sigma \in \Gamma_\varepsilon(E)$ whether it intersects a disk σ' in $\pi_\varepsilon(E')$ in $O(\log^2 n)$ time. If so, then we add (σ, σ') to G and continue to some other ε-cell E''. Since each disk appears in exactly one subpopulation $\Gamma_\varepsilon(E)$ this takes $O(n\varepsilon^{-2} \log^2 n \log \Psi)$ total time. ◀

Dynamically maintaining our spanner

We show how to maintain G dynamically under disk insertions and deletions. We will maintain the type i and type ii edges independently. Maintaining the type i edges will be relatively straightforward, as it essentially only requires us to invoke the dynamic updates from Lemma 1. Maintaining the type ii edges will require more work.

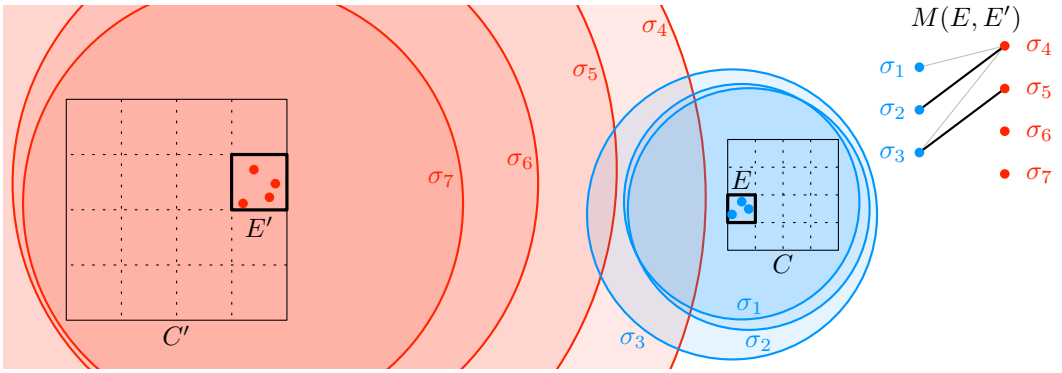

Figure 8 The bichromatic disk intersection graph of disks in $\Gamma_\varepsilon(E)$ (blue) and $\pi_\varepsilon(E')$ (red) and the maximal matching $M(E, E')$ (black). $Q(E, E')$ contains σ_1, and $Q'(E, E')$ contains σ_6 and σ_7.

Following the definition of type ii edges, let us consider pairs of ε-cells (E, E') defined as follows: for every cell C in the quadtree $T(\mathcal{S})$, for every $i \in [1, \lceil \log \Psi \rceil]$, and for every cell C' where $|C'| = 2^{i-1}|C|$ and C' intersects $(2^{i+5} + 1) * C$, consider pairs (E, E') where E is an ε-cell of C and E' is an ε-cell of C'. Recall that there is a type ii edge between E and E' if and only if there exists a pair of disks $(\sigma, \sigma') \in \Gamma_\varepsilon(E) \times \pi_\varepsilon(E')$ that intersect. Our approach will be to maintain a maximal matching in the bichromatic disk intersection graph of disks in $\Gamma_\varepsilon(E) \times \pi_\varepsilon(E')$. To this end, we will construct, separately for every pair of ε-cells (E, E'), the following three auxiliary data structures (see Figure 8).

▶ **Definition 15.** *For every pair of ε-cells (E, E'), where for their quadtree cells C and C' it holds that $|C'| = 2^{i-1}|C|$ for some $i \in [1, \lceil \log \Psi \rceil]$ and C' intersects $(2^{i+5} + 1) * C$, we:*

- *Store a maximal bichromatic matching $M(E, E')$ of the bipartite graph $\Gamma_\varepsilon(E) \times \pi_\varepsilon(E')$, where there is an edge $(\sigma, \sigma') \in \Gamma_\varepsilon(E) \times \pi_\varepsilon(E')$ if and only if σ and σ' intersect.*
- *Store a disk intersection query structure $Q(E, E')$, defined in Lemma 6, for all disks in $\Gamma_\varepsilon(E)$ that do not appear in the matching $M(E, E')$.*
- *Store a disk intersection query structure $Q'(E, E')$, defined in Lemma 6, for all disks in $\pi_\varepsilon(E')$ that do not appear in the matching $M(E, E')$.*

Now that we have set up the three auxiliary structures, we are ready to prove the main result of the section, which is to dynamically maintain the spanner G.

▶ **Lemma 9.** *Let $\mathcal{S}$ be a fully dynamic set of disks in the plane with diameters in $[4, \Psi]$ for a fixed and known Ψ. For any fixed $\varepsilon \in (0, 1)$, we can maintain a $(1 + \varepsilon)$-spanner of $\mathcal{D}(\mathcal{S})$ of size $O(n\varepsilon^{-2} \log \Psi \log(\varepsilon^{-1}))$ using $O((\frac{\Psi}{\varepsilon})^2 \log^4 n \log \Psi \log^2(\varepsilon^{-1}))$ expected amortised update time and $O((\frac{\Psi}{\varepsilon})^2 n \log n \log \Psi \log(\varepsilon^{-1}))$ space.*

Proof. We assume that $\mathcal{S}$ remains inside $[0, \Psi^*]^2$ (we remove this assumption in the full version of the paper [29]). As we maintain the same spanner as in Lemma 13, the spanner size bound follows.

Notation. Consider the quadtree cells of at least unit size obtained by recursively splitting the bounding box. We let $\mathcal{C}$ be the set of pairs of such cells (C, C') satisfying: $|C| = |C'|$ and $C' \subset 3 * C$. Recall that $Span(C, C')$ is a dynamic Euclidean spanner on the points $\pi(C) \cup \pi(C')$. Let $\mathcal{E}$ be the set of pairs of ε-cells (E, E') satisfying: E is an ε-cell of C, E' is an ε-cell of C', and there is an $i \in [1, \lceil \log \Psi \rceil]$ such that $|C'| = 2^{i-1}|C|$ and $C' \subset (2^{i+5} + 1) * C$.

Disk insertions. When inserting a disk σ into $\mathcal{S}$ we perform the following operations:

1. We consider all quadtree cells, down to unit size, that are contained in σ and contain the centre of σ. This is the storing family $\mathcal{F}(\sigma)$ of σ. For each cell $C \in \mathcal{F}(\sigma)$ we either identify C in $T(\mathcal{S})$ or add C to $T(\mathcal{S})$.
2. We consider all quadtree cells $C \in \mathcal{F}(\sigma)$. For every quadtree cell C' such that $(C, C') \in \mathcal{C}$, we insert the centre of σ into the Euclidean spanner $\mathit{Span}(C, C')$.
3. We consider the unique storing cell C_σ of σ and the unique ε-cell E of C_σ that contains the centre of σ. We consider all E' such that $(E, E') \in \mathcal{E}$ and test whether σ intersects a disk σ' in $\pi_\varepsilon(E')$ that is not already in the matching $M(E, E')$ by querying $Q'(E, E')$.
   - If so, we add (σ, σ') to $M(E, E')$ and delete σ' from $Q'(E, E')$.
   - Otherwise, we add σ to $Q(E, E')$.
4. To better match Definition 15, we now add a prime to our notation and consider all $C' \in \mathcal{F}(\sigma)$. There is a unique ε-cell E' of C' that contains the centre of σ. We consider all ε-cells E such that $(E, E') \in \mathcal{E}$. We test whether σ intersects a disk σ'' in $\Gamma_\varepsilon(E)$ that is not already in the maximal bichromatic matching $M(E, E')$ by querying $Q(E, E')$.
   - If so, we add (σ'', σ) to $M(E, E')$ and delete σ'' from $Q(E, E')$.
   - Otherwise, we add σ to $Q'(E, E')$.
5. For every edge updated in any Euclidean spanner $\mathit{Span}(C, C')$, we update the corresponding type i edge in G and whenever an empty bichromatic matching $M(E, E')$ becomes non-empty, we also add the corresponding type ii edge to G.

Disk deletions. When deleting a disk σ from $\mathcal{S}$ we perform the following operations:

1. We consider all quadtree cells, down to unit size, that are contained in σ and contain the centre of σ. This is the storing family $\mathcal{F}(\sigma)$ of σ. If every cell $C \in T(\mathcal{S})$ records $|\pi(C)|$, then we can simply iterate over all cells $C \in \mathcal{F}(\sigma)$ and decrement this counter. At the end of the procedure, we then remove all cells $C \in F(\sigma)$ for which we set the counter to zero, to update $T(\mathcal{S})$.
2. For all cells $C \in \mathcal{F}(\sigma)$, for every quadtree cell C' such that $(C, C') \in \mathcal{C}$, we delete the centre of σ from the Euclidean spanner $\mathit{Span}(C, C')$.
3. We consider the unique storing cell C_σ of σ and its ε-cell E that contains its centre. We consider all E' such that $(E, E') \in \mathcal{E}$ and query whether σ is in the matching $M(E, E')$.
   - If there is an edge $(\sigma, \sigma') \in M(E, E')$, we delete (σ, σ') from the matching. To maintain a maximal matching, we need to test whether we can re-match σ'. We query whether σ' intersects a disk σ'' in $\Gamma_\varepsilon(E)$ that is not already in $M(E, E')$ by querying $Q(E, E')$.
     - If so, we add (σ'', σ') to $M(E, E')$ and delete σ'' from $Q(E, E')$.
     - Otherwise, we insert σ' into $Q'(E, E')$.
   - Otherwise, if there is no edge $(\sigma, \sigma') \in M(E, E')$ then we must delete σ from $Q(E, E')$.
4. To better match Definition 15, we now add a prime to our notation and consider all $C' \in \mathcal{F}(\sigma)$. There is a unique ε-cell E' of C' which contains the centre of σ. We consider all ε-cells E such that $(E, E') \in \mathcal{E}$ and query whether σ is in the matching $M(E, E')$.
   - If there is an edge (σ'', σ) in $M(E, E')$ then we delete (σ'', σ) from the matching. To maintain a maximal matching, we consider re-matching σ'' by querying whether σ'' intersects a disk $\sigma' \in \pi_\varepsilon(E')$ that is not already in $M(E, E')$, by querying $Q'(E, E')$.
     - If so, then we add (σ'', σ') to $M(E, E')$ and delete σ' from $Q'(E, E')$.
     - Otherwise, we insert σ'' into $Q(E, E')$.
   - If σ is not matched in $M(E, E')$ then we delete σ from $Q'(E, E')$.
5. For every updated Euclidean spanner $\mathit{Span}(C, C')$, we update the corresponding type i edge in G and whenever a matching $M(E, E')$ becomes empty (or, when its corresponding edge in G had σ as an endpoint) we update the corresponding type ii edge in G.

Update time. Let σ be either an inserted or a deleted disk. There are $O(\log \Psi)$ quadtree cells $C \in \mathcal{F}(\sigma)$. For each C there are $O(1)$ quadtree cells C' such that $(C, C') \in \mathcal{C}$ for which we maintain a Euclidean spanner $Span(C, C')$. Inserting or deleting a point from this spanner takes $O(\varepsilon^{-2} \log n \log^2(\varepsilon^{-1}))$ time. In total, updating the Euclidean spanners and the type i edges of G (step 2 and 5) takes $O(\varepsilon^{-2} \log n \log \Psi \log^2(\varepsilon^{-1}))$ time per update in $\mathcal{S}$.

We note that steps 3 and 4 execute the same procedure, but the more expensive operation is step 4 which first iterates over all $O(\log \Psi)$ cells $C' \in \mathcal{F}(\sigma)$. Every such cell C' has a unique ε-cell E' containing the centre of σ, and for each fixed E', there are $O(\Psi^2 \varepsilon^{-2})$ ε-cells E' such that $(E, E') \in \mathcal{E}$. Indeed, the definition of $\mathcal{E}$ involves a pair of quadtree cells (C, C') where C is smaller than C'. For a fixed quadtree cell, C', for the bottom level of the quadtree (the level where all quadtree cells have unit size) there are $O(\Psi^2)$ cells that can lie within distance $O(\Psi)$ from C (every such cell C generates at most $O(\varepsilon^{-2})$ pairs $(E, E') \in \mathcal{E}$). For the level above, there are half as many, and so by a geometric series we obtain that in Step 4 there are $O(\Psi^2 \varepsilon^{-2})$ pairs (E, E'). For each pair (E, E') we perform $O(1)$ queries and updates on the three auxiliary data structures. The running time for these queries and updates is dominated by the $O(\log^4 n)$ amortised expected time to update $Q(E, E')$ and $Q'(E, E')$ (Lemma 6). In total, updating $M(E, E')$, $Q(E, E')$, $Q'(E, E')$ and the type ii edges of G (step 3, 4, and 5) takes $O(\Psi^2 \varepsilon^{-2} \log^4 n \log \Psi)$ expected amortised time per update.

Space usage. We analyse the space usage of the three main components (the Euclidean spanner, the maximal bichromatic matchings, and the intersection data structures) separately.

The size and space usage of the Euclidean spanner are asymptotically the same. As before, the total space of all Euclidean spanners is thus $O(n\varepsilon^{-2} \log \Psi \log(\varepsilon^{-1}))$. Next, we analyse the total size of all bichromatic matchings. Consider a cell C in the quadtree and an ε-cell E of C. There are $O(\varepsilon^{-2} \log \Psi)$ cells E' such that $(E, E') \in \mathcal{E}$. We thus consider only $O(\varepsilon^{-2} \log \Psi)$ maximal bichromatic matchings for E. As each disk appears in the subpopulation $\Gamma_\varepsilon(E)$ of exactly one ε-cell E, the total number of edges in all matchings is $O(n\varepsilon^{-2} \log \Psi)$.

Finally, we analyse the size of the disk intersection data structures. A disk appears in disk intersection structures $Q'(E, E')$ for $O(\log \Psi)$ ε-cells E', at most once per quadtree level. For such an ε-cell E', there are $O(\Psi^2 \varepsilon^{-2})$ ε-cells E for which we store a $Q'(E, E')$ data structure. Note that again that by a geometric series over all quadtree levels, it follows that for a fixed ε-cell E' there are a total of $O(\Psi^2 \varepsilon^{-2})$ ε-cells E for which we store a $Q'(E, E')$ data structure. As the size of the disk intersection data structure is $O(n \log n)$, the total size of all of the $Q'(E, E')$ data structures is $O(n\Psi^2 \varepsilon^{-2} \log n \log \Psi)$. The total space of the $Q(E, E')$ data structures is dominated by the $Q'(E, E')$ data structures. ◀

References

1 Mohammad Ali Abam, Marjan Adeli, Hamid Homapour, and Pooya Zafar Asadollahpoor. Geometric spanners for points inside a polygonal domain. In *Symposium on Computational Geometry (SoCG)*, LIPIcs, pages 186–197. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.SOCG.2015.186.

2 Mohammad Ali Abam, Mark de Berg, and Mohammad Javad Rezaei Seraji. Geodesic spanners for points on a polyhedral terrain. *SIAM Journal on Computing (SICOMP)*, 48(6):1796–1810, 2019. doi:10.1137/18M119358X.

3 Ingo Althöfer, Gautam Das, David P. Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete Computational Geometry (DCG)*, 9:81–100, 1993. doi:10.1007/BF02189308.

4 Shinwoo An, Eunjin Oh, and Jie Xue. Single-source shortest path problem in weighted disk graphs. In *International Symposium on Computational Geometry (SoCG)*, pages 7:1–7:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.SOCG.2025.7.

5 Sunil Arya, Gautam Das, David M. Mount, Jeffrey S. Salowe, and Michiel H. M. Smid. Euclidean spanners: short, thin, and lanky. In *ACM Symposium on Theory of Computing (STOC)*, pages 489–498. ACM, 1995. doi:10.1145/225058.225191.

6 Surender Baswana, Sumeet Khurana, and Soumojit Sarkar. Fully dynamic randomized algorithms for graph spanners. *ACM Transactions on Algorithms (TALG)*, 8(4):35:1–35:51, 2012. doi:10.1145/2344422.2344425.

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

8 Mark de Berg and Sergio Cabello. An O(n log n) algorithm for single-source shortest paths in disk graphs. In *European Symposium on Algorithms (ESA)*, LIPIcs, pages 81:1–81:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ESA.2025.81.

9 Sujoy Bhore, Balázs Keszegh, Andrey Kupavskii, Hung Le, Alexandre Louvet, Dömötör Pálvölgyi, and Csaba D. Tóth. Spanners in planar domains via Steiner spanners and non-Steiner tree covers. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4292–4326. SIAM, 2025. doi:10.1137/1.9781611978322.145.

10 Ahmad Biniaz. Plane hop spanners for unit disk graphs: Simpler and better. *Computational Geometry*, 89:101622, 2020. doi:10.1016/J.COMGEO.2020.101622.

11 Glencora Borradaile, Hung Le, and Christian Wulff-Nilsen. Minor-free graphs have light spanners. In *IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 767–778. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.76.

12 Bruce W. Brewer and Haitao Wang. An optimal algorithm for shortest paths in unweighted disk graphs. In Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman, editors, *European Symposium on Algorithms (ESA)*, LIPIcs, pages 31:1–31:8. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ESA.2025.31.

13 Fraser Cadger, Kevin Curran, Jose A. Santos, and Sandra Moffett. A survey of geographical routing in wireless ad-hoc networks. *IEEE Communications Surveys and Tutorials*, 15(2):621–653, 2013. doi:10.1109/SURV.2012.062612.00109.

14 Paul B. Callahan and S. Rao Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM (JACM)*, 42(1):67–90, 1995. doi:10.1145/200836.200853.

15 Nicolas Catusse, Victor Chepoi, and Yann Vaxès. Planar hop spanners for unit disk graphs. In *Workshop on Algorithms for Sensor Systems, Wireless Ad Hoc Networks, and Autonomous Mobile Entities (ALGOSENSORS)*, pages 16–30, 2010. doi:10.1007/978-3-642-16988-5_2.

16 T.-H. Hubert Chan, Anupam Gupta, Bruce M. Maggs, and Shuheng Zhou. On hierarchical routing in doubling metrics. *ACM Transactions on Algorithms (TALG)*, 12(4):55:1–55:22, 2016. doi:10.1145/2915183.

17 Timothy Chan. Personal communication.

18 Timothy M Chan. Dynamic geometric data structures via shallow cuttings. *Discrete and Computational Geometry (DCG)*, 64(4):1235–1252, 2020. doi:10.1007/S00454-020-00229-5.

19 Timothy M Chan, Sariel Har-Peled, and Mitchell Jones. On locality-sensitive orderings and their applications. *SIAM Journal on Computing (SICOMP)*, 49(3):583–600, 2020. doi:10.1137/19M1246493.

20 Timothy M. Chan and Zhengcheng Huang. Constant-hop spanners for more geometric intersection graphs, with even smaller size. In *Symposium on Computational Geometry (SoCG)*, volume 258 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:16, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2023.23.

21 Timothy M. Chan and Zhengcheng Huang. Dynamic geometric connectivity in the plane with constant query time. In *International Symposium on Computational Geometry (SoCG)*, pages 36:1–36:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.SOCG.2024.36.

22 Timothy M. Chan and Dimitrios Skrepetos. Approximate shortest paths and distance oracles in weighted unit-disk graphs. *Journal of Computational Geometry (JoCG)*, 10(2):3–20, 2019. doi:10.20382/JOCG.V10I2A2.

23 Julia Chuzhoy and Merav Parter. Fully dynamic algorithms for graph spanners via low-diameter router decomposition. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 785–823. SIAM, 2025. doi:10.1137/1.9781611978322.23.

24 Brent N Clark, Charles J Colbourn, and David S Johnson. Unit disk graphs. *Discrete mathematics*, 86(1-3):165–177, 1990. doi:10.1016/0012-365X(90)90358-O.

25 Kenneth L. Clarkson. Approximation algorithms for shortest path motion planning (extended abstract). In *ACM Symposium on Theory of Computing (STOC)*, pages 56–65. ACM, 1987. doi:10.1145/28395.28402.

26 Jonathan B. Conroy and Csaba D. Tóth. Hop-spanners for geometric intersection graphs. In *International Symposium on Computational Geometry (SoCG)*, pages 30:1–30:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SOCG.2022.30.

27 Jonathan B. Conroy and Csaba D. Tóth. Hop-spanners for geometric intersection graphs. In *Symposium on Computational Geometry (SoCG)*, LIPIcs, pages 30:1–30:17, 2022. doi:10.4230/LIPIcs.SoCG.2022.30.

28 Jorge Cortés, Sonia Martínez, and Francesco Bullo. Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions. *IEEE Transactions on Automatic Control*, 51(8):1289–1298, 2006. doi:10.1109/TAC.2006.878713.

29 Sarita de Berg, Ivor van der Hoog, Eva Rotenberg, Johanne M. Vistisen, and Sampson Wong. A dynamic $(1+\varepsilon)$-spanner for disk intersection graphs, 2026. arXiv:2604.25397.

30 James R Driscoll, Neil Sarnak, Daniel Dominic Sleator, and Robert Endre Tarjan. Making data structures persistent. In *ACM Symposium on Theory of Computing (STOC)*, pages 109–121, 1986. doi:10.1145/12130.12142.

31 Adrian Dumitrescu, Anirban Ghosh, and Csaba D. Tóth. Sparse hop spanners for unit disk graphs. *Computational Geometry*, page 101808, 2022. doi:10.1016/j.comgeo.2021.101808.

32 Martin Fürer and Shiva Prasad Kasiviswanathan. Spanners for geometric intersection graphs with applications. *Journal of Computational Geometry (JoCG)*, 3(1):31–64, 2012. doi:10.20382/JOCG.V3I1A3.

33 Jie Gao, Leonidas J. Guibas, John Hershberger, Li Zhang, and An Zhu. Geometric spanners for routing in mobile networks. *IEEE Journal on Selected Areas in Communications (JSAC)*, 23(1):174–185, 2005. doi:10.1109/JSAC.2004.837364.

34 Jie Gao and Li Zhang. Well-separated pair decomposition for the unit-disk graph metric and its applications. *SIAM Journal on Computing (SICOMP)*, 35(1):151–169, 2005. doi:10.1137/S0097539703436357.

35 Lee-Ad Gottlieb and Liam Roditty. An optimal dynamic spanner for doubling metric spaces. In *European Symposium on Algorithms (ESA)*, Lecture Notes in Computer Science, pages 478–489. Springer, 2008. doi:10.1007/978-3-540-87744-8_40.

36 Michelangelo Grigni and Papa A. Sissokho. Light spanners and approximate TSP in weighted graphs with forbidden minors. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 852–857. ACM/SIAM, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545492.

37 Joachim Gudmundsson, Christos Levcopoulos, and Giri Narasimhan. Fast greedy algorithms for constructing sparse geometric spanners. *SIAM Journal on Computing (SICOMP)*, 31(5):1479–1500, 2002. doi:10.1137/S0097539700382947.

38 Sariel Har-Peled. *Geometric Approximation Algorithms*, volume 173 of *Mathematical Surveys and Monographs*. American Mathematical Society, Providence, RI, 2011. doi:10.1090/surv/173.

39 Sariel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM Journal on Computing (SICOMP)*, 35(5):1148–1184, 2006. doi:10.1137/S0097539704446281.

40 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM (JACM)*, 48(4):723–760, 2001. doi:10.1145/276698.276715.

41 Ivor van der Hoog, André Nusser, Eva Rotenberg, and Frank Staals. Fully-adaptive dynamic connectivity of square intersection graphs. In *International Symposium on Mathematical Foundations of Computer (MFCS)*, LIPIcs, pages 63:1–63:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.MFCS.2024.63.

42 Haim Kaplan, Matthew J. Katz, Rachel Saban, and Micha Sharir. The unweighted and weighted reverse shortest path problem for disk graphs. In *European Symposium on Algorithms (ESA)*, pages 67:1–67:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ESA.2023.67.

43 Haim Kaplan, Katharina Klost, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Triangles and girth in disk graphs and transmission graphs. In *European Symposium on Algorithms (ESA)*, pages 64:1–64:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.64.

44 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 (DCG)*, 64(3):838–904, 2020. doi:10.1007/S00454-020-00243-7.

45 Sándor Kisfaludi-Bak and Geert van Wordragen. A quadtree, a Steiner spanner, and approximate nearest neighbours in hyperbolic space. *Journal of Computational Geometry (JoCG)*, 16(2):145–174, 2024. doi:10.20382/JOCG.V16I2A5.

46 Sándor Kisfaludi-Bak and Geert van Wordragen. Near-optimal dynamic Steiner spanners for constant-curvature spaces. In *Symposium on Computational Geometry (SoCG)*, 2026.

47 Katharina Klost. An algorithmic framework for the single source shortest path problem with applications to disk graphs. *Computational Geometry*, 111:101979, 2023. doi:10.1016/J.COMGEO.2022.101979.

48 Hung Le and Shay Solomon. Truly optimal Euclidean spanners. *SIAM Journal on Computing (SICOMP)*, 54(4):S19–135, 2025. doi:10.1137/20M1317906.

49 Chih-Hung Liu. Nearly optimal planar k nearest neighbors queries under general distance functions. *SIAM Journal on Computing (SICOMP)*, 51(3):723–765, 2022. doi:10.1137/20M1388371.

50 Giri Narasimhan and Michiel Smid. *Geometric spanner networks*. Cambridge University Press, 2007. doi:10.1017/CBO9780511546884.

51 David Peleg and Alejandro A Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989. doi:10.1002/jgt.3190130114.

52 Micha Sharir and Pankaj K. Agarwal. *Davenport-Schinzel sequences and their geometric applications*. Cambridge University Press, 1995.

53 Ivor van der Hoog, Elena Khramtcova, and Maarten Löffler. Dynamic smooth compressed quadtrees. In *Symposium on Computational Geometry, (SoCG)*, volume 99, pages 45:1–45:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.SOCG.2018.45.

54 Chenyu Yan, Yang Xiang, and Feodor F. Dragan. Compact and low delay routing labeling scheme for unit disk graphs. In *Symposium on Algorithms and Data Structures (WADS)*, pages 566–577, 2009. doi:10.1007/978-3-642-03367-4_49.

55 Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing (SICOMP)*, 11(4):721–736, 1982. doi:10.1137/0211059.

A Achieving $\tilde{O}(n)$ space within a fixed bounding box

The data structure of Lemma 9 has a quadratic size-dependency on the maximum disk size Ψ. This is primarily due to the fact that the data structure requires for every ε-cell that stores a set of disks $\mathcal{S}'$, $O(\Psi^2\varepsilon^{-2})$ copies of an intersection data structure, each of which includes some subset of $\mathcal{S}'$. We argue that in many of these cases, these copies are near-identical. We show that we can obtain better space-bounds by making the data structure sensitive to the number of edges across all maximal bichromatic matchings instead.

We will achieve this through a form of data structure persistence. Persistence is traditionally a way to retain older versions of a dynamic data structure. *Partial persistence* allows queries to all previous versions of the data structure, but only allows updates to the latest version. *Full persistence* allows both queries and updates to any version of the data structure. There are various ways to make a data structure persistent, as described by Driscoll, Sarnak, Sleator, and Tarjan [30]. The type of persistence we require slightly differs from both partial and full persistence: it is more general than partial persistence, but because our auxiliary intersection data structures cannot natively be implemented on a pointer machine whilst maintaining constant in-degree, we cannot easily use full persistence. We therefore define an alternative type of persistence, called *branch persistence*. To be precise, our goal is to make the intersection data structure of Lemma 6 branch persistent.

We adapt the *fat node persistence* technique from [30], which is applicable to all data structures that work in the pointer machine model (in [30], this is called a *linked data structure*). When using fat node persistence, each node in the pointer machine is made *fat*, meaning that they not just store one value, but at most one value for each version that we keep track of. Whenever an update is performed on the node, the current value is not replaced, but a new value is added including a version stamp. This way, the each old version can still be accessed by searching for the corresponding version value in each node.

A.1 Branch persistence

Let $I(\mathcal{S})$ be a dynamic pointer machine data structure storing the n objects in $\mathcal{S}$. Instead of defining *versions*, which are typically used in persistence, we will distinguish between the *root* version of the data structure and different *branches*. Each branch i stores a subset $\mathcal{S}_i \subseteq \mathcal{S}$. A query then specifies a branch i, and performs the given query on the set $\mathcal{S}_i$. Similarly, an update specifies in which branch the update should be performed. To ensure that $\mathcal{S}_i$ remains a subset of $\mathcal{S}$, we only allow insertions of objects that are in $\mathcal{S}$. Additionally, one can perform a *root-update* that performs the update on the root that stores the data structure on $\mathcal{S}$ *and* then performs the same update on each branch. We illustrate this principle in Figure 9.

▶ **Definition 16.** *Given a dynamic pointer machine data structure $I(\mathcal{S})$ storing a (dynamic) set of n objects $\mathcal{S}$, a* branch persistent *version is a data structure where:*

- *the* root *(with label 0) corresponds to the current set $\mathcal{S}$,*
- *each* branch*, with integer label i, corresponds to a subset $S_i \subseteq \mathcal{S}$.*

Let B denote the current number of branches. The following operations are supported:

- query(i, ρ)*: perform a query with ρ on $I(\mathcal{S}_i)$.*
- branch-update$(i, \sigma,$ `Insert`$/$`Delete`*): if $\sigma \in \mathcal{S}$ insert σ into or delete σ from $\mathcal{S}_i$.*
- root-update$(\sigma,$ `Insert`$/$`Delete`*): insert σ into or delete σ from $\mathcal{S}$, then perform the same update on each branch i using a* branch-update*.*
- branch(i)*: if there is no branch i, create a new branch with label i that has $\mathcal{S}_i = \mathcal{S}$.*
- rebuild*: rebuild the data structure from scratch.*

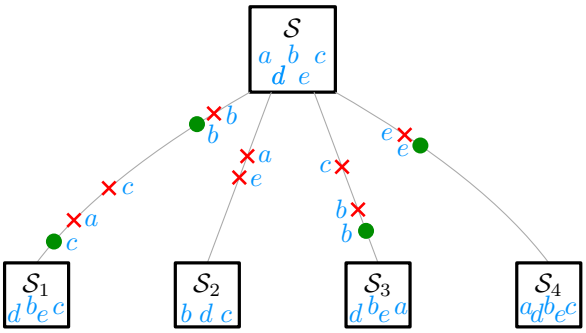

Figure 9 After creating four branches, and making the indicated *branch-updates* (each branch update either deletes or inserts the indicated object), the sets $\mathcal{S}_i$ consist of the indicated objects.

▶ **Lemma 17.** *A dynamic pointer machine data structure $I(\mathcal{S})$ with $Q(n)$ query time and $U(n) \in \Omega(\log n)$ update time can be made branch persistent, supporting B branches and with an upper bound of N on $|\mathcal{S}|$ between two rebuilds, such that:*

- Queries *are supported in $O(Q(N)\log B)$ time.*
- Root-updates *are supported in $O(U(N)B\log B)$ time and increase the space by $O(U(N)B)$.*
- Branch-updates *are supported in $O(U(N)\log B)$ time and increase the space by $O(U(N))$.*
- Rebuilds *are supported in $O((|\mathcal{S}|+z)U(|\mathcal{S}|)\log B)$ time, where z is the size of the symmetric difference $z := \sum_{i\in[B]} |\mathcal{S}-\mathcal{S}_i|$. The space after the rebuild is $O((|\mathcal{S}|+z)U(|\mathcal{S}|))$.*

Proof. We support branch persistence via the *fat node persistence* technique from [30]. The core idea is that each node in the pointer machine has up to B versions, at most one per branch. These versions are labelled and stored in a balanced tree so that given any integer i, one can find the corresponding version of that pointer node in $O(\log B)$ time. This "version lookup" follows the following principle to access $I(\mathcal{S}_i)$: let ν be a pointer machine node, if there exists a version ν_i of ν then read this version, otherwise, read ν_0. We maintain the *invariant* that reading the pointer machine in this manner recovers the data structure $I(\mathcal{S}_i)$ for all branches i. As a consequence, queries take $O(Q(N)\log B)$ time. As *branch*(i) creates a new branch with $\mathcal{S}_i = \mathcal{S}$, this operation is supported in constant time by incrementing B.

Updating the data structure. Immediately after creating a new branch i, the invariant holds as $\mathcal{S}_i = \mathcal{S}$. We support the remaining updates as follows.

To support *branch-update*$(i, \sigma,$ `Insert`/`Delete`$)$ we note that by our invariant, we can access $I(\mathcal{S}_i)$ with $O(\log B)$ overhead per operation. Because the data structure $I(\mathcal{S}_i)$ can be updated in $U(N)$ time, such an update changes values (or pointers) in $U(N)$ nodes in the pointer structure. For every such change to a node ν, we introduce (or overwrite) only ν_i representing its state in $I(\mathcal{S}_i)$. If ν did not exist before the update, we simply create a new node with version ν_i. A *branch-update* thus takes $U(N)\log B$ time and adds $U(N)$ space. As the update on branch i only changes ν_i for any node ν, the invariant is immediately maintained for all other branches which use a different integer to read their node versions.

To support *root-update*$(\sigma,$`Insert`/`Delete`$)$ we execute *branch-update*$(i,\sigma,$ `Insert`/`Delete`$)$ for all B branches, including the root branch, at a factor B overhead.

Rebuilding. To support rebuilds, we additionally maintain for all branches a balanced binary tree storing all objects in $\mathcal{S}$ that are *not* in $\mathcal{S}_i$, i.e. $\mathcal{S} \setminus \mathcal{S}_i$, in some arbitrary fixed order. We call this tree the *difference tree*. When updating $\mathcal{S}_i$, the difference tree can be updated in $O(\log N)$ time and, as we assumed that $U(n) \in \Omega(\log n)$, this does not incur any overhead. When we create a new branch, we create an empty difference tree in constant time.

A rebuild is executed as follows. First, we construct given $\mathcal{S}$ the root data structure $I(\mathcal{S})$ in $O(|\mathcal{S}|U(|\mathcal{S}|))$ time by performing an insertion for all $\sigma \in \mathcal{S}$. Then, for each old branch i, i.e. each branch that existed before the rebuild, we create a new branch in constant time. We then iterate over all elements σ in the corresponding difference tree and perform *branch-update*$(i, \sigma, \texttt{Delete})$. Since the sum of the difference tree sizes equals z, the total reconstruction time is $O((|\mathcal{S}|+z)U(|\mathcal{S}|)\log B)$ and the total space is $O((|\mathcal{S}|+z)U(|\mathcal{S}|))$. ◀

A.2 Applying branch persistence

We now apply branch persistence to the data structure maintaining our spanner, so that we no longer store $O(\Psi^2\varepsilon^{-2})$ copies of the disk intersection data structure for each ε-cell. Instead, we store a branch persistent data structure for each intersection query data structure. The data structure from Lemma 6 for intersection queries can be made branch persistent using amortised expected update time and *expected space* as follows.

▶ **Lemma 18.** *The data structure from Lemma 6, storing a dynamic set of disks $\mathcal{S}$ where between any two rebuilds $|\mathcal{S}| \leq N$, supports branch persistence such that:*

- *Queries are supported in $O(\log^2 N \log B)$ time.*
- *Let r and b be the number of* root- *and* branch-updates *since the last rebuild and let z be the size of the symmetric difference $z := \sum_i |\mathcal{S} - \mathcal{S}_i|$ at this rebuild. The rebuild and these updates take $O((\overline{n} + z + rB + b)\log^4 N \log B)$ expected total time and the expected space usage is $O((\overline{n} + z + rB + b)\log^4 N)$, where $\overline{n}$ denotes the size of $\mathcal{S}$ at the rebuild.*

Proof. We apply Lemma 17 to the dynamic disk intersection data structure of Lemma 6. The disk intersection data structure has $Q(n) = O(\log^2 n)$ query time and $U(n) = O(\log^4 n)$ amortised expected update time. Lemma 17 directly implies the stated query time of the branch persistent version of the data structure. The amortised expected insertion time of the data structure implies that k updates on an initially empty data structure take $O(k \log^4 k \log B)$ total expected time. Since each update introduces space proportional to the original update time, the space usage is expected $O(k \log^4 k)$. The stated running time and space after a rebuild and some updates then follow from Lemma 17. ◀

Auxiliary data structures. In our original data structure, we stored two dynamic intersection data structures $Q(E, E')$ and $Q'(E, E')$ (see Definition 15) for *every pair* of ε-cells (E, E'), where E' is bigger than E and is "close" to E, where "close" is defined relative to only the size of E'. The structure $Q(E, E')$ stored all disks in the subpopulation $\Gamma_\varepsilon(E)$ excluding disks that appeared in the maximal bichromatic matching $M(E, E')$, while $Q'(E, E')$ stored the disks in the population $\pi_\varepsilon(E')$, again excluding any disks in $M(E, E')$. In our augmented data structure, we will only store two data structures $\mathcal{Q}(E)$ and $\mathcal{Q}'(E)$ for each ε-cell E. These data structures are branch persistent, such that for each relevant pair (E, E') we can store an additional branch (if necessary) instead of an almost identical copy of the data structure. Formally, we maintain the following structures.

▶ **Definition 19.** *For each ε-cell E in the quadtree with non-empty population $\pi_\varepsilon(E)$ we store the following two auxiliary data structures:*

1. *$\mathcal{Q}(E)$ stores the subpopulation $\Gamma_\varepsilon(E)$ in the branch persistent data structure of Lemma 18.*
2. *$\mathcal{Q}'(E)$ stores the population $\pi_\varepsilon(E)$ in the branch persistent data structure of Lemma 18.*

Furthermore, for every pair of ε-cells (E, E'), where for their quadtree cells C and C' it holds that $|C'| = 2^{i-1}|C|$ for some $i \in [1, \lceil \log \Psi \rceil]$ and C' intersects $(2^{i+5}+1) * C$, we:

1. *Store a maximal bichromatic matching $M(E, E')$ of the bipartite graph $\Gamma_\varepsilon(E) \times \pi_\varepsilon(E')$, where there is an edge $(\sigma, \sigma') \in \Gamma_\varepsilon(E) \times \pi_\varepsilon(E')$ if and only if σ and σ' intersect.*
2. *If $M(E, E')$ is non-empty, create a branch in $\mathcal{Q}(E)$ storing $\{\sigma \in \Gamma_\varepsilon(E) \mid \sigma \notin M(E, E')\}$.*
3. *If $M(E, E')$ is non-empty, create a branch in $\mathcal{Q}'(E')$ storing $\{\sigma \in \pi_\varepsilon(E') \mid \sigma \notin M(E, E')\}$.*

Note that, like with the original data structure, we risk (locally) having $O(\varepsilon^{-2}\Psi^2)$ branches. However, the following lemma shows that this cannot occur too often, and that we can bound the *global* symmetric difference between all persistent data structures and their branches.

▶ **Lemma 20.** *Let $\mathcal{M}$ be the set of all edges across all bichromatic matchings for all pairs of ε-cells. Consider, for a fixed ε-cell E, the set of disks $\mathcal{S}^{\mathcal{Q}(E)}$ and $\mathcal{S}^{\mathcal{Q}'(E)}$ stored in $\mathcal{Q}(E)$ and $\mathcal{Q}'(E)$ respectively and denote by $z_E := \sum_i |\mathcal{S}^{\mathcal{Q}(E)} - \mathcal{S}_i^{\mathcal{Q}(E)}| + \sum_j |\mathcal{S}^{\mathcal{Q}'(E)} - \mathcal{S}_j^{\mathcal{Q}'(E)}|$ the sum over all symmetric differences across all branches of these data structures. Then we can upper bound the total symmetric difference as follows:*

$$z := \sum_{\text{all } \varepsilon\text{-cells } E} z_E = 2|\mathcal{M}| \in O(n\varepsilon^{-2} \log \Psi)$$

Proof. Fix some ε-cell E. For any branch i of $\mathcal{Q}(E)$, the symmetric difference between the set $\mathcal{S}^{\mathcal{Q}(E)}$ and $\mathcal{S}_i^{\mathcal{Q}(E)}$ are exactly those disks that are in the matching $\mathcal{M}(E, E')$ where E' is the ε-cell corresponding to branch i. Similarly, the symmetric difference between $\mathcal{S}^{\mathcal{Q}'(E)}$ and $\mathcal{S}_j^{\mathcal{Q}'(E)}$ are exactly those disks that are in the matching $\mathcal{M}(E'', E)$ where E'' is the ε-cell corresponding to branch j. It follows that an edge $(\sigma, \sigma') \in M(E, E') \subseteq \mathcal{M}$ is counted twice, once for z_E and once for $z_{E'}$. So, z equals $2|\mathcal{M}|$.

What remains is to upper bound $|\mathcal{M}|$. Consider a cell C in the quadtree $T(\mathcal{S})$ and an ε-cell E of C. We store edges across maximal bichromatic matchings between $\Gamma_\varepsilon(E) \times \pi_\varepsilon(E')$ for some "sufficiently close" ε-cell E'. In particular, each disk in σ appears in only one unique subpopulation $\Gamma_\varepsilon(E)$ which corresponds to the ε-cell E of its storing cell C. There are subsequently $O(\log \Psi)$ cells C' such that $|C'| = 2^{i-1}|C|$ for some $i \in [1, \lceil \log \Psi \rceil]$ and $C' \subset (2^{i+5}+1) * C$. For every such cell C', there are $O(\varepsilon^{-2})$ ε-cells E' for which we consider a maximal bichromatic matching between $\Gamma_\varepsilon(E) \times \pi_\varepsilon(E')$. It follows that each disk σ is part of some ordered pair $(\sigma, \sigma') \in \mathcal{M}$ at most $O(\varepsilon^{-2} \log \Psi)$ times which implies the lemma. ◀

▶ **Theorem 1.** *Let $\mathcal{S}$ be a fully dynamic set of disks in the plane with diameters in $[4, \Psi]$ for a fixed and known Ψ. For any fixed $0 < \varepsilon < 1$, we can maintain a $(1+\varepsilon)$-spanner of $\mathcal{D}(\mathcal{S})$ of size $O(n\,\varepsilon^{-2} \log \Psi \log(\varepsilon^{-1}))$ using $O((\frac{\Psi}{\varepsilon})^2 \log^4 n \log^2 \Psi \log^2(\varepsilon^{-1}))$ expected amortised update time and $O(n\,\varepsilon^{-2} \log^4 n \log \Psi)$ expected space.*

The complete proof of Theorem 1 is included in the full version of this paper [29]. This paper contains part of the proof, specifically a description of disk insertions.

As before, we leave the generalisation to the case without a bounding box to the full paper [29]. We thus assume that at all times $\mathcal{S}$ remains contained in a bounding box $[0, \Psi^*]^2$ where Ψ^* is the smallest power of two that is larger than Ψ.

Notation. We repeat the notation introduced in the proof of Lemma 9. Consider all quadtree cells that are at least unit size that can be obtained by recursively splitting the bounding box. We denote by $\mathcal{C}$ the set of all pairs of such cells (C, C') satisfying: $|C| = |C'|$ and $C' \subset 3 * C$. We denote by $\mathcal{E}$ the set of all pairs of ε-cells (E, E') satisfying: E is a ε-cell of C, E' is a ε-cell of C', and there is an $i \in [1, \lceil \log \Psi \rceil]$ such that $|C'| = 2^{i-1}|C|$ and $C' \subset (2^{i+5}+1) * C$. We use $M(E, E')$, $\mathcal{Q}(E)$ and $\mathcal{Q}'(E)$ as defined in Definition 19.

Disk insertions. Inserting a disk σ is done in the same way as in the original data structure. The difference is that we work on branches of the persistent data structure instead of copies of the disk intersection data structure. To be precise, we perform the following operations:

1. We first consider the storing family $\mathcal{F}(\sigma)$ of σ. For each cell $C \in \mathcal{F}(\sigma)$, we either identify the corresponding cell in $T(\mathcal{S})$ or add this cell to $T(\mathcal{S})$.
2. For all cells $C \in \mathcal{F}(\sigma)$, for every quadtree cell C' such that $(C, C') \in \mathcal{C}$, we insert the centre of σ into the Euclidean spanner $\mathit{Span}(C, C')$.
3. We consider the unique storing cell C_σ of σ and its ε-cell E that contains the centre of σ. We perform a *root-update*$(\sigma, \texttt{Insert})$ on $\mathcal{Q}(E)$ to insert σ in the root version and all branches. We then consider all pairs $(E, E') \in \mathcal{E}$.
   - Consider the data structure $\mathcal{Q}(E)$.
     - The ε-cell E' corresponds to a unique branch i of $\mathcal{Q}(E)$.
   - Similarly, consider the data structure $\mathcal{Q}'(E')$.
     - If the subpopulation $\Gamma_\varepsilon(E)$ was empty before adding σ, then $\mathcal{Q}'(E')$ does not have a branch corresponding to E yet. We add this branch and obtain its id j, storing it in the pair (E, E').
     - If $\Gamma_\varepsilon(E)$ was not empty before, then we instead obtain the branch id j from (E, E').

   We then test whether σ intersects a disk σ' in $\pi_\varepsilon(E')$ that is not already in the maximal bichromatic matching $M(E, E')$ by querying the branch j of $\mathcal{Q}'(E)$.
   - If so, then we add (σ, σ') to $M(E, E')$. We delete σ from the branch i of $\mathcal{Q}(E)$, and delete σ' from the branch j of $\mathcal{Q}'(E')$ – maintaining the invariant that these branches store all disks in $\Gamma_\varepsilon(E)$ or $\pi_\varepsilon(E')$ that do not appear in $M(E, E')$.
4. To match the notation of Definition 19, we now introduce primes in our notation and consider all quadtree cells $C' \in \mathcal{F}(\sigma)$. A fixed cell C' has a unique ε-cell E' which contains the centre of σ. We perform *root-update*$(\sigma, \texttt{Insert})$ on $\mathcal{Q}'(E')$ to insert σ in the root version and all branches. We then consider all pairs $(E, E') \in \mathcal{E}$:
   - Consider the data structure $\mathcal{Q}(E)$.
     - If the population $\pi_\varepsilon(E')$ was empty before adding σ, then $\mathcal{Q}(E)$ does not have a branch corresponding to E' yet. We add this branch and obtain its id i and store it in the pair (E, E').
     - If $\pi_\varepsilon(E')$ was not empty before, then we instead obtain the branch id i from (E, E').
   - Similarly, consider the ε-cell E' and $\mathcal{Q}'(E')$.
     - Then E corresponds to a unique branch j of $\mathcal{Q}'(E')$ which we obtain from (E, E').

   We test whether σ intersects a disk σ'' in $\Gamma_\varepsilon(E)$ that is not already in the maximal bichromatic matching $M(E, E')$ by querying the branch i of $\mathcal{Q}(E)$.
   - If so, then we add (σ'', σ) to the matching $M(E, E')$. We delete σ from the branch j of $\mathcal{Q}'(E')$, and we delete σ'' from the branch i of $\mathcal{Q}(E)$ – maintaining the invariant that these branches store all disks in $\Gamma_\varepsilon(E)$ or $\pi_\varepsilon(E')$ that do not appear in $M(E, E')$.
5. For every edge updated in any Euclidean spanner $\mathit{Span}(C, C')$, we update the corresponding type i edge in G. Similarly, whenever an empty bichromatic matching becomes non-empty, we add the corresponding type ii edge to G.

Disk deletion, space usage, and update time. For the complete proof of Theorem 1, see the full paper [29].