Fully-Adaptive Dynamic Connectivity of Square Intersection Graphs

Ivor van der Hoog
Technical University of Denmark, Lyngby, Denmark

André Nusser
CNRS, Inria, I3S, Université Côte d'Azur, France

Eva Rotenberg
Technical University of Denmark, Lyngby, Denmark

Frank Staals
Utrecht University, The Netherlands

Abstract

A classical problem in computational geometry and graph algorithms is: given a dynamic set $\mathcal{S}$ of geometric shapes in the plane, efficiently maintain the connectivity of the intersection graph of $\mathcal{S}$. Previous papers studied the setting where, before the updates, the data structure receives some parameter P. Then, updates could insert and delete disks as long as at all times the disks have a diameter that lies in a fixed range $[\frac{1}{P}, 1]$. As a consequence of that prerequisite, the aspect ratio ψ (i.e. the ratio between the largest and smallest diameter) of the disks would at all times satisfy $\psi \leq P$. The state-of-the-art for storing disks in a dynamic connectivity data structure is a data structure that uses $O(Pn)$ space and that has amortized $O(P \log^4 n)$ expected amortized update time. Connectivity queries between disks are supported in $O(\log n / \log \log n)$ time.

In the dynamic setting, one wishes for a more flexible data structure in which disks of any diameter may arrive and leave, independent of their diameter, changing the aspect ratio freely. Ideally, the aspect ratio should merely be part of the analysis. We restrict our attention to axis-aligned squares, and study fully-dynamic square intersection graph connectivity. Our result is fully-adaptive to the aspect ratio, spending time proportional to the current aspect ratio ψ, as opposed to some previously given maximum P. Our focus on squares allows us to simplify and streamline the connectivity pipeline from previous work. When n is the number of squares and ψ is the aspect ratio after insertion (or before deletion), our data structure answers connectivity queries in $O(\log n / \log \log n)$ time. We can update connectivity information in $O(\psi \log^4 n + \log^6 n)$ amortized time. We also improve space usage from $O(P \cdot n \log n)$ to $O(n \log^3 n \log \psi)$ – while generalizing to a fully-adaptive aspect ratio – which yields a space usage that is near-linear in n for any polynomially bounded ψ.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Theory of computation → Computational geometry

Keywords and phrases Computational geometry, planar geometry, data structures, geometric intersection graphs, fully-dynamic algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.63

Funding *Ivor van der Hoog*: Received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 899987.
André Nusser: Supported by the French government through the France 2030 investment plan managed by the National Research Agency (ANR), as part of the Initiative of Excellence of Université Côte d'Azur under reference number ANR-15-IDEX-01.
Eva Rotenberg: Supported by the Independent Research Fund Denmark grant 2020-2023 (9131-00044B) "Dynamic Network Analysis" and the Carlsberg Foundation Young Researcher Fellowship CF21-0302 "Graph Algorithms with Geometric Applications".

1 Introduction

Geometric intersection graphs are one of the most well-studied geometrically-flavoured graph classes: Their nodes are geometric shapes, and an edge between two such shapes exists if and only if they intersect. This makes the description complexity of a geometric intersection graph very compact; it is linear in the number of objects, while the underlying graph potentially has a quadratic number of edges. In this work, we consider square intersection graphs in the dynamic setting. Intersection graphs are one of the few examples of dynamic graphs where fully-dynamic insertion and deletion of vertices is motivated and interesting. Since a vertex can come or leave with $\Theta(n)$ edges, applying any existing edge-updatable dynamic graph algorithm in a blackbox manner would lead to $\Omega(n)$ update time. Yet, the geometric nature of these graphs often allows for sublinear or even polylogarithmic update times.

A classical problem in dynamic graph algorithms is dynamic connectivity. In this problem, we want to maintain a data structure under edge or node insertions and deletions that allows for fast queries that return whether a given pair of vertices is connected. Connectivity was one of the first problems to be studied in the dynamic setting dating back to the 80s [11, 25], and has received ample attention ever since. Dynamic connectivity in general graphs has been studied in many settings; randomised or deterministic, amortised or worst-case [11, 13, 29, 18, 24], and the partially dynamic incremental or decremental settings [26, 27, 28, 1]. Due to its fundamental nature, and its many applications, dynamic connectivity has also received much attention for simpler graph classes. Examples of such graph classes include trees [25, 3], planar graphs [10, 21], and graphs of bounded genus [9, 15].

Naturally, the dynamic connectivity problem also drew attention for the class of geometric intersection graphs. This setting is particularly interesting as a single node insertion can drastically change the number of connected components, as it can introduce a linear number of new edges. On the other hand, the geometric structure can be exploited in the data structures. The first result for geometric connectivity in geometric intersection graphs with update and query time independent of the object diameters is by Chan et al. [7] who presents a dynamic (Euclidean) disk intersection data structure with an update time of $O(n^{20/21+\varepsilon})$ and a query time of $O(n^{1/7+\varepsilon})$. This has recently been improved to $O(n^{7/8})$ amortized update time with constant query time [6]. As progress seemed difficult in this setting, the setting in which the disks in the data structure have restricted diameters was considered. For a fixed diameter range, where there is some value P given in advance and diameters have to be contained in an interval $[\frac{1}{P}, 1]$, Kaplan et al. [17] showed that there is a data structure with expected amortized $O(P^2 \log^{10} n)$ update time, query time $O(\log n / \log\log n)$, using $O(nP \log n)$ space. Recently, Kaplan et al. [16] improved this to $O(P \log^7 n)$ expected amortized update time with the same update time and $O(nP)$ space.

From disks to squares. While the above works are stated for Euclidean disks, we note that the approach in [16] can be combined with [30] to obtain a data structure that works for the simpler setting of connectivity between axis-aligned squares. The obtained update time is amortized $O(P \log^4 n)$. Disk intersection graphs are often motivated by communication networks where the disks are interpreted as some sort of transmission diameter. This is an idealization of a complicated physical process and actual ad-hoc communication networks do not correspond to perfectly circular disks [20]. Thus, it is reasonable to switch to a different metric for computational reasons while maintaining the core idea of the underlying problem. Recently, similar progress was made for computing a single-source shortest path tree in an intersection graph by assuming square regions instead of disks [19].

Table 1 Complexities are asymptotic. All update times are amortized. The query time for all approaches is $O(\log n/\log\log n)$. $\lambda_s(n)$ denotes the maximum length of a Davenport-Schinzel sequence of order s on n symbols.

Object	Aspect ratio	Update time	Space	Ref.
disks	fixed $[\frac{1}{P}, 1]$	$P \cdot \log^7 n \cdot \lambda_6(\log n)$ exp.	nP	[16]
disks	fixed $[\frac{1}{P}, 1]$	$P \cdot \log^4$ exp.	nP	[16]+[22]
squares	fixed $[\frac{1}{P}, 1]$	$P \cdot \log^4 n$	nP	[16]+[30]
squares	adaptive ψ	$\psi \log^4 n + \log^6 n$	$n \log^3 n \log \psi$	Thm 10

We study the dynamic connectivity problem where the input S is a set of (axis-aligned) squares, while being fully adaptive to their aspect ratio. We let ψ denote the adaptive aspect ratio. Formally, if S is the input before an update and S' is the input after an update we define $\psi = \max\{\frac{\max_{\sigma\in S}|\sigma|}{\min_{\sigma'\in S}|\sigma'|}, \frac{\max_{\sigma\in S'}|\sigma|}{\min_{\sigma'\in S'}|\sigma'|}\}$. Our date structure maintains connectivity between squares with $O(\psi \log^4 n + \log^6 n)$ update time, $O(\log n/\log\log n)$ query time, and using $O(n \log^3 n \log \psi)$ space, see Table 1 for a comparison. our approach only requires near-linear space while maintaining near-linear update time and polylogarithmic query time.

Implications of adaptivity and our reduced space usage. To understand the implications of the adaptivity of our new solution, consider the scenario where the set of squares starts with a square A with diameter 1 and B with diameter $\frac{1}{n}$. Now, suppose the sequence of updates first removes B, then inserts a sequence of n squares of diameter 1, and finally reinserts B. Previous work [16] would require as input the interval $[\frac{1}{n}, 1]$ and the promise that all updates only insert squares in this interval. The space usage and the total update time of [16] is quadratic. In our case, since for almost all updates, the aspect ratio is constant. Moreover, the space usage and total update time is near-linear.

2 Problem statement and technical overview

Let $\mathcal{S} \subset \mathbb{R}^2$ be a set of axis-aligned squares. The intersection graph $G[\mathcal{S}]$ is the graph with vertex set $\mathcal{S}$ and with an edge between squares $\sigma, \sigma' \in \mathcal{S}$ whenever they intersect. We say that two squares σ, σ' are *connected* if there exists a path between their corresponding vertices in $G[\mathcal{S}]$. The set $\mathcal{S}$ is a fully dynamic set subject to (adversarial) insertions and deletions of squares. We wish to maintain $\mathcal{S}$ in a data structure supporting *connectivity queries* between squares in $\mathcal{S}$. We denote the diameter of square $\sigma \in \mathcal{S}$ by $|\sigma|$. We consider three settings with different restrictions on the square diameters in $\mathcal{S}$:

- The *fixed diameter range* setting. Here, the input specifies some P and each $\sigma \in \mathcal{S}$ has a diameter in $[\frac{1}{P}, 1]$ for some fixed P.
- The *bounded aspect ratio* setting. Here, the input specifies some P and at all times, for all $\sigma, \sigma' \in \mathcal{S}$ we have $\frac{|\sigma|}{|\sigma'|} \leq P$.
- The *adaptive aspect ratio* ψ setting in which arbitrary insertions and deletions may occur. Let S be the set of squares before an update and S' be the set of squares after an update. We define $\psi = \max\{\frac{\max_{\sigma\in S}|\sigma|}{\min_{\sigma'\in S}|\sigma'|}, \frac{\max_{\sigma\in S'}|\sigma|}{\min_{\sigma'\in S'}|\sigma'|}\}$ the aspect ratio relevant for the update.

We measure the algorithmic complexity in $n := |\mathcal{S}|$ and ψ, where n is the present size of the dynamic set $\mathcal{S}$. At all times, we maintain some minimal axis-aligned square F that contains $\mathcal{S}$, and the coordinates of F are powers of 2.

Results. The data structure of Kaplan et al. [16], when adapted to axis-aligned squares by applying Range Trees [30], can store a dynamic set of axis-aligned squares as follows: The input specifies some value P and all squares have fixed diameter range with ratio P. Their solution uses $O(nP)$ space and supports updates to $\mathcal{S}$ in $O(P \log^4 n)$ amortized time (Table 1). They answer connectivity queries in $O(\log n / \log \log n)$ worst-case time.

There are several reasons why [17] uses $O(nP)$ space and why their update bound cannot depend on the adaptive ψ instead of P (which we discuss further down). We present an adaption of their work that relies on the fact that $\mathcal{S}$ is a set of axis-aligned squares. Under this assumption, we adapt their data structure to work for adaptive ψ. We improve the space usage to near linear in n and $\log \psi$. In full generality, we also improve the performance: allowing for update times proportional to the density around the update σ. More concretely, we parameterize the runtime by the size of two sets $\mathcal{C}(\sigma)$ and $\mathcal{P}(\sigma)$. Intuitively, these sets contain squares in σ or squares around σ. These sets have size at most $O(\min\{\psi, n\})$.

Our result is a technical contribution, that examines and refines the data structure in [16] in the special case where $\mathcal{S}$ is a set of axis-aligned squares. To detail our contribution, we now present a technical overview, where we reference several concepts whose formal definitions are presented in their respective sections. The core component is a quadtree that stores $\mathcal{S}$.

The existing pipeline. We can describe the data structure of [16] (adapted to squares) on a high-level: They construct a quadtree $H(\mathcal{S})$ in which quadtree cells may store squares in $\mathcal{S}$. For any quadtree cell C, we denote by $\pi(C)$ the set of squares stored in C. A crucial definition is the concept of a *perimeter*. For a square σ, its perimeter $\mathbb{P}^*(\sigma, P)$ is intuitively a ring of $\Theta(P)$ quadtree cells of size at least $\frac{1}{4P}$ that are sufficiently close to the boundary of σ. Using this concept, their data structure is a pipeline of five components (Fig 1):

1. The set $\mathcal{S}$ gets stored in a compressed quadtree $H(\mathcal{S})$, such that for every $\sigma \in \mathcal{S}$, the quadtree contains $\mathbb{P}^*(\sigma, P)$.[1] This quadtree has $\Theta(P \cdot n)$ cells. For each $\sigma \in \mathcal{S}$, its storing cell is the maximal quadtree cell contained in σ.
2. They store all (maximal) quadtree cells contained in some $\sigma \in \mathcal{S}$ in a special ancestor data structure. We leave out the details of this structure, as we show that it suffices to use the well-studied marked-ancestor data structure by Alstrup, and Husfeldt, Rauhe [2].
3. For each square $\sigma \in \mathcal{S}$ with storing cell C_σ, for each quadtree cell $C_2 \in \mathbb{P}^*(\sigma, P)$, they store a square intersection data structure (a range tree). This data structure stores the squares $R \subset \pi(C_\sigma)$ that have C_2 in their perimeter (i.e. $R = \{\gamma \in \pi(C_\sigma) \mid C_2 \in \mathbb{P}^*(\sigma, P)\}$).
4. For each square σ with storing cell C_σ, for each quadtree cell $C_2 \in \mathbb{P}^*(\sigma, P)$, they store a maximal bichromatic matching (MBM) in the graph $G[R \cup B]$ with $B = \pi(C_2)$ with $R = \{\gamma \in \pi(C_\sigma) \mid C_2 \in \mathbb{P}^*(\sigma, P)\}$.
5. They store a *proxy graph* over the quadtree in the dynamic connectivity data structure by Holm, Lichtenberg, and Thorup (HLT) [14]. This graph contains an edge between two cells C_1, C_2 if and only if their maximal bichromatic matching is not empty.

They subsequently support connectivity queries for a query (σ, ρ) as follows. Given σ, obtain a pointer to its storing cell C_σ. Using their ancestor data structure, they obtain the largest ancestor C_a that is contained in a square $\sigma^* \in \mathcal{S}$. Let C^* be its storing cell. Doing the same procedure for ρ gives a cell R^*. They show that (σ, ρ) are connected in $G[\mathcal{S}]$ if and only if (C^*, R^*) are connected in the proxy graph; which they test in $O(\log n / \log \log n)$ time.

[1] Whilst originally their approach is a forest of quadtrees, we note that since each root of the forest is disjoint, the whole solution can be stored as a quadtree.

Why the space usage is high, and update times are not adaptive. For every $\sigma \in \mathcal{S}$, this pipeline creates a set of $\Theta(P)$ quadtree cells with sizes in $[\frac{1}{4P}, 1]$ which we denote by $\mathbb{P}^*(\sigma, P)$. For each quadtree cell $C_2 \in \mathbb{P}^*(\sigma, P)$, σ gets stored in a square intersection data structure, even if the quadtree cell C_2 is empty and/or has no other squares nearby. This approach makes it require a lot of space. Moreover, this approach fails when the aspect ratio becomes adaptive: suppose that $\mathcal{S}$ has $n-1$ squares with diameter $\frac{1}{P}$ and one unit square. We replace the unit square with a square of diameter $\frac{1}{P^2}$. The aspect ratio remains bounded by P. However (after rescaling the plane by a factor $\frac{1}{P}$) the quadtree no longer contains for every $\sigma \in \mathcal{S}$ the set $\mathbb{P}^*(\sigma, P)$ as (after rescaling the plane) it only contains quadtree cells of size 1. Reconstructing these requires $\Omega(Pn)$ time.

Our adaption. We improve this pipeline in several ways based on a few key insights: First, we revisit the definition of *perimeter*; presenting a new definition $\mathcal{P}(\sigma)$ which intuitively contains only cells in $\mathbb{P}^*(\sigma, \psi)$ that store at least one square. Since we only store data structures on cells that store at least one square, we save space and allow ψ to become fully adaptive. This introduces a new challenge, as we need to work with significantly fewer precomputed inormation. Concretely, we do the following (Figure 2):

1. We define a new type of quadtree $T(\mathcal{S})$ that uses only $O(n \log \psi)$ quadtree cells.
2. We replace their custom ancestor data structure by the well-studied *M*arked *A*ncestor *T*ree (MAT), simplifying the data structure.
3. For squares metric we create a new data structure that has deterministic guarantees and that avoids storing many copies.
4. For each square $\sigma \in \mathcal{S}$ with storing cell C_σ, for each quadtree cell C_2 in our new perimeter $\mathcal{P}(\sigma)$, we store a Maximal Bichromatic Matching (MBM*) in a graph $G[R \cup B]$. Using our new definition of perimeter, we define $R \leftarrow \{\gamma \in \pi(C_\sigma) \mid C_2 \in \mathcal{P}(\sigma)\}$ and $B \leftarrow \pi(C_2)$. We present a new algorithm to maintain this Maximal Bichromatic Matching.
5. Finally, we use HLT [14] on a proxy graph with an edge for every non empty MBM*.

We go through our data structures one by one in the order indicated in Figure 2.

3 Storing disks in quadtrees

Recall that F is a (dynamic) square bounding box of $\mathcal{S}$. By construction, the side length of F is 2^ω for some integer ω. We define the square F to be a *quadtree cell* and we recursively define quadtree cells to be any square obtained by *splitting* a cell into four equally sized closed squares. A *quadtree* T on F is any hierarchical decomposition obtained by recursively splitting cells. This hierarchical decomposition has a natural representation as a tree: the root is the cell F and every cell has either 0 or 4 children depending on whether it was split. We denote by $\mathbb{F}$ the (infinite) set of cells that are obtained by recursively splitting all cells, starting from F. We say that a cell $C \in \mathbb{F}$ is at *level* ℓ whenever its side length is 2^ℓ. We treat any quadtree T as a set of cells, i.e., $T \subset \mathbb{F}$.

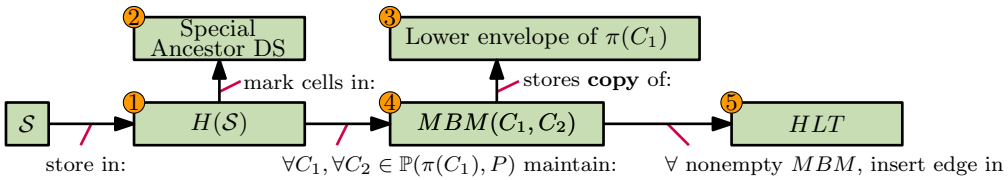

Figure 1 The five-component pipeline by Kaplan et al. where the arrows indicate dependencies.

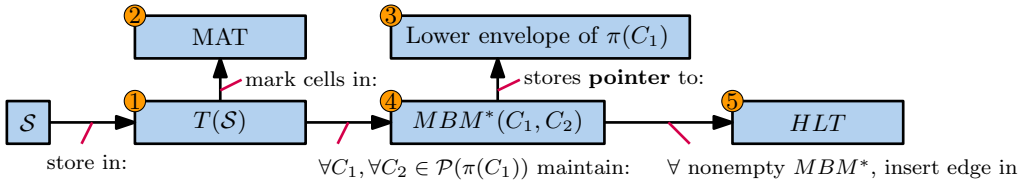

Figure 2 Our five-component pipeline where the arrows indicate dependencies.

Löffler, Simons, and Strash [23] use quadtrees to store arbitrary squares (or disks). Let F be fixed and σ be some square with center s. The unique *storing cell* $C_\sigma \in \mathbb{F}$ is a largest cell in $\mathbb{F}$ that contains the center s and that itself is contained in σ (if s lies at the intersection of multiple largest contained cells, we define the bottom left cell to be C_σ). Löffler, Simons and Strash subsequently say that C_σ *stores* σ. For a cell $C \in \mathbb{F}$, we denote by $\pi(C)$ all square of $\mathcal{S}$ stored in C. We will define five different cell sets to define our quadtree storing $\mathcal{S}$.

Quadtree cell sets. We assume that we have some bounding box F (which induces the set $\mathbb{F}$) and some set of storing cells in $\mathbb{F}$. We subsequently define two types of subsets of $\mathbb{F}$:

- definitions that originate from [16] and depend on some $P \in \mathbb{R}$ (blackboard font),
- and new definitions depending on only the storing cells (calligraphic font).

Quadtree cells C, C' are *neighboring* whenever they are not descendants of one another and intersect in their boundary. For any property, a quadtree cell C in $\mathbb{F}$ is *maximal* if there does not exist an ancestor of C in $\mathbb{F}$ with the same property.

Let $\sigma \in \mathcal{S}$ have a storing cell C_σ. We define (Figure 3):

- $\mathcal{N}(\sigma) \subset \mathbb{F}$ as the cells of size $|C_\sigma|$ neighboring C_σ or a neighbor of C_σ.
- $\mathbb{C}^*(\sigma, P) \subset \mathbb{F}$ as the maximal cells $C' \in \mathbb{F}$ with $C' \subset \sigma$ and $|C| \in [\frac{1}{4P}, 1]$.
- $\mathcal{C}(\sigma) \subset \mathbb{F}$ as the maximal cells $C' \in \mathbb{F}$ with $C' \subset \sigma$ that contain at least one storing cell.
- $\mathbb{P}^*(\sigma, P) \subset \mathbb{F}$ as the *perimeter* of σ. These are all $C' \in \mathbb{F}$ contained in a cell in $\mathcal{N}(\sigma)$ with $|C'| \in [\frac{1}{4P}, 1]$ with the additional property that there exists a square $\rho \subset \mathbb{R}^2$ where C' would be the storing cell of ρ if $\rho \in S$, and, ρ intersects the boundary of σ.
- $\mathcal{P}(\sigma) \subset \mathbb{F}$ as all *storing cells* with diameter at most $|\sigma|$ that, when scaled around their center by a factor 5, intersect *the boundary* σ. Note that these may be contained in σ.

For $R \subseteq \mathcal{S}$, we define $\mathcal{N}(R)$ to be the union of all $\mathcal{N}(\sigma)$ with $\sigma \in R$. All other sets (e.g., $\mathcal{P}(R)$) are defined analogously. Let $\ell_{\min}$ (resp. $\ell_{\max}$) be the smallest (resp. largest) level that contains any cell in any of the five sets. Per definition, $\ell_{\max} - \ell_{\min} \in O(\log \psi)$.

▶ **Lemma 1** (Lemma 4.2 in [16]). *For any $\sigma \in \mathcal{S}$, if regions are disks under an L_p metric with a diameter in $[\frac{1}{4P}, 1]$ then: $|\mathbb{C}^*(\sigma, P)| \in O(P)$ and $|\mathbb{P}^*(\sigma, P)| \in O(P)$.*

Compressed quadtrees. Denote by $X \subset \mathbb{F}$ some set of cells. Denote by T_X the minimal quadtree over some bounding box F that contains all cells in X. The size of T_X can be arbitrarily large, even when $|X|$ is constant. To reduce quadtree space complexity, a quadtree may be *compressed* [12]. An α-compressed quadtree (for some variable $\alpha \geq 1$) is defined as follows: let C be a quadtree cell in T_X and C_α be the smallest descendant of C such that (1) $|C| \geq 2^\alpha |C_\alpha|$ and (2) all cells in X that are contained in C are also contained in C_α. Then C has not 4 children, but only C_α as its child. Given some constant α, every quadtree has a unique maximally compressed equivalent that has size linear in $|X|$ [12]. Given the above definitions, we want to mention two different quadtrees that store $\mathcal{S}$:

- [23] defines $L(\mathcal{S})$ as the compressed quadtree storing $\mathcal{N}(\mathcal{S})$.
- [16] defines $H(\mathcal{S})$ as the compressed quadtree storing $\mathcal{N}(\mathcal{S})$, $\mathbb{C}^*(\mathcal{S})$ and $\mathbb{P}^*(\mathcal{S})$.

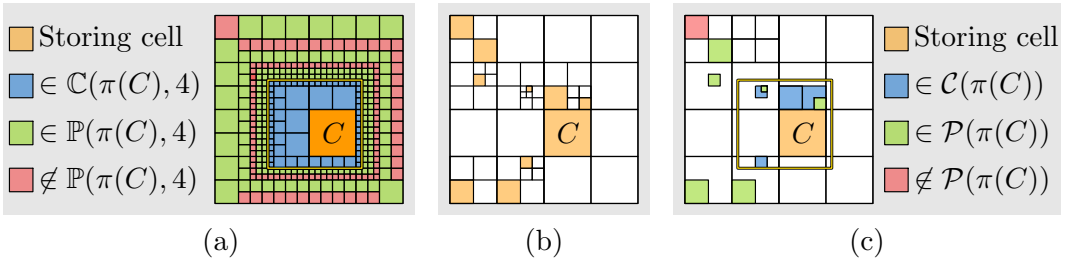

Figure 3 (a) We show for a square σ its storing cell in orange. We set $P = \psi = 2$ and show our sets. Many cells in $\mathbb{C}^*(\pi(C), 8)$ are also in $\mathbb{P}^*(\pi(C), 8)$. (b) The minimal quadtree that contains a set of storing cells. (c) Given the quadtree with storing cells, we illustrate our sets. Red cells are storing cells that occur in neither $\mathcal{C}(\pi(C))$ nor $\mathcal{P}(\pi(C))$.

Our quadtree. We define our quadtree $T(\mathcal{S})$ as the compressed quadtree storing $\mathcal{N}(\mathcal{S})$, where cells in $\mathcal{C}(\mathcal{S})$ are uncompressed. Note that any quadtree that contains $\mathcal{N}(\mathcal{S})$, also contains the cells in $\mathcal{C}(\mathcal{S})$. The key difference between $L(\mathcal{S})$ and $T(\mathcal{S})$ is that we decompress the cells in $\mathcal{C}(\mathcal{S})$, adding them to memory (i.e., we treat these cells as storing cells in the quadtree). Since these cells are uncompressed, this structure uses more space than the $O(n)$ cells in $\mathcal{N}(\mathcal{S})$. If we view quadtrees as a collection of (uncompressed) cells, then $L(\mathcal{S}) \subset T(\mathcal{S}) \subset H(\mathcal{S})$. By Lemma 1, Kaplan et al. [16] prove that $|\mathbb{C}^*(\mathcal{S}, P)|, |\mathbb{P}^*(\mathcal{S}, P)| \in O(Pn)$. It would be easy to show that $|\mathcal{C}(\mathcal{S})|, |\mathcal{P}(\mathcal{S})| \in O(n\psi)$. But through clever counting, we prove that storing $T(\mathcal{S})$ uses only $O(n \log \psi)$ space instead (Theorem 4).

3.1 Space complexity of the quadtree

We upper bound the size of $\mathcal{N}(\mathcal{S})$ and $\mathcal{C}(\mathcal{S})$ (and thus the size of $T(\mathcal{S})$).

▶ **Observation 2.** *There are $O(n)$ cells in $\mathcal{N}(\mathcal{S})$.*

▶ **Lemma 3.** *Let C be a storing cell. Denote by Z any cell, such that there could exist a σ stored in Z where an ancestor of C lies in $\mathcal{C}(\sigma)$. There are at most $O(\log \psi)$ such cells and we can report them in $O(\log \psi \log n)$ time.*

Proof. Let $|C| = 2^\ell$ (i.e., C is at level ℓ). By definition, Z is in a level $j \geq \ell$. Fix a level j. For any cell Z at level j, $\mathcal{C}(\pi(Z))$ contains an ancestor of C only if a square in $\pi(Z)$ intersects (or contains) C_j (the ancestor of C at level j). As the diameter of squares in $\pi(Z)$ is at most a factor 5 larger than the diameter of Z, there are at most $O(1)$ cells at level j that *could* store a square ρ that intersects C_j (the neighbors of C_j, their neighbors and possibly their neighbors). We can find these cells in $O(\log n)$ time by doing a point location in each cell for the level j. The fact that per definition of ψ, all storing cells and all cells in $\mathcal{C}(\mathcal{S})$ lie in a range of $O(\log \psi)$ levels concludes the proof. ◀

▶ **Theorem 4.** *At all times, the compressed quadtree $T(\mathcal{S})$ uses $O(n \log \psi)$ space.*

Proof. Since $T(\mathcal{S})$ is the quadtree that stores $\mathcal{N}(S)$ and $\mathcal{C}(S)$, and compressed quadtrees have linear space in the number of uncompressed cells, Observation 2 and Lemmas 3 immediately imply the theorem. ◀

We additionally upper bound the size of two more quadtree cell types:

▶ **Lemma 5.** *Let $\mathcal{S}$ be a set of squares with aspect ratio ψ. For all $\sigma \in \mathcal{S}$ with storing cell C_σ there are at most $O(\psi)$ cells in $\mathcal{C}(\sigma)$ and $\mathcal{P}(\sigma)$ and $O(\psi^2)$ cells in $\mathcal{P}(\pi(C_\sigma))$.*

Proof. Consider any set of squares S. We rescale the plane such that the diameter of squares in $\mathcal{S}$ lies in $[\frac{1}{\psi}, 1]$. We apply Lemma 1 to conclude that $|\mathbb{C}(\sigma, \psi)|, |\mathbb{P}(\sigma, \psi)| \in O(\psi)$. Note that the smallest storing cell in $T(\mathcal{S})$ then has size $\frac{1}{4\psi}$. Having rescaled, $\mathcal{C}(\sigma) \subseteq \mathbb{C}^*(\sigma, \phi)$.

Suppose that after rescaling there is a cell $C \in \mathcal{P}(\sigma)$ that is not in $\mathbb{P}^*(\sigma, \psi)$. Then C, scaled by a factor 5, intersects the boundary of σ. Yet if $C \notin \mathbb{P}^*(\sigma, \psi)$ then C cannot store any square ρ that intersects σ. Denote by C' a neighbor of C of size $2|C|$ that lies closer to the center of σ. It must be that $C' \in \mathbb{P}^*(\sigma, \psi)$ (indeed, we can construct a square with diameter $4|C|$ stored in C' that intersects σ). This way, each cell in $\mathbb{P}^*(\sigma, \psi)$ can get charged by at most $O(1)$ cells $C' \in \mathcal{P}(\sigma)$ where $C' \notin \mathbb{P}^*(\sigma, \psi)$. Thus, $|\mathcal{P}(\sigma)| \in O(\psi)$. The upper bound on $|\mathcal{P}(\pi(C_\sigma))|$ follows from the standard packing argument. ◀

▶ **Lemma 6.** *Let C be a storing cell. Denote by Z any cell, such that there could exist a σ stored in Z with $C \in \mathcal{P}(\sigma)$. We can report all $O(\log \psi)$ such cells in $O(\log \psi \log n)$ time.*

Proof. Let $|C| = 2^\ell$ (i.e., C is at level ℓ in the quadtree). By definition, every quadtree cell Z of the lemma statement is stored at a level $j \geq \ell$. Fix a level $j \geq \ell$ and let C_j be the ancestor of C at level j. If for any cell Z at level j, $C \in \mathcal{P}(\pi(Z))$ then it must be that the cells Z and C_j (when both are scaled around their center by a factor 5) intersect. There are at most $O(1)$ such cells Z at level j for which this can be true. We can find these cells at level j in $O(\log n)$ time by performing $O(1)$ point locations in the quadtree (querying a neighborhood of 25×25 around C_j). The fact that all cells in $\mathcal{P}(\mathcal{S})$ lie in a range of $O(\log \psi)$ levels concludes the proof. ◀

4 Maintaining and navigating quadtrees

A compressed quadtree T_X that stores a set X of quadtree cells can be dynamically maintained in $O(\log |X|)$ time per insertion and deletion [12]. Moreover, leaf location queries are supported in $O(\log |X|)$ time, which take as input some point $q \in \mathbb{R}^2$ and output the leaf of T_X that contains q. By Theorem 4, $|X| = O(n \log \psi)$ in our setting. Since we assume that $\psi \in O(n^c)$, see Section 2, we can say that our insertion, deletion and point location operations in the compressed quadtree take $O(\log n)$ time. Compressed quadtrees additionally support level locations where for any query point $q \in \mathbb{R}^2$ and level ℓ, the output is the quadtree cell at level ℓ that contains q; this can be used to dynamically maintain for all $\sigma \in \mathcal{S}$ the set $\mathcal{N}(\sigma)$ in our quadtree in $O(\log n)$ time per update in $\mathcal{S}$ [5].

We maintain $L(\mathcal{S})$ in $O(\log n)$ time per update to $\mathcal{S}$, while supporting point location queries. We maintain in $O(\log n)$ time per update in $\mathcal{S}$ the values $d_{\max}$ and $d_{\min}$ that denote the maximal and minimal diameter in $\mathcal{S}$ respectively. We apply 3 more data structures:

Marked Ancestor Trees (MAT). Alstrup, Husfeldt, and Rauhe [2] introduce marked-ancestor trees. Let T be a dynamic tree. Each node in T is either marked or unmarked. Given a node $v \in T$, the MAT supports changing the mark of v or updating T in $O(\log \log n)$ time. Additionally, given a node $v \in T$, one can find the lowest/highest marked node on the path from v to the root in $O(\log n / \log \log n)$ time. We augment our quadtree with a MAT.

Orthogonal range trees. Willard and Lueker [30] show a data structure to store a set of n squares using $O(n \log n)$ space. Given a query rectangle ρ, it can report an input square contained in ρ in $O(\log^4 n)$ time, if such a square exists. Given a query square ρ it can report the number of squares that contain ρ in $O(\log^4 n)$ time. We implement the range tree using general balanced trees [4], so that we can support updates in amortized $O(\log^4 n)$ time.

Segment trees. Segment trees [8] store a set of n horizontal segments using $O(n \log n)$ space, so that for a vertical query segment Q we obtain all k input segments that intersect Q in $O(\log^2 n + k)$ time. The data structure can again be made dynamic, supporting updates in $O(\log^2 n)$ amortized time. For any σ, we store the horizontal sides of C_σ (scaled by a factor 5) in such an orthogonal intersection data structure. Furthermore, we create a second such a data structure storing all vertical sides.

▶ **Theorem 7.** *Let $\mathcal{S}$ be a set of axis-aligned squares. We augment $T(\mathcal{S})$ with an $O(n \log n)$-size data structure using $O(n \log n)$ space, supporting inserting/deleting a square σ in $O(|\mathcal{C}(\sigma)| \cdot \log^4 n + \log^6 n)$ time, and*

- *all cells in $\mathcal{C}(\mathcal{S})$ are marked in our marked-ancestor tree;*
- *for any query square γ, we can obtain $\mathcal{P}(\gamma)$ in $O(\log^2 n + |\mathcal{P}(\gamma)|)$ time.*
- *for any query cell C, we obtain the set $\mathcal{Z}(C) := \{Z \mid C \in \mathcal{P}(\pi(Z))\}$ in $O(\log^5 n)$ time.*

Proof. By Theorem 4, our quadtree requires $O(n \log \psi)$ space. Using the standard operations on compressed quadtrees, we can maintain $\mathcal{N}(\mathcal{S})$ in $O(\log n)$ time per update. What remains for quadtree maintenance is to identify, decompress and mark all cells in $\mathcal{C}(\mathcal{S})$.

Maintaining $\mathcal{C}(\mathcal{S})$. Every cell $C \in \mathbb{F}$ has a counter that counts for how many $\sigma \in \mathcal{S}$, $C \in \mathcal{C}(\sigma)$. Whenever the counter is zero, we do not store it explicitly. Otherwise, $C \in \mathcal{C}(\mathcal{S})$ and we need to make sure that C is decompressed and marked. For each update of a square σ with storing cell C_σ, there are two types of counter updates:

1. updating counters of $C \in \mathcal{C}(\sigma)$, and
2. updating the counters of C_σ and its ancestors.

We start with the first case. Instead of increasing counters, we do something slightly stronger as we can report all of $\mathcal{C}(\sigma)$. By definition, $C_\sigma \in \mathcal{C}(\sigma)$ and we add it to our output. We split σ into eight rectangles that are bounded by σ and the boundary of C_σ, see Figure 4. We process each rectangle separately. Consider such a rectangle R. We query our orthogonal range tree to report a storing cell C_1 in R in $O(\log^4 n)$ time. Given C_1, we walk in $O(\log \psi)$ time up the quadtree to find its largest ancestor C_1' that is still contained in σ. By definition, $C_1' \in \mathcal{C}(\sigma)$ and we add it to our output. Subsequently, we partition R into nine rectangles that are bounded by the boundaries of C_1' and recurse. For each cell in $\mathcal{C}(\sigma)$ we perform eight orthogonal range queries. For each range query, we either conclude that the range contains no cells in $\mathcal{C}(\sigma)$, or we identify at least one cell in $\mathcal{C}(\sigma)$. As we recurse on rectangles that are bounded by cell boundaries and we explore all of R, we find all cells of $\mathcal{C}(\sigma)$ in $O(|\mathcal{C}(\sigma)| \cdot \log^2 n)$ time. As we find them, we may adjust their counters.

Now onto the second case, where we simply recompute all counters from scratch. There are at most $O(\log \psi)$ ancestors $C_\sigma, C_1, \ldots C_k$ of C_σ that may be contained in a square in $\mathcal{S}$. For each of these, we recompute their counters from scratch. Fix an ancestor C_i with parent C_{i+1}. By Lemma 3, there are at most $O(\log \psi)$ cells Z such that Z could store a square γ with $C_i \in \mathcal{C}(\gamma)$. We obtain all such Z in $O(\log \psi \log n)$ time and iterate over each of them. For a fixed Z, we use the range tree to count how many $\gamma \in \pi(Z)$ contain C_i in $O(\log^4 n)$ time. We then count how many $\gamma \in \pi(Z)$ contain C_{i+1}. The difference between these counts is the number of squares $\gamma^* \in \pi(Z)$ for which $C_i \in \mathcal{C}(\gamma^*)$. We compute and sum all these numbers to recompute the count of C_i. It follows from the fact that $O(\log \psi) \subset O(\log n)$ that we can maintain $\mathcal{C}(\mathcal{S})$ in $O(|\mathcal{C}(\sigma)| \cdot \log^2 n + \log^6 n)$ time.

Querying for $\mathcal{P}(\gamma)$. We show how to obtain for any query squares γ the set $\mathcal{P}(\gamma)$ in $O(\log^2 n + |\mathcal{P}(\gamma)|)$ time. For each storing cell C, we consider the cell C^* that is C scaled by a factor 5 around its center. We store each of the boundary segments of C^* in our Segment Tree. There is exactly one storing cell per square in $\mathcal{S}$, so maintaining the Segment Tree intersection data structure takes $O(\log^2 n)$ time per update and uses $O(n \log n)$ space. For any query square γ, we have $C \in \mathcal{P}(\gamma)$ if and only if one of the boundary segments of γ intersects one of the boundary segments of C^*. Thus, we can immediately use the intersection data structure to compute $\mathcal{P}(\gamma)$ in $O(\log^2 n + |\mathcal{P}(\gamma)|)$ time, as each cell in $\mathcal{P}(\gamma)$ is reported at most a constant number of times.

Querying for $\mathcal{Z}(C)$. Denote by Z any cell, such that there could exist a γ stored in Z with $C \in \mathcal{P}(\gamma)$. By Lemma 6, we can report all at $O(\log \psi)$ such cells in $O(\log \psi \log n)$ time. For every such Z, we conceptually rotate the plane such that Z lies above C. Let C^* be the cell C increased by a factor 5 around its center. Any square ρ in $\pi(Z)$ intersects C^* in its boundary if and only if one of two conditions hold: ρ contains the top left endpoint of C^* but not the bottom left endpoint, or ρ contains the top right endpoint of C^* but not the bottom right endpoint. We select the top right endpoint of C^* and we count how many squares in $\pi(Z)$ contain the top right endpoint in $O(\log^4 n)$ time. We do the same for the bottom right endpoint. If the counts differ, there is at least one square in $\pi(Z)$ that intersects C^* in its boundary and thus $C \in \mathcal{P}(\pi(Z))$. Doing this for all $O(\log \psi)$ levels takes $O(\log^5 n)$ time. ◀

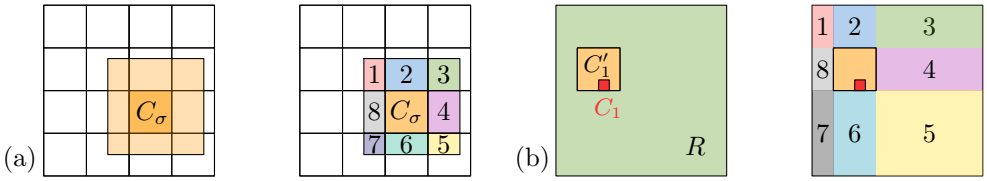

Figure 4 (a) Given a storing cell C_σ, we partition σ into nine rectangles (one being C_σ). (b) For each rectangle R, we do a range query to find a storing cell C_1 (if it exists). For the largest ancestor $C_1' \subset \sigma$ of C_1, we partition R into nine rectangles once again and recurse.

5 Specific square intersection data structures

In this section we develop a solution for the following data structure problem: Let $\mathcal{R}$ be a set of m squares. Let $R_1, \ldots, R_k$, be k subsets of $\mathcal{R}$ that we refer to as *conflict sets* and let $\ell \leq k$ be the maximum number of conflict sets that any square from $\mathcal{R}$ appears in. We want to store $\mathcal{R}$ and all the conflict sets $R_1, \ldots, R_k$ in a data structure that has size near linear in m and $z = \sum_i |R_i|$, and support the following operations in the following time:

Insert(ρ), ($O(l \cdot \log^3 m)$ time): Insert a square ρ into $\mathcal{R}$.
Delete(ρ), ($O(l \cdot \log^3 m)$ time): Delete a square ρ from $\mathcal{R}$ and every R_i that it occurs in.
Insert(ρ, R_i), ($O(\log^3 m)$ time): Insert a square ρ in the conflict set R_i. If $R_i = \emptyset$, create a new conflict set.
Delete(ρ, R_i), ($O(\log^3 m)$ time): Delete a square ρ from the conflict set R_i.
Query(σ, R_i, C), ($O(\log^3 m)$ time): Given a query σ whose center lies below all centers of all squares in $\mathcal{R}$, and a horizontal line segment C below σ, return (if it exists) a square $\rho \in \mathcal{R}$ that intersects σ, but is not in the conflict set R_i, and that does not contain C.

In Section 6 we solve this problem as follows: we map every square $\rho = [\ell_\rho, r_\rho] \times [b_\rho, t_\rho] \in \mathcal{R}$ to a point $p_\rho = (b_\rho, \ell_\rho, r_\rho)$ in $\mathbb{R}^3$, and store these points in a 3D-range tree T augmented for range counting queries [8]. Hence, every third-level subtree T_ν stores the number of points m_ν in T_ν. Furthermore, for each such subtree, and each conflict set R_i, consider the subset of points stored in the leaves of T_ν for which the corresponding square appears in R_i. If this set is non-empty then node ν also stores the size $m_{\nu,i} = |\{p_\rho \mid p_\rho \in T_\nu \wedge \rho \in R_i\}|$ of this set. Furthermore, we maintain a bipartite graph between the squares in $\mathcal{R}$ and the conflict sets R_i, so that given a square $\rho \in \mathcal{R}$ we can find the ℓ conflict sets it appears in in $O(\ell)$ time. We implement all trees using general balanced trees [4] so that we can perform updates efficiently. A 3D-range tree uses $O(m \log^2 m)$ space. Each square in the multiset $\bigcup_i R_i$ contributes to $O(\log^3 m)$ nodes of T, and hence the entire structure uses at most $O((m+z)\log^3 m)$ space (Lemma 8). In Section 7 we use this structure for connectivity queries.

6 Square intersection data structure

Let $\mathcal{R}$ be a set of m squares. Let $R_1, \ldots, R_k$, be k subsets of $\mathcal{R}$ that we refer to as *conflict sets* and let $\ell \leq k$ be the maximum number of conflict sets that any square from $\mathcal{R}$ appears in. We want to store $\mathcal{R}$ and all the conflict sets $R_1, \ldots, R_k$ in a data structure that has size near linear in m and $z = \sum_i |R_i|$, and support the following operations:

Insert(ρ): Insert a square ρ into $\mathcal{R}$.
Delete(ρ): Delete a square ρ from $\mathcal{R}$ and every R_i that it occurs in.
Insert(ρ, R_i): Insert a square ρ in the conflict set R_i. If $R_i = \emptyset$, create a new conflict set.
Delete(ρ, R_i): Delete a square ρ from the conflict set R_i. If R_i becomes empty, delete R_i.
Query(σ, R_i, C): Given a query square σ whose center lies below all centers of all squares in $\mathcal{R}$, and a horizontal line segment C below σ, return (if it exists) a square $\rho \in \mathcal{R}$ that intersects σ, but is not in the conflict set R_i, and that also does not contain C.

We map every square $\rho = [\ell_\rho, r_\rho] \times [b_\rho, t_\rho] \in \mathcal{R}$ to a point $p_\rho = (b_\rho, \ell_\rho, r_\rho)$ in $\mathbb{R}^3$, and store these points in a 3D-range tree T augmented for range counting queries [8]. Hence, every third-level subtree T_ν stores the number of points m_ν in T_ν. Furthermore, for each such subtree, and each conflict set R_i, consider the subset of points stored in the leaves of T_ν for which the corresponding square appears in R_i. If this set is non-empty then node ν also stores the size $m_{\nu,i} = |\{p_\rho \mid p_\rho \in T_\nu \wedge \rho \in R_i\}|$ of this set. Furthermore, we maintain a bipartite graph between the squares in $\mathcal{R}$ and the conflict sets R_i, so that given a square $\rho \in \mathcal{R}$ we can find the ℓ conflict sets it appears in in $O(\ell)$ time. We implement all trees using general balanced trees [4] so that we can perform updates efficiently. A 3D-range tree uses $O(m \log^2 m)$ space. Each square in the multiset $\bigcup_i R_i$ contributes to $O(\log^3 m)$ nodes of T, and hence the entire structure uses at most $O((m+z)\log^3 m)$ space. We show how to answer our queries and how to update the data structure:

▶ **Lemma 8.** *Let $\mathcal{R}$ be a set of m squares, let $z = \sum |R_i|$, and let there be at most $k \leq m$ conflict sets. Let each $\rho \in \mathcal{R}$ appear in at most l conflict sets. There is a data structure $\mathcal{D}^*(\mathcal{R})$ of size $O((m+z)\log^3 m)$ that supports:*
Insert(ρ) *in $O(l \cdot \log^3 m)$ amortized deterministic time,*
Delete(ρ) *in $O(l \cdot \log^3 m)$ amortized deterministic time,*
Insert(ρ, R_i) *in $O(\log^3 m)$ amortized deterministic time,*
Delete(ρ, R_i) *in $O(\log^3 m)$ amortized deterministic time, and*
Query(σ, R_i, C) *in $O(\log^3 m)$ amortized deterministic time.*

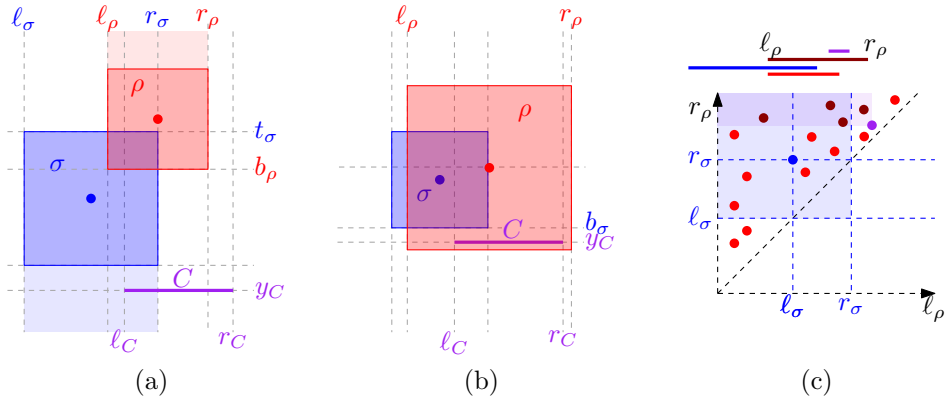

Figure 5 (a) Since the center of ρ lies above the center of σ, we can essentially treat ρ as a rectangle unbounded from the top, and σ as a rectangle unbounded from the bottom. (b) A square ρ may intersect σ but is not allowed to contain C. (c) The x-extents of the objects map to points in $\mathbb{R}^2$. Squares (whose x-extent) intersects (the x-extent) of σ lie in the blue region, and are not allowed to lie in the purple region.

Proof. To insert a square $\rho \in \mathcal{R}$ we use the standard insertion procedure for (dynamic) 3D-range trees; we insert the point p_ρ into $O(\log^3 m)$ subtrees. If, one of our subtrees becomes too unbalanced, we rebuild it from scratch. Rebuilding a dD-range tree on a set P of n points can be done in $O(n \log^{d-1} n)$ time. However, we also still have to update the $m_{\nu,i}$ counts for each ternary subtree T_ν and each conflict list. We can do this in $O(ln \log^d n)$ time as follows. For each point $p_\rho \in P$ we obtain the at most l conflict sets R_i it appears in, and for each leaf corresponding to p_ρ we simply walk upward updating the $m_{\nu,i}$ counts appropriately. It follows that the amortized insertion time is $O(l \log^3 m)$. Deletions are handled similarly in $O(l \log^3 m)$ amortized time.

To insert or delete a square ρ in one of the conflict sets R_i we update the $m_{\nu,i}$ counts in the $O(\log^3 m)$ affected nodes (and we insert or delete the appropriate edge in the bipartite graph). If one of the $m_{\nu,i}$ counts reaches zero after a deletion, we stop storing it. Any counts that we do not store explicitly are considered to be zero.

Consider a query with square $\sigma = [\ell_\sigma, r_\sigma] \times [b_\sigma, t_\sigma]$, horizontal segment $C = [\ell_C, r_C] \times \{y_C\}$, and conflict set R_i (see also Figure 5). We will argue that there are $O(1)$ axis parallel boxes $Q_1, .., Q_{O(1)}$ such that the subset of squares from $\mathcal{R}$ that intersect σ but do not contain C is the subset of points that lies in $\bigcup_j Q_j$. Our range tree allows us to obtain $O(\log^3 m)$ ternary subtrees T_ν that together represent the points in this region (in $O(\log^3 m)$ time). For each such subtree we then consider the counts m_ν and $m_{\nu,i}$: if they are equal all points (squares) in T_ν also appear in R_i, and hence there are no candidate points (squares) to be found in T_ν. Otherwise, we have $m_\nu > m_{\nu,i}$, and hence T_ν does contain a point p_ρ for which ρ intersects σ, does not contain C, and for which $\rho \notin R_i$. Moreover, one of the two children of ν, say node μ, must then also have $m_\mu > m_{\mu,i}$. This way we can find p_ρ in time proportional to the height of T_ν. It follows that the total query time is $O(\log^3 m)$. All that remains is to describe the regions $Q_1, .., Q_{O(1)}$.

Since the center of σ is guaranteed to lie below all centers of squares in $\mathcal{R}$, and $y_C \leq b_\sigma$, we can essentially treat all squares as three-sided rectangles. In particular, a square $\rho \in \mathcal{R}$ intersects σ if and only if $p_\rho = (b_\rho, \ell_\rho, r_\rho)$ lies in the query range $Q = (-\infty, t_\sigma] \times (-\infty, r_\sigma] \times [\ell_\sigma, \infty)$ (see Figure 5). Using that $y_c \leq b_\sigma \leq (b_\rho + t_\rho)/2$, we find that $C \subset \rho$ if and only if p_ρ lies in the range $Q' = (\infty, y_C] \times (-\infty, \ell_C] \times [r_C, \infty)$. Hence, ρ intersects σ, but does not contain C if and only if $p_\rho \in Q \setminus Q'$. Since both Q and Q' are orthogonal boxes this region can be expressed as the union of $O(1)$ orthogonal ranges. ◀

7 Maximal Bichromatic Matchings

The Maximal Bichromatic Matching data structure (MBM) in [16] relies upon a square intersection data structure $\mathcal{D}$. Consider a pair of disjoint quadtree cells C_1, C_2 and two sets $R \subseteq \pi(C_1)$ and $B \subseteq \pi(C_2)$. The MBM stores two square intersection data structures: $\mathcal{D}(R)$ and $\mathcal{D}(B)$, plus a maximal bichromatic matching M_{RB} of the graph $G[R \cup B]$.

Given two such cells C_1, C_2, they dynamically maintain the matching as follows: For all edges in M_{RB}, dynamically remove the endpoints from the square intersection structures storing $\mathcal{D}(R \backslash M_{RB})$ and $\mathcal{D}(B \backslash M_{RB})$. When a new square σ gets inserted into B, query $\mathcal{D}(R \backslash M_{RB})$ to find a square in $R \backslash M_{RB}$ that intersects σ. If such a square ρ exists, add the edge (ρ, σ) to the matching M_{RB}. Subsequently delete ρ from $\mathcal{D}(R \backslash M_{RB})$. With a similar procedure for deletions, one can dynamically maintain M_{RB} in time proportional to the update and query time of the intersection data structure $\mathcal{D}$.

Defining the sets R and B. The sets R and B must be carefully chosen if we want to avoid spending quadratic space in n or ψ. Recall the pipeline of Kaplan et al. [16]. For each storing cell C_1, for each $C_2 \in \mathbb{P}^*(\pi(C_1))$, they store an MBM between the pair (C_1, C_2). We know that there may be $\Theta(\psi^2)$ cells in the set $\mathbb{P}^*(\pi(C_1))$. Suppose that for each pair C_1, C_2 they set $R \leftarrow \pi(C_1)$ and $B \leftarrow \pi(C_2)$. Every square in $\pi(C_1)$ may get stored $O(\psi^2)$ times and the total space usage is $O(\psi^2 n)$. To improve space and time usage, the authors of [16] instead set $R \leftarrow \{\sigma \in \pi(C_1) \mid C_2 \in \mathbb{P}^*(\sigma)\}$ and $B \leftarrow \pi(C_2)$, see Figure 6(a). Since each square $\sigma \in \pi(C_1)$ has $O(\psi)$ cells in its perimeter $\mathbb{P}^*(\sigma)$, each square σ is stored $O(\psi)$ times and the total space is $O(\psi \cdot |\pi(C_1)|)$. A charging argument then shows that the total space required is $O(\psi n)$. The data structures can be updated in $O(\psi)$ times: the update time of the intersection data structures $\mathcal{D}(R \backslash M_{RB})$ and $\mathcal{D}(B \backslash M_{RB})$.

Defining an MBM for adaptive ψ. When the aspect ratio is adaptive (or, even when it is bounded), the approach by Kaplan et al. [16] requires an update time linear in ψn, since, after increasing ψ, the perimeter $\mathbb{P}^*(\pi(\sigma))$ increases in size by $O(\psi)$ for every storing cell C_1. We could try to avoid this issue by replacing their definition of perimeter with ours. That is, for every cell C_1, we would consider the $O(\psi^2)$ cells C_2 in $\mathcal{P}(\pi(C_1))$. For the pair C_1, C_2, we want to maintain an MBM between sets $R \leftarrow \{\sigma \in \pi(C_1) \mid C_2 \in \mathcal{P}(\sigma)\}$ and $B \leftarrow \pi(C_2)$; by storing R and B each in their separate data structure for square intersection queries. However, such a structure can also not be efficiently dynamically maintained, see Figure 6(b). Thus, we must avoid storing the sets $R \backslash M_{RB}$ and $B \backslash M_{RB}$ explicitly in a data structure.

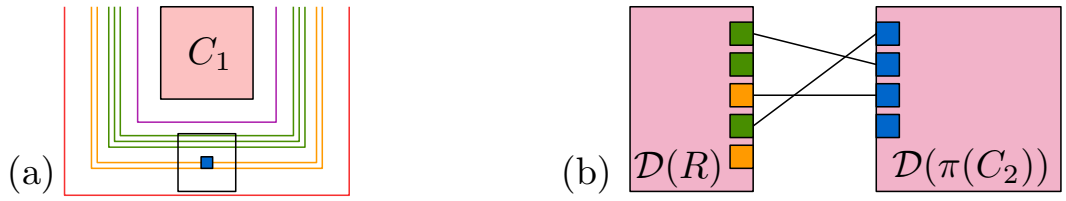

Figure 6 (a) C_1 with a blue $C_2 \in \mathbb{P}^*(\pi(C_1))$. The elements of the set $R = \{\sigma \in \pi(C_1) \mid C_2 \in \mathbb{P}^*(\sigma)\}$ are the green and yellow squares. (b) If we split C_2 to create a cell C', then the corresponding R' would consist only of the orange squares. Since there exists no efficient way to split intersection data structures, constructing the new data structure on R' takes linear time.

Applying our data structure problem. We now apply our previous data structure problem. Let C_1 be a storing cell in our quadtree $T(\mathcal{S})$. We maintain the data structure $\mathcal{D}^*(\pi(C_1))$ (i.e. we set $\mathcal{R} \leftarrow \pi(C_1)$). Let there be k cells in the perimeter $\mathcal{P}(\pi(C_1))$. Then by Lemma 5, we have $k \in O(\psi^2)$. Let C_i be some storing cell in $\mathcal{P}(\pi(C_1))$, we denote by M_i some maximal matching in the graph $G[R \cup B]$ for $R \leftarrow \{\gamma \in \pi(C_1) \mid C_i \in \mathcal{P}(\gamma)\}$ and $B \leftarrow \pi(C_i)$. Denote by R_i the squares in $\pi(C_1)$ that are part of M_i; we say that R_i is a conflict set. The result of this transformation are k conflict sets of $\pi(C_1)$. Moreover, each square ρ in $\pi(C_1)$ may appear in at most $l \in O(|\mathcal{P}(\rho)|) \subset O(\psi)$ conflict sets. We now apply Lemma 8 four times (one for each direction). We maintain for all C_1 this data structure $\mathcal{D}^*(\pi(C_1))$ and show:

▶ **Theorem 9.** *Let $\mathcal{S}$ be a set of n squares with adaptive aspect ratio ψ. We can maintain for pair of storing cells (C_1, C_2) a maximal bichromatic matching in $G[R \cup \pi(C_2)]$ with $R \leftarrow \{\gamma \in \pi(C_1) \mid C_2 \in \mathcal{P}(\gamma)\}$. Our solution uses $O(n \log \psi \log^3 n)$ space. Inserting/deleting a square σ requires $O(|\mathcal{P}(\sigma)| \cdot \log^3 n + \log^5 n)$ amortized time.*

Proof. We first analyse our space usage and then show how to maintain each matching.

Upper bounding size. For C_1, denote by z_1 the sum of all C_i, over all edges in the maximal matching of $G[R \cup \pi(C_i)]$ with $R \leftarrow \{\gamma \in \pi(C_1) \mid C_2 \in \mathbb{P}^*(\gamma)\}$. Lemma 8 presents a data structure with size $O((|\pi(C_1)| + z_1) \log^3 |\pi(C_1)|)$. Denote by $\mathcal{M}(\mathcal{S})$ the set of all edges, across all maximal bichromatic matchings, for all pairs of storing cells (C_1, C_2). Let $\mathcal{M}(\mathcal{S})$ contain z^* elements. It follows that all these data structures use at most $O((n + z^*) \cdot \log^3 n)$ total space. We upper bound the number of edges in z^* by charging each edge to one of their endpoints. Intuitively, we charge each matched edge to the squares of the smallest quadtree cell. Every square $\sigma \in \mathcal{S}$ receives at most $O(\log \psi)$ charges and z^* is upper bound by $n \log \psi$.

More formally, we over-estimate the edges in $\mathcal{M}(\mathcal{S})$. Fix for *every* pair (C_1, C_2) with $C_2 \in \mathcal{P}(\pi(C_1)))$ an arbitrary maximal bichromatic matching in the graph $G[\pi(C_1) \cup \pi(C_2)]$ (i.e., we ignore the fact that we match between sets $R \subseteq \pi(C_1)$ and $B \subseteq \pi(C_2)$, and fix some potentially larger matching in the bigger graph $G[\pi(C_1) \cup \pi(C_2)]$). Denote for C_1 by $\mathcal{M}_{\prec}(C_1)$ the set of all matchings between C_1 and C_2 where $|C_1| \prec |C_2|$ for $\prec \in \{<, =, >\}$. Any edge $e \in \mathcal{M}(\mathcal{S})$ is in $\mathcal{M}_=(C_1) \cup \mathcal{M}_<(C_1)$ for some storing cell C_1. First, we upper bound $|\mathcal{M}_=(C_1)|$. There are $O(1)$ cells C_2 with $|C_1| = |C_2|$ and $C_1 \in \mathcal{P}(\pi(C_2))$ or vice versa. For every such C_2, there can be at most $O(|\pi(C_1)|)$ edges in a MBM in $G[\pi(C_1) \cup \pi(C_2)]$. Thus, there are at most $O(|\pi(C_1)|)$ edges in $\mathcal{M}_=(C_1)$. Second, by Lemma 6, there are at most $O(\log \psi)$ cells C_2 with $C_1 \in \mathcal{P}(\pi(C_2))$ and $|C_1| < |C_2|$. Again, every matching in $G[\pi(C_1) \cup \pi(C_2)]$ has at most $O(|\pi(C_1)|)$ edges, thus $\mathcal{M}_<(C_1)$ contains at most $O(|\pi(C_1)| \cdot \log \psi)$ edges. Now:

$$z^* = |\mathcal{M}(\mathcal{S})| \leq \sum_{\text{storing cell } C_1} |\mathbb{M}_=(C_1)| + |\mathbb{M}_<(C_1)| \leq \sum_{\text{storing cell } C_1} |\pi(C_1)| \cdot \log \psi \leq n \log \psi$$

It follows we use at most $O((n + z^*) \cdot \log^3 n) \subset O(n \log \psi \log^3 n))$ space.

Maintaining the MBM. Suppose that we delete a square σ from $\mathcal{S}$ (this is the more difficult case). We can find its storing cell C_σ in $O(\log n)$ time using standard quadtree navigation. We obtain $\mathcal{D}^*(\pi(C_\sigma))$ with its $k \in O(\psi^2)$ conflict sets. Recall that σ appears in at most $l \in O(|\mathcal{P}(\sigma)|)$ conflict sets. By Lemma 8, we may remove σ from the data structure $\mathcal{D}^*(\pi(C_\sigma))$ in $O(l \log^2 n) \subseteq O(|\mathcal{P}(\sigma)| \cdot \log^2 n)$ amortized time. What remains is to update all the matchings. We recall that we maintain a matching between (C_σ, C_2) in two cases: either the cell $C_2 \in \mathcal{P}(\pi(C_\sigma))$ or $C_\sigma \in \mathcal{P}(\pi(C_2))$. There are at most $O(|\mathcal{P}(\pi(C_\sigma))|)$ cells of the first case, and $O(\log \psi)$ cells of the second case. By Theorem 7, we may obtain all such cells in $O(\log^5 n + |\mathcal{P}(\sigma)|)$ time.

Processing a cell C_2. Fix a cell C_2 with a corresponding conflict set R_2 in $\mathcal{D}^*(\pi(C_\sigma))$. We test if σ was an endpoint of the matching (σ, ρ) by searching over the conflict set R_2. If so, then we delete σ from the conflict set R_2. What remains is to try and rematch ρ.

Thus, we want to find a square in $R = \{\gamma \in \pi(\sigma) \mid C_2 \in \mathcal{P}(\gamma)\}$, that is not already in the conflict set R_2 (i.e. not already in the matching between $G[R \cup B]$). Denote by K the cell C_σ scaled by a factor 5 around its center and by $\underline{K}$ the bottom facet of K. We claim that ρ can be matched to a square in R if and only if $\text{Query}(\rho, R_2, \underline{K})$ from Lemma 8 is not empty.

Indeed, for any $\gamma \in \pi(C_\sigma)$ that intersects ρ and contains $\underline{K}$, must contain K. By definition, $C_2 \notin \mathcal{P}(\gamma)$ and thus $\gamma \notin R$. For any $\gamma \in \pi(C_\sigma)$ that intersects ρ where $\gamma \in R_2$, by definition $\gamma \notin R \backslash M_{RB}$. For any $\gamma \in \pi(C_\sigma)$ that does not intersect ρ, there is no edge between γ and ρ in $G[R \cup B]$. It follows that with one query we may rematch ρ in $O(\log^3 n)$ amortized time.

Since there are at most $O(|\mathcal{P}(\sigma)| + \log \psi)$ cells C_2 to consider, we can maintain every MBM in $O(|\mathcal{P}(\sigma)| \cdot \log^3 n + \log^5 n)$ time. ◀

8 Dynamic connectivity in square intersection graphs

Having formally introduced and analysed every component, we can now fully state what our data structure maintains. For an illustration, we refer back to Figure 2. We store a data structure that uses at most $O(n \log^3 n \log \psi)$ space:

(1) We store $\mathcal{S}$ in a quadtree $T(\mathcal{S})$.
- This quadtree contains for each cell $\sigma \in \mathcal{S}$ the neighborhood $\mathcal{N}(\sigma)$. Additionally, we
- maintain all $C \in \mathcal{C}(\mathcal{S})$ with $O(|\mathcal{C}(\sigma)| \cdot \log^4 n + \log^6 n)$ amortized time (Thm 7).
- This quadtree requires $O(n \log \psi)$ space (Thm 7).

(2) We augment our quadtree with a Marked-Ancestor Tree (MAT).
- We mark each cell $C \in \mathcal{C}(\mathcal{S})$ in the MAT (Thm 7).

(3) For any storing cell C, we define a conflict set R_i for all cells $C_i \in \mathcal{P}(\pi(C))$. We store $\pi(C)$ with the conflict sets in our square intersection data structure $\mathcal{D}^*(\pi(C))$.
- Let $z^* = \sum_C \sum_i |R_i|$, the total space required is $O((n + z^*) \log^3 n)$ (Lem 8).
- By the proof of Theorem 9, $z^* \in O(n \log \psi)$ so we use $O(n \log \psi \log^3 n)$ total space.

(4) For each storing cell C_1 and each $C_2 \in \mathcal{P}(\pi(C_1))$, we store a Maximal Bichromatic Matching (MBM) in $G[R \cup B]$.
- We set R as the set of squares in C_1 that have C_2 in their perimeter ($R \leftarrow \{\gamma \in \pi(C_1) \mid C_2 \in \mathcal{P}(\gamma)\}$ and $B \leftarrow \pi(C_2)$.
- Updates in $\mathcal{S}$ require $O(|\mathcal{P}(\sigma)| \cdot \log^3 n + \log^5 n)$ amortized time (Thm 9).

(5) Finally, for any pair (C_1, C_2), if their MBM is not empty, we store an edge between them.
- We maintain the resulting "proxy graph" in the HLT data structure [14].
- Inserting or deleting a square σ introduces at most $O(|\mathcal{P}(\sigma)| + \log \psi)$ new edges.

We finally show that this data structure implies the following:

▶ **Theorem 10.** *Let $\mathcal{S}$ be a set of squares with adaptive aspect ratio ψ. We can store $\mathcal{S}$ in a dynamic data structure of size $O(n \log^3 n \log \psi)$ with $O((|\mathcal{C}(\sigma)| + |\mathcal{P}(\sigma)|) \log^4 n + \log^6 n)$ amortized deterministic update time such that for any pair of squares (σ, ρ) we can query for the connectivity between σ and ρ in $O(\log n / \log \log n)$ time.*

Proof. Our pipeline functions identical to the pipeline of [16]. Given σ, we obtain a pointer to its storing cell C_σ in $O(1)$ time. We then query the marked-ancestor tree in $O(\log n / \log \log n)$ time to find the largest ancestor C_α of C_σ that is marked. The cell C_α is marked by at least one squares γ that contains C_α in its interior. We obtain a pointer to γ and its storing cell C^* in $O(1)$ time. We note that if there is also some squares γ' that marked C_α, we may arbitrarily get a pointer to either γ or γ'. Doing the same procedure for ρ gives a cell R^*. We

test whether C^* and R^* are connected in the proxy graph $O(\log n / \log \log n)$ time. We now claim that these two cells are connected in the proxy graph if and only if (ρ, σ) are connected. The key observation to prove this claim is that, if we were to rescale the plane, our graph contains the proxy graph maintained by Kaplan et al. [16] as a subgraph. Indeed at the time of a query, ψ is fixed. Thus, we may rescale the plane such that every square has a diameter in $[\frac{1}{\psi}, 1]$. Let $H(\mathcal{S})$ be the quadtree of [16], then $T(\mathcal{S}) \subset H(\mathcal{S})$. Kaplan et al. maintain for every pair (C_1, C_2) with $C_2 \in \mathbb{P}^*(\pi(C_1))$ an Maximal Bichromatic Matching in the graph $G[R' \cup B']$ for $R' \leftarrow \{\gamma \in \pi(C_1) \mid C_2 \in \mathbb{P}^*(\gamma)\}$ and $B' \leftarrow \pi(C_2)$. For each nonempty MBM between a pair (C_1, C_2), they store an edge in the proxy graph.

Note that if the MBM is nonempty, then both C_1 and C_2 are storing cells. It follows that $C_2 \in \mathcal{P}(\pi(C_1))$; and that $R' = R$. Thus, we store for each non-empty MBM a maximal bichromatic matching in the graph $G[R \cup B] = G[R' \cup B']$ as in [16]. This implies that after rescaling, whenever there exists an edge in the proxy graph of [16], there exists an edge in our data structure. Thus, we may immediately apply the proof of Theorem 4.3 in [16] to conclude that (σ, ρ) are connected if and only if (C^*, R^*) are. ◀

References

1 Anders Aamand, Adam Karczmarz, Jakub Lacki, Nikos Parotsidis, Peter M. R. Rasmussen, and Mikkel Thorup. Optimal decremental connectivity in non-sparse graphs. *CoRR*, abs/2111.09376, 2021. `arXiv:2111.09376`.

2 Stephen Alstrup, Thore Husfeldt, and Theis Rauhe. Marked ancestor problems. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. no. 98CB36280)*, pages 534–543. IEEE, 1998.

3 Stephen Alstrup, Jens P. Secher, and Maz Spork. Optimal on-line decremental connectivity in trees. *Inf. Process. Lett.*, 64(4):161–164, 1997. `doi:10.1016/S0020-0190(97)00170-1`.

4 Arne Andersson. General balanced trees. *J. Algorithms*, 30(1):1–18, 1999. `doi:10.1006/jagm.1998.0967`.

5 Kevin Buchin, Maarten Löffler, Pat Morin, and Wolfgang Mulzer. Preprocessing imprecise points for delaunay triangulation: Simplified and extended. *Algorithmica*, 61(3):674–693, 2011.

6 Timothy M Chan and Zhengcheng Huang. Dynamic geometric connectivity in the plane with constant query time. *arXiv preprint arXiv:2402.05357*, 2024.

7 Timothy M. Chan, Mihai Pătraşcu, and Liam Roditty. Dynamic connectivity: Connecting to networks and geometry. *SIAM J. Comput.*, 40(2):333–349, 2011. `doi:10.1137/090751670`.

8 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: Algorithms and applications, 3rd Edition*. Springer, 2008. URL: `https://www.worldcat.org/oclc/227584184`.

9 David Eppstein. Dynamic generators of topologically embedded graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 599–608. ACM/SIAM, 2003. URL: `http://dl.acm.org/citation.cfm?id=644108.644208`.

10 David Eppstein, Zvi Galil, Giuseppe F Italiano, and Thomas H Spencer. Separator based sparsification for dynamic planar graph algorithms. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of Computing*, pages 208–217, 1993.

11 Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.*, 14(4):781–798, 1985. `doi:10.1137/0214055`.

12 Sariel Har-Peled. Quadtrees-hierarchical grids. *Lecture notes*, 2010.

13 Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999. `doi:10.1145/320211.320215`.

14 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.

15 Jacob Holm and Eva Rotenberg. Good r-divisions imply optimal amortized decremental biconnectivity. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 42:1–42:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.42.

16 Haim Kaplan, Alexander Kauer, Katharina Klost, Kristin Knorr, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Dynamic connectivity in disk graphs. In *38th International Symposium on Computational Geometry (SoCG 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

17 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar voronoi diagrams for general distance functions and their algorithmic applications. *Discrete & Computational Geometry*, 64(3):838–904, 2020.

18 Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1131–1142. SIAM, 2013. doi:10.1137/1.9781611973105.81.

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

20 Fabian Kuhn, Rogert Wattenhofer, and Aaron Zollinger. Ad-hoc networks beyond unit disk graphs. In *Proceedings of the 2003 Joint Workshop on Foundations of Mobile Computing*, DIALM-POMC '03, pages 69–78, New York, NY, USA, 2003. Association for Computing Machinery. doi:10.1145/941079.941089.

21 Jakub Lacki and Piotr Sankowski. Optimal decremental connectivity in planar graphs. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 608–621. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.608.

22 Chih-Hung Liu. Nearly optimal planar k nearest neighbors queries under general distance functions. *SIAM Journal on Computing*, 51(3):723–765, 2022.

23 Maarten Löffler, Joseph A Simons, and Darren Strash. Dynamic planar point location with sub-logarithmic local updates. In *Algorithms and Data Structures: 13th International Symposium, WADS 2013, London, ON, Canada, August 12-14, 2013. Proceedings 13*, pages 499–511. Springer, 2013.

24 Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 950–961. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.92.

25 Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

26 Robert Endre Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *J. Comput. Syst. Sci.*, 18(2):110–127, 1979. doi:10.1016/0022-0000(79)90042-4.

27 Robert Endre Tarjan and Jan van Leeuwen. Worst-case analysis of set union algorithms. *J. ACM*, 31(2):245–281, 1984. doi:10.1145/62.2160.

28 Mikkel Thorup. Decremental dynamic connectivity. *J. Algorithms*, 33(2):229–243, 1999. doi:10.1006/jagm.1999.1033.

29 Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 343–350. ACM, 2000. doi:10.1145/335305.335345.

30 Dan E Willard and George S Lueker. Adding range restriction capability to dynamic data structures. *Journal of the ACM (JACM)*, 32(3):597–617, 1985.