Fully Dynamic Maximum Independent Sets of Disks in Polylogarithmic Update Time

Sujoy Bhore
Department of Computer Science and Engineering, Indian Institute of Technology Bombay, Mumbai, India

Martin Nöllenburg
Institute of Logic and Computation, Algorithms and Complexity Group, TU Wien, Vienna, Austria

Csaba D. Tóth
Department of Mathematics, California State University Northridge, Los Angeles, CA; and Department of Computer Science, Tufts University, Medford, MA, USA

Jules Wulms
Department of Mathematics and Computer Science, TU Eindhoven, Eindhoven, Netherlands

Abstract

A fundamental question is whether one can maintain a maximum independent set (MIS) in polylogarithmic update time for a dynamic collection of geometric objects in Euclidean space. For a set of intervals, it is known that no dynamic algorithm can maintain an exact MIS in sublinear update time. Therefore, the typical objective is to explore the trade-off between update time and solution size. Substantial efforts have been made in recent years to understand this question for various families of geometric objects, such as intervals, hypercubes, hyperrectangles, and fat objects.

We present the first fully dynamic approximation algorithm for disks of arbitrary radii in the plane that maintains a constant-factor approximate MIS in polylogarithmic expected amortized update time. Moreover, for a fully dynamic set of n unit disks in the plane, we show that a 12-approximate MIS can be maintained with worst-case update time $O(\log n)$, and optimal output-sensitive reporting. This result generalizes to fat objects of comparable sizes in any fixed dimension d, where the approximation ratio depends on the dimension and the fatness parameter. Further, we note that, even for a dynamic set of disks of unit radius in the plane, it is impossible to maintain $O(1+\varepsilon)$-approximate MIS in truly sublinear update time, under standard complexity assumptions.

Our results build on two recent technical tools: (i) The MIX algorithm by Cardinal et al. (ESA 2021) that can smoothly transition from one independent set to another; hence it suffices to maintain a family of independent sets where the largest one is an $O(1)$-approximate MIS. (ii) A dynamic nearest/farthest neighbor data structure for disks by Kaplan et al. (DCG 2020) and Liu (SICOMP 2022), which generalizes the dynamic convex hull data structure by Chan (JACM 2010), and quickly yields a "replacement" disk (if any) when a disk in one of our independent sets is deleted.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Dynamic algorithm, Independent set, Geometric intersection graph

Related Version *A full version of this paper is available at*: `arXiv:2308.00979`

Funding *Csaba D. Tóth*: Research was partially supported by the NSF award DMS-2154347.

1 Introduction

The maximum independent set (MIS) problem is one of the most fundamental problems in theoretical computer science, and it is one of Karp's 21 classical NP-complete problems [29]. In the MIS problem, we are given a graph $G = (V, E)$, and the objective is to choose a subset of the vertices $S \subseteq V$ of maximum cardinality such that no two vertices in S are adjacent. The intractability of MIS carries even under strong algorithmic paradigms. For instance, it is known to be hard to approximate: no polynomial-time algorithm can achieve

© Sujoy Bhore, Martin Nöllenburg, Csaba D. Tóth, and Jules Wulms;
licensed under Creative Commons License CC-BY 4.0

an approximation factor $n^{1-\varepsilon}$ (for $|V| = n$ and a constant $\varepsilon > 0$) unless P=ZPP [39]. In fact, even if the maximum degree of the input graph is bounded by 3, no polynomial-time approximation scheme (PTAS) is possible [7].

Geometric Independent Set. In geometric settings, the input to the MIS problem is a collection $\mathcal{L} = \{\ell_1, \ldots, \ell_n\}$ of geometric objects, e.g., intervals, disks, squares, rectangles, etc., and we wish to compute a maximum independent set in their intersection graph G. That is, each vertex in G corresponds to an object in $\mathcal{L}$, and two vertices form an edge if and only if the two corresponding objects intersect. The objective is to choose a maximum cardinality subset $\mathcal{L}' \subseteq \mathcal{L}$ of independent (i.e., pairwise disjoint) objects.

A large body of work has been devoted to geometric MIS problems, due to their wide range of applications in scheduling [4], VLSI design [26], map labeling [1], data mining [30, 6], and many others. Stronger theoretical results are known for the geometric MIS problem: For instance, even for unit disks in the plane, the problem remains NP-hard [18] and W[1]-hard [34], but it admits a PTAS [26]. Later, PTASs were also developed for arbitrary disks, squares, and more generally hypercubes and fat objects in constant dimensions [27, 14, 2, 21].

In their seminal work, Chan and Har-Peled [17] showed that for an arrangement of pseudo-disks,[1] a local-search-based approach yields a PTAS. However, for non-fat objects, the scenario is quite different. For instance, it had been a long-standing open problem to find a constant-factor approximation algorithm for the MIS problem on axis-aligned rectangles. In a recent breakthrough, Mitchell [36] answered this question in the affirmative. Through a refined analysis of the recursive partitioning scheme, a dynamic programming algorithm finds a constant-factor approximation. This constant factor was subsequently improved to 3 [22].

Dynamic Geometric Independent Set. In dynamic settings, objects are inserted into or deleted from the collection $\mathcal{L}$ over time. The typical objective is to achieve (almost) the same approximation ratio as in the offline (static) case while keeping the update time, i.e., the time to update the solution after insertion/deletion, as small as possible. We call it the *Dynamic Geometric Maximum Independent Set* problem (for short, DGMIS).

Henzinger et al. [25] studied DGMIS for various geometric objects, such as intervals, hypercubes, and hyperrectangles. Many of their results extend to the weighted version of DGMIS, as well. Based on a lower bound of Marx [35] for the offline problem, they showed that any dynamic $(1+\varepsilon)$-approximation for squares in the plane requires $\Omega(n^{1/\varepsilon})$ update time for any $\varepsilon > 0$, ruling out the possibility of sub-polynomial time dynamic approximation schemes. On the positive side, they obtained dynamic algorithms with update time polylogarithmic in both n and N, where the corners of the objects are in a $[0, N]^d$ integer grid, for any constant dimension d. Gavruskin et al. [23] studied DGMIS for intervals in $\mathbb{R}$ under the assumption that no interval is contained in another interval and obtained an optimal solution with $O(\log n)$ amortized update time. Bhore et al. [8] presented the first fully dynamic algorithms with polylogarithmic update time for DGMIS, where the input objects are intervals and axis-aligned squares. For intervals, they presented a fully dynamic $(1+\varepsilon)$-approximation algorithm with logarithmic update time. Later, Compton et al. [19] achieved a faster update time for intervals, by using a new partitioning scheme. Recently, Bhore et al. [10] studied the MIS problem for intervals in the streaming settings, and obtained lower bounds. Very recently, Bhore and Chan [9] showed that the complement of DGMIS, i.e., dynamic vertex cover, can be maintained efficiently for a wide class of geometric objects.

[1] In an arrangement of pseudo-disks the boundaries of each pair of objects intersects at most twice.

For axis-aligned squares in $\mathbb{R}^2$, Bhore et al. [8] presented a randomized algorithm with an expected approximation ratio of roughly 2^{12} (generalizing to roughly 2^{2d+5} for d-dimensional hypercubes) with amortized update time $O(\log^5 n)$ (generalizing to $O(\log^{2d+1} n)$ for hypercubes). Moreover, Bhore et al. [11] studied the DGMIS problem in the context of dynamic map labeling and presented dynamic algorithms for several subfamilies of rectangles that also perform well in practice. Cardinal et al. [13] designed dynamic algorithms for fat objects in fixed dimension d with sublinear worst-case update time. Specifically, they achieved $\tilde{O}(n^{3/4})$ update time[2] for disks in the plane, and $\tilde{O}(n^{1-\frac{1}{d+2}})$ for Euclidean balls in $\mathbb{R}^d$.

However, despite the remarkable progress on the DGMIS problem in recent years, the following question remained unanswered.

▶ **Question 1.** *Does an algorithm exist that, for a given dynamic set of disks in the plane, maintains a constant-factor approximate MIS in polylogarithmic update time?*

Our Contributions In this paper, we answer Question 1 in the affirmative (Theorems 1–3); see Table 1. As a first step, we address the case of unit disks in the plane.

Objects	Approx. Ratio	Update time	Reference
Intervals	$1+\varepsilon$	$O(\varepsilon^{-1}\log n)$	[19]
Squares	$O(1)$	$O(\log^5 n)$ amortized	[8]
Arbitrary radii disks	$O(1)$	$(\log n)^{O(1)}$ expec. amortized	Theorem 3
Unit disks	$O(1)$	$O(\log n)$ worst-case	Theorem 1
	$1+\varepsilon$	$n^{(1/\varepsilon)^{\Omega(1)}}$	Theorem 4
f-fat objects in $\mathbb{R}^d$	$O_f(1)$	$O_f(\log n)$ worst-case	Theorem 2
d-dimensional hypercubes	$(1+\varepsilon)\cdot 2^d$	$O_{d,\varepsilon}(\log^{2d+1} n \cdot \log^{2d+1} U)$	[25]

Table 1 Summary of results on dynamic independent sets for n geometric objects.

▶ **Theorem 1.** *For a fully dynamic set of unit disks in the plane, a 12-approximate MIS can be maintained with worst-case update time $O(\log n)$, and optimal output-sensitive reporting.*

We prove Theorem 1 in the full version [12]. Similarly to classical approximation algorithms for the static problem [26], we lay out four shifted grids such that any unit disk lies in a grid cell for at least one of the grids, see Figure 1. For each grid, we maintain an independent set that contains at most one disk from each grid cell, thus we obtain four independent sets $S_1, \ldots, S_4$ at all times, where the largest one is a constant-factor approximation of the MIS. Using the MIX algorithm [13], we can maintain an independent set $S \subset \bigcup_{i=1}^4 S_i$ of size $\Omega(\max\{|S_1|, |S_2|, |S_3|, |S_4|\})$ at all times, which is a $O(1)$-approximation of the MIS.

Moreover, our dynamic data structure for unit disks easily generalizes to fat objects of comparable sizes in $\mathbb{R}^d$ for any constant dimension $d \in \mathbb{N}$ (see the full version [12]).

▶ **Theorem 2.** *For every $d, f \in \mathbb{N}$ and real parameters $0 < r_1 < r_2$, there exists a constant C with the following property: For a fully dynamic collection of f-fat sets in $\mathbb{R}^d$, each of size between r_1 and r_2, a C-approximate MIS can be maintained with worst-case update time $O(\log n)$, and optimal output-sensitive reporting.*

[2] The $\tilde{O}(\cdot)$ notation ignores logarithmic factors.

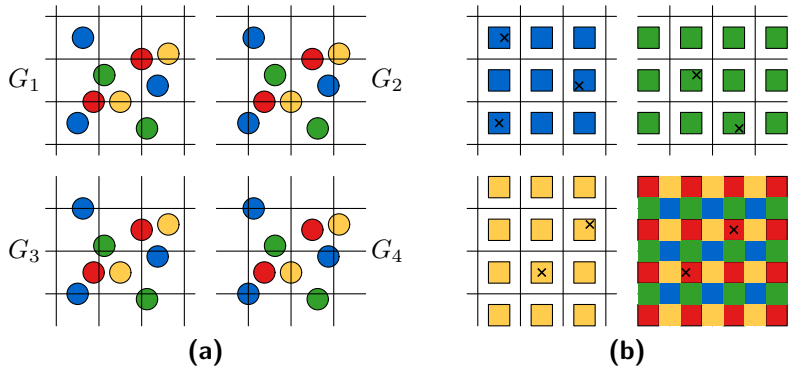

Figure 1 (a) The four shifted grids $G_1, \ldots, G_4$, which respectively do not intersect the blue, green, yellow, and red disks. **(b)** The radius-1 squares inside grid cells, along with the center points of the disks that lie completely inside grid cells, as crosses. In the bottom right, besides red squares for G_4, the squares of all other grids are added to show that the squares together partition the plane.

Our main result is a dynamic data structure for MIS of disks of arbitrary radii in $\mathbb{R}^2$.

▶ **Theorem 3.** *For a fully dynamic set of disks of arbitrary radii in the plane, an $O(1)$-approximate MIS can be maintained in polylogarithmic expected amortized update time.*

To prove Theorem 3, we extend the core ideas developed for unit disks with several new ideas, see Section 3. Specifically, we still maintain a constant number of "grids" such that every disk lies in one of the grid cells. Due to the different disk sizes, however, we need shifted grids at multiple scales, where each disk lies in a grid cell of comparable size. We achieve this with a new *nonatree* data structure, which recursively subdivides squares into 3×3 congruent subsquares. For each shifted nonatree, we maintain an independent set S_i that contains at most one disk from each cell. While the set S_i can be computed in a bottom-up traversal of the nonatree using the greedy strategy [33, 20], the challenge is to perform dynamic updates in polylogarithmic update time even though ascending paths in the nonatrees can be of linear length. We address this challenge with a combination of techniques outlines in Section 3.2. One key component is the use of the dynamic farthest neighbor data structure by Kaplan et al. [28] (which generalizes Chan's famous dynamic convex hull data structure [15, 16]). We adapt this data structure to work in concert with nonatrees to find the next level where we can add another disk of the same or larger size to greedily add to the independent set in polylogarithmic time, and with polylogarithmic expected amortized update time. Finally, we use again the MIX algorithm for disks in the plane [13] to maintain a single independent set $S \subset \bigcup_i S_i$, which is a constant-factor approximation of the MIS.

One bottleneck in this framework is the farthest neighbor data structure [28, 32], which provides *expected amortized* polylogarithmic update time and works only for families of "nice" objects in the plane (such as disks or homothets of a convex polygon, etc.). This is the only reason why our algorithm does not guarantee *deterministic worst-case* update time, and it does not extend to balls in $\mathbb{R}^d$ for $d \geq 3$, or to arbitrary fat objects in the plane. All other steps of our machinery support deterministic polylogarithmic worst-case update time, as well as balls in $\mathbb{R}^d$ for any constant dimension $d \in \mathbb{N}$, and fat objects in the plane.

Another limitation for generalizing our framework is the MIX algorithm, which smoothly transitions from one independent set to another. It was established by Cardinal et al. [13] for fat objects in $\mathbb{R}^d$ for any constant $d \in \mathbb{N}$ and their proof heavily relies on separator theorems. However, they show, e.g., that a sublinear MIX algorithm is impossible for rectangles in $\mathbb{R}^2$.

Finally, we note that, even for a dynamic set of unit disks in the plane, it is impossible to maintain a $(1+\varepsilon)$-approximate MIS with amortized update time $n^{O((1/\varepsilon)^{1-\delta})}$ for any ε, $\delta > 0$, unless the Exponential Time Hypothesis (ETH) fails. This follows from a reduction to a result by Marx [35], resembling the same result for hypercubes by Henzinger et al. [25].

▶ **Theorem 4.** *For a fully dynamic set of unit disks in $\mathbb{R}^2$, no $(1+\varepsilon)$-approximation algorithm exists for DGMIS with amortized update time $n^{O((1/\varepsilon)^{1-\delta})}$, for any $\varepsilon, \delta > 0$, unless ETH fails.*

Due to space constraints, some details are deferred to the full paper on ArXiv [12].

2 Preliminaries

MIX Algorithm. A general strategy for computing an MIS is to maintain a small number of *candidate* independent sets $S_1, \ldots, S_k$ with a guarantee that the largest set is a good approximation of an MIS, and each insertion and deletion incurs only constantly many changes in S_i for all $i = 1, \ldots, k$. To answer a query about the size of the MIS, we can simply report $\max\{|S_1|, \ldots, |S_k|\}$ in $O(k)$ time. Similarly, we can report an entire (approximate) MIS by returning a largest candidate set. However, if we need to maintain a single (approximate) MIS at all times, we need to smoothly switch from one candidate to another. This challenge has recently been addressed by the MIX algorithm introduced by Cardinal et al. [13]:

> **MIX algorithm**: The algorithm receives two independent sets S_1 and S_2 whose sizes sum to n as input, and smoothly transitions from S_1 to S_2 by adding or removing one element at a time such that at all times the intermediate sets are independent sets of size at least $\min\{|S_1|, |S_2|\} - o(n)$.

Cardinal et al. [13] constructed an $O(n \log n)$-time MIX algorithm for fat objects in $\mathbb{R}^d$, for constant dimension $d \in \mathbb{N}$, which we use as follows. Assume that $\mathcal{D}$ is a fully dynamic set of disks in the plane, and we are given candidate independent sets $S_1, \ldots, S_k$ with the guarantee that $\max\{|S_1|, \ldots, |S_k|\} \geq c \cdot \mathrm{OPT}$ at all times, where OPT is the size of the MIS and $0 < c \leq 1$ is a constant; further assume that the size of S_i, $i \in \{1, \ldots, k\}$, changes by at most a constant $u \geq 1$ for each insertion or deletion in $\mathcal{D}$. We show that MIX can be used to maintain a single approximate MIS S at all times, when we are allowed to make up to $10u$ changes in S for each insertion or deletion in $\mathcal{D}$.

▶ **Lemma 5.** *For a collection of candidate independent sets $S_1, \ldots, S_k$, the largest of which is a c-approximate MIS at all times, we can dynamically maintain an $O(c)$-approximate MIS with $O(1)$ changes per update.*

Dynamic Farthest Neighbor Data Structure. Given a set of functions $\mathcal{F} = \{f_1, \ldots, f_n\}$, $f_i : \mathbb{R}^2 \to \mathbb{R}$ for $i = 1, \ldots, n$, the *upper envelope* of $\mathcal{F}$ is the graph of the function $U : \mathbb{R}^2 \to \mathbb{R}$, $L(p) = \max\{f_i(p) \mid 1 \leq i \leq n\}$. A *vertical stabbing query* with respect to the upper envelope, for query point $p \in \mathbb{R}^2$, asks for the function f_i such that $U(p) = f_i(p)$.

Given a set $\mathcal{D}$ of n disks in the plane, we can use this machinery to find, for a query disk d_q, the disk in $\mathcal{D}$ that is farthest from d_q. Specifically, for each disk $d \in \mathcal{D}$ centered at c_d with radius r_d, define the signed Euclidean distance function $f_d : \mathbb{R}^2 \to \mathbb{R}$, $f_d(p) = |pc_d| - r_d$, which is negative if p is in the interior of d, positive if p is in the exterior of d and 0 if p is on the boundary of d. For $\mathcal{F} = \{f_d \mid d \in \mathcal{D}\}$ we have $U(p) = f_d(p)$ for a disk $d \in \mathcal{D}$ farthest from p. Importantly, for query disk d_q, we find a farthest disk from d_q by querying its center.

In the fully dynamic setting, functions are inserted and deleted to/from $\mathcal{F}$, and we wish to maintain a data structure that supports vertical stabbing queries w.r.t. the upper envelope of $\mathcal{F}$. For linear functions f_i (i.e., hyperplanes in $\mathbb{R}^3$), Chan [15] devised a fully dynamic randomized data structure with polylogarithmic query time and polylogarithmic amortized expected update time. After several incremental improvements, the current best version is a deterministic data structure for n hyperplanes in $\mathbb{R}^3$ with $O(n \log n)$ preprocessing time, $O(\log^4 n)$ amortized update time, and $O(\log^2 n)$ worst-case query time [16].

Kaplan et al. [28] generalized Chan's data structure for dynamic sets of functions $\mathcal{F}$, where the upper (or lower) envelope of any k functions has $O(k)$ combinatorial complexity. This includes, in particular, the signed distance functions from disks [3]. Their data structure requires $O(n \log^3 n)$ storage in expectation and supports insertions in $O(\lambda_s(\log n) \log^5 n)$ amortized expected time, deletions in $O(\lambda_s(\log n) \log^9 n)$ amortized expected time, and vertical stabbing queries in $O(\log^2 n)$ worst-case deterministic time. Here n is the number of functions currently in $\mathcal{F}$ and $\lambda_s(t)$ is the maximum length of a Davenport-Schinzel sequence [37] on t symbols of order s. Subsequently, Liu [32, Corollary 16] improved the deletion time to $O(\lambda_s(\log n) \log^7 n)$ amortized expected time. For signed Euclidean distances of disks, we have $s = 6$ [28] and $\lambda_6(t) \ll O(t \log t) \ll O(t^2)$. For simplicity, we assume $O(\log^9 n)$ expected amortized update time and $O(\log^2 n)$ worst-case query time. We obtain the following.

▶ **Lemma 6.** *For a dynamic set $\mathcal{D}$ of n disks of arbitrary radii in the plane, there is a randomized data structure that supports disk insertion in $O(\log^7 n)$ amortized expected time, disk deletion in $O(\log^9 n)$ amortized expected time, and the following disjointness query in $O(\log^2 n)$ worst-case time: For a query disk d_q, find a disk in $\mathcal{D}$ disjoint from d_q, or report that all disks in $\mathcal{D}$ intersect d_q.*

We refer to the data structure in Lemma 6 as the *dynamic farthest neighbor* (DFN) data structure. We remark that Chan [16] improved the update time when the functions $\mathcal{F} = \{f_1, \ldots, f_n\}$ are distances from n *point sites* in $\mathbb{R}^2$. De Berg and Staals [5] generalized these results to dynamic k-nearest neighbor data structures for n point sites in $\mathbb{R}^2$.

3 Disks of Arbitrary Radii in the Plane

In this section, we study the DGMIS problem for a set of disks of arbitrary radii. The general idea of our new data structure is to break the set of disks $\mathcal{D}$ into subsets of disks of comparable radius. We will use several instances of shifted grids $G_1^i, \ldots, G_4^i$, as we also use in the unit disk case, where the grid cells now have side length 3^i, and are shifted by $\frac{3^i}{2}$, for $i \in \mathbb{Z}$. The resulting hierarchies of recursively 3×3 subdivided grid cells forms so-called nonatrees. In Section 3.1, we explain how to compute a constant-factor approximation for static instances by bottom-up traversals of the nonatrees. We then make several changes in the static data structures, to support efficient updates, while maintaining a constant factor approximation. Since the height of a nonatree (even a compressed nonatree) may be $\Theta(n)$ for n disks (see Figure 2), we cannot afford to traverse ascending paths in their entirety with our polylogarithmic budget for the update time. We address this challenge with a combination of the techniques outlined in Section 3.2. One key component is the use of the dynamic farthest neighbor data structure of Lemma 6. Finally, in Section 3.3, we stitch all these ingredients together to show how to maintain a constant-factor approximate maximum independent set in a fully dynamic setting, with expected amortized polylogarithmic update time.

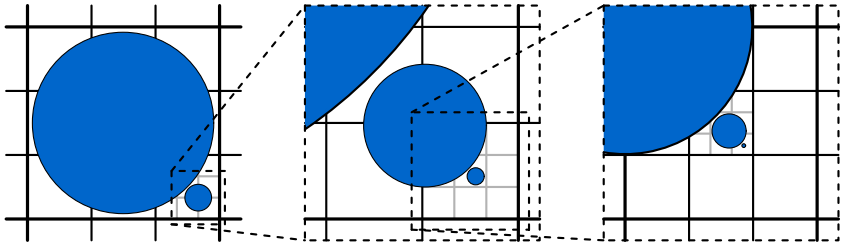

Figure 2 A nonatree with height linear in the number of stored disks, whose radii decay exponentially. A compressed nonatree (with compressed nodes) has linear height.

3.1 Static Hierarchical Data Structures

Dividing Disks over Buckets. The shifted grids $G_1^i, \ldots, G_4^i$ form the set $\mathcal{G}_i$. In this set $\mathcal{G}_i$ we store disks with radius r, where $\frac{3^{i-1}}{4} < r \leq \frac{3^i}{4}$. We refer to the data structures associated with one value i as the *bucket* i. Compared to the unit disk case, in which we consider only disks of radius $\frac{1}{4}$ times the side length of the grid cells, we now have to deal with disks of varying sizes even in one set $\mathcal{G}_i$ of shifted grids. In both cases, every disk is completely inside at least one grid cell. To see this, observe that no two vertical or two horizontal grid lines in one grid of bucket i can intersect a single disk with a radius lying in the range $(\frac{3^{i-1}}{4}, \frac{3^i}{4}]$. Indeed, such disks have a diameter at most $\frac{3^i}{2}$, while grid lines are at least 3^i apart.

Furthermore, our choice for side length 3^i for bucket i was not arbitrary: Consider also adjacent bucket $i-1$ and observe that each cell c of grid G_1^i is further subdivided into nine cells of grid G_1^{i-1}, in a 3×3 formation. We say that c is *aligned* with the nine cells in bucket $i-1$. We define the same parent-child relations as in a quadtree: If a grid cell c in a lower bucket is inside a cell c_p of an adjacent higher bucket, we say that c is a child (cell) of c_p, or that c_p is the parent (cell) of c. In general, we write $c_1 \prec c_2$ if cell c_1 is a descendant of cell c_2; $c_1 \preceq c_2$ if equality is allowed. We call the resulting structure a *nonatree*, and we will refer to the nonatree that relates all grids G_1^j as N_1. In Figure 3a we illustrate the grids of two consecutive buckets in a nonatree.

Crucially, all grids G_2^j also align, and the same holds for G_3^j and G_4^j. This happens because horizontally and vertically, grid cells are subdivided into an odd number of cells (three in our case), and the shifted grids are displaced by half the side length of the grid cells. Thus, for G_2^j and G_4^j, the horizontal shift in buckets i and $i-1$ ensures that every third vertical grid line of bucket $i-1$ aligns with a vertical grid line of bucket i. The exact same happens for the horizontal grid lines of G_3^j and G_4^j, due to the vertical shift. Thus, the horizontally shifted grids also form a nonatree N_2, and similarly, we define N_3 and N_4.

For each bucket i, we maintain five self-balancing search trees. Let $\mathcal{D}_i \subseteq \mathcal{D}$ be the subset of disks stored in $\mathcal{G}_i$ and let $S_1, \ldots, S_4$ be an independent set in $G_1^i, \ldots, G_4^i$, then we maintain in $T_{\mathcal{D}}^i$ all disks in $\mathcal{D}_i$ and in $T_1^i, \ldots, T_4^i$ the disks in $S_1, \ldots, S_4$.

Approximating a Maximum Independent Set. We will now use the data structures to compute an approximate MIS for disks with arbitrary radii. Note that, we defined buckets for $i \in \mathbb{Z}$, but we will use only those buckets that store any disks, which we call *relevant* buckets. Within these buckets, we call grid cells that contain disks the *relevant* grid cells. Figure 3 illustrates the concepts introduced in this paragraph and the upcoming paragraphs.

Let B be the sequence of relevant buckets, ordered on their parameter i. To compute a solution, we will consider the buckets in B in ascending order, starting from the lowest bucket, which holds the smallest disk, and has grids with the smallest side length, up to the highest

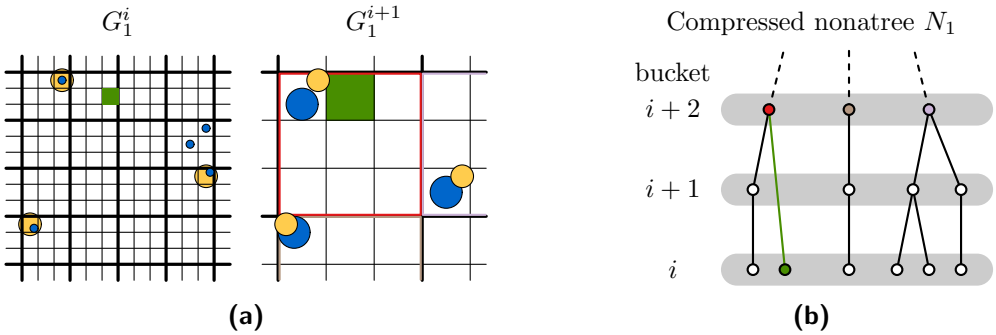

Figure 3 (a) Two compatible grids in buckets i and $i+1$, with (blue) disks of D in relevant cells. In particular, the green cell in G_1^i is relevant, but its (green) parent cell in G_1^{i+1} is not. Three (yellow) obstacle disks of G_1^i are drawn in both grids. Only one blue disk in G_1^{i+1} is disjoint from an obstacle, and can be chosen in the greedy bottom-up strategy. **(b)** Part of the compressed nonatree N_1 corresponding to **(a)**: The colored nodes of bucket $i+2$ correspond to colored squares in **(a)** of the same color. Because the green cell in G_1^{i+1} is not relevant, and does not have relevant children in two subtrees, it is not represented in N_1. Instead, the green node, corresponding to the green relevant cell in G_1^i, directly connects to an ancestor in bucket $i+2$ (by the green edge).

bucket with the largest disks, and largest side lengths. We follow a greedy bottom-up strategy for finding a constant-factor approximation of an MIS of disks. To prevent computational overhead in this approach, our nonatrees are *compressed*, similar to compressed quadtrees [24, Chapter 2]: Each nonatree consists of a root cell, all relevant grid cells, and all cells that have relevant grid cells in at least two subtrees. As such, each (non-root) internal cell of our nonatrees either contains a disk, or merges at least two subtrees that contain disks, and hence the total number of cells in a compressed nonatree is linear in the number of disks it stores, which is upper bounded by $O(n)$.

Specifically, two high-level steps can be distinguished in our approach:

1. In the lowest relevant bucket, we simply select an arbitrary disk from each relevant grid cell. In other relevant buckets, we consider for each grid cell $c \in G_k^i$ the subdivision of c in G_k^j in the preceding relevant bucket $j < i$. We try to combine the independent set from the relevant child(ren) of c with at most one additional disk in c. To communicate upwards which disks have been included in our independent set, we use *obstacle disks* (these are not necessarily input disks; see the next step). Once all relevant cells have been handled, we output the largest independent set among the four sets computed for the shifted nonatrees $N_1, \ldots, N_4$, to get an $O(1)$-approximation, as shown in Lemmata 7–9.
2. The obstacle disk in the previous step may cover more area than the disks in the independent set of the children of c. Hence, we consider computing the obstacle disk only for independent sets originating from a single child cell. In this case, we choose as the obstacle the smallest disk covering the contributing child cell in question. The obstacle will then be of comparable size to that child cell, and hence also comparable to the contributed disk, intersecting at most a constant number of disks in the parent cell c. Otherwise, if the independent set of the children originates from more than one child, we simply do not add a disk from c, even if that may be possible. Lemmata 10 and 11 show that we still obtain a constant-factor approximate MIS under these constraints.

We will now elaborate on the high-level steps, and provide a sequence of lemmata that can be combined to prove the approximation ratio of the computed independent set.

In the first step, we deviate from an optimal solution in three ways: We follow a greedy bottom-up approach, we take at most one disk per grid cell, and we do not combine the solutions of the shifted nonatrees. We focus on the latter concern first, starting by defining the intersection between a disk and a nonatree, as follows. We say that a disk d intersects (the grid lines of) a nonatree N_k, if and only if its radius r_d is in the range $(\frac{3^{i-1}}{4}, \frac{3^i}{4}]$ and it intersects grid lines of G_k^i.

▶ **Lemma 7.** *For a set S of disks in $\mathbb{R}^2$, the grid lines of at least one nonatree, out of the shifted nonatrees $N_1, \ldots, N_4$, do not intersect at least $|S|/4$ disks.*

We show that taking only a single disk per grid cell into our solution is a 35-approximation of a MIS, using a simple packing argument [31]; see also [38].

▶ **Lemma 8.** *If S is a MIS of the disks in a grid cell of a nonatree N_k, then $|S| \leq 35$.*

To round out the first step, we prove that our greedy strategy contributes at most a factor 5 to our approximation factor.

▶ **Lemma 9.** *Let S be a maximum independent set of the disks in a nonatree N_k such that each grid cell in N_k contributes at most one disk. An algorithm that considers the grid cells in N_k in bottom-up fashion, and computes an independent set S' by greedily adding at most one non-overlapping disk per grid cell to S', is a 5-approximation of S.*

For the second step, we use several data structures and algorithmic steps that help us achieve polylogarithmic update and query times in the dynamic setting. For now we analyze solely the approximation factor incurred by these techniques. We start by analyzing the approximation ratio for not taking any disk from a cell c, if several of its children contribute disks to the computed independent set.

▶ **Lemma 10.** *Let S be a MIS of the disks in a nonatree N_k such that each grid cell in N_k contributes at most one disk. The independent set $S' \subset S$, that contains all disks in S except disks from cells that have two relevant child cells, is a 2-approximation of S.*

Next we consider the obstacle disk that we compute when only one child cell contributes disks to the independent set. Before we elaborate on the approximation ratio of this algorithmic procedure, we first explain the steps in more detail.

For the leaf cells of a nonatree, it is unnecessary to compute an obstacle disk, since these cells contribute at most a single disk, which can act as its own obstacle disk. For a cell c that is an internal node of the nonatree, with at most one relevant child that contributes to the independent set, we have two options for the obstacle disk of c. We use the obstacle disk of the child cell to determine whether there is a disk in c disjoint from the child obstacle, to either find a disjoint disk d or not. If we find such a disk d, we compute a new obstacle disk for c, by taking the smallest enclosing disk of c. If there is no such disk d, then we use the obstacle disk of the child as the obstacle disk for c. This ensures that the obstacle disk does not grow unnecessarily, which is relevant when proving the following approximation factor.

▶ **Lemma 11.** *Let c be a cell in bucket i of nonatree N_k that contributes a disk to an independent set. The computed obstacle disk d_o can overlap with no more than 23 pairwise disjoint disks in higher buckets.*

▶ **Lemma 12.** *For a set of disks in the plane, one of our shifted nonatrees $N_1, \ldots, N_4$ maintains an independent set of size $\Omega(|\mathrm{OPT}|)$, where OPT is a MIS.*

3.2 Modifications to Support Dynamic Maintenance

In Section 3.1, we defined four hierarchical grids (nonatrees) $N_1, \ldots, N_4$, described a greedy algorithm to compute independent sets $S_1, \ldots, S_4$ that are consistent with the grids, and showed that a largest of the four independent sets is a constant-factor approximate MIS.

In this section, we make several changes in the static data structures, to support efficient updates, while maintaining a constant-factor approximation. Then in Section 3.3, we show that the modified data structures can be maintained dynamically in expected amortized polylogarithmic update time. We start with a summary of the modifications:

- **Sparsification.** We split each nonatree N_i, $i \in \{1, \ldots, 4\}$, into two trees N_i^{odd} and N_i^{even}, one containing the odd levels and the other containing the even levels. As a result, the radii of disks at different (non-empty) levels differ by at least a factor of 3.
- **Clearance.** For a radius-r disk d, let $3d$ denote the concentric disk of radius $3r$. Recall that our greedy strategy adds disks to an independent set S in a bottom-up traversal of a nonatree. When S contains a disk $d \in \mathcal{D}$, then larger disks that intersect $3d$ cannot be added to S. In particular, we use obstacle disks of the form $3d'$, where d' is the smallest enclosing disk of a cell. A simple volume argument shows that this modification still yields a constant-factor approximation. As a result, if a new disk is added, it intersects at most one larger disk in S. This simplifies the update operation in Section 3.3.
- **Obstacle Disks and Obstacle Cells.** In Section 3.1, we defined obstacle disks for the disks in S_k. To support dynamic updates, we use slightly larger obstacle disks, to implement the clearance in our data structures. These obstacle disks are associated with cells of the nonatree N_k, which are called obstacle cells (true obstacles). Cells of the nonatree with two or more children are also considered as obstacle cells (merge obstacles), thus the obstacle cells decompose each nonatree into ascending paths.
- **Barrier Disks.** The naïve approach for a dynamic update of the independent set S in a nonatree N would work as follows: When a new disk d is inserted or deleted, we find a nonatree N and a cell $c \in N$ associated with d; and then in an ascending path of N from c to the root, we re-compute the disks in S associated with the cells. Unfortunately, the height of the nonatree may be linear (recall Fig. 2), and we cannot afford to traverse an ascending path from c to the root. Instead, we run the greedy process only locally, on an ascending path of N between two cells $c_1 \prec c_2$ that contain disks $s_1, s_2 \in S$, respectively. The greedy process guarantees that new disks added to S are disjoint from any smaller disk in S, including s_1. However, the new disks might intersect the larger disk $s_2 \in S$. In this case, we remove s_2 from S, keep it as a "placeholder" in a set B of *barrier disks*, and ensure that $S \cup B$ remains a dominating set of $\mathcal{D}$.

Sparsification. Recall that for a set $\mathcal{D}$ of n disks, $\mathcal{D}_i$ denoted the subset of disks of radius r, where $\frac{3^{i-1}}{4} < r \leq \frac{3^i}{4}$, for all $i \in \mathbb{Z}$. Let $N_1, \ldots, N_4$, be the four nonatrees defined in Section 3.1. For every $k \in \{1, \ldots, 4\}$, we create two copies of N_k, denoted N_k^{even} and N_k^{odd}. For i even (resp., odd), we associate the disks in $\mathcal{D}_i$ to the nonatrees N_k^{even} (resp., N_k^{odd}). We state a simple corollary to Lemma 12.

▶ **Lemma 13.** *For a set of disks in $\mathbb{R}^2$, one of our shifted nonatrees $N_1, \ldots, N_8$ maintains an independent set of size $|\text{OPT}|/C$, where OPT is a MIS and C is an absolute constant.*

The advantage of partitioning the nonatrees into odd and even levels is the following.

▶ **Lemma 14.** *Let $d_1, d_2 \in \mathcal{D}$ be disks of radii $r_1, r_2 > 0$, respectively, associated with cells c_1 and c_2 in a nonatree N_k, $k \in \{1, \ldots, 8\}$. If $c_1 \prec c_2$, then $3\,r_1 < r_2$.*

Clearance. The guiding principle of the greedy strategy is that, if we add a disk d to the independent set, we exclude all larger disks that intersect d. For our dynamic algorithm, we wish to maintain a stronger property:

▶ **Definition 15.** *Let S be an independent set of the disks in a nonatree N_k such that each grid cell in N_k contributes at most one disk. For $\lambda \geq 1$, we say that S has λ-**clearance** if the following holds: If $d_1, d_2 \in S$ are associated with cells c_1 and c_2, resp., and $c_1 \prec c_2$, then d_2 is disjoint from $\lambda\, d_1'$, where d_1' is the smallest enclosing disk of c_1.*

Note that $d_1 \subset d_1'$ and $\lambda\, d_1 \subset \lambda\, d_1'$. In particular λ-clearance implies that d_2 is disjoint from $\lambda\, d_1$. This weaker property suffices for some of our proofs (e.g., Lemma 16). An easy volume argument shows that a modified greedy algorithm that maintains 3-clearance still returns a constant-factor approximate MIS. The key advantage of an independent set with 3-clearance is the following property, which is helpful for our dynamic algorithm (see Figure 4a):

▶ **Lemma 16.** *Let S be an independent set of the disks in a nonatree N_k such that each grid cell in N_k contributes at most one disk; and assume that S has 3-clearance. Then every disk that lies in a cell in N_k intersects at most one larger disk in S.*

Obstacle Cells: Decomposing a Nonatree into Ascending Paths. A cell $c \in N_k$ is an *obstacle cell* if it is associated with a disk in S_k (a *true obstacle*), or it has at least two children that each contain a disk in S_k (a *merge obstacle*). For every obstacle cell c, we define an *obstacle disk* as $o(c) = 3d'$, where d' is the smallest enclosing disk of the cell c. The obstacle cells decompose the nonatree into ascending paths in which each cell has relevant descendants in only a single subtree (see Figure 5a). Inside an ascending path, disks either intersect the obstacle disk of the (closest) obstacle cell below them, or are part of S_k and therefore define a true obstacle cell (see Figures 5b and 5c). We show a useful property of the obstacle disks, that allows us to consider ascending paths independently.

▶ **Lemma 17.** *When a disk d in cell $c \in N_k$ is added to S_k, it can intersect only the disk $d_o \in S_k$ associated with the next obstacle cell c_o in the ascending path $P(d)$ from c towards the root, if d_o even exists.*

Barrier Disks. For a set of disks $\mathcal{D}$, we will maintain an independent set $S \subset \mathcal{D}$, and a set $B \subset \mathcal{D}$ of *barrier disks*. When a disk d associated with a cell $c \in N_k$ is inserted or deleted from $\mathcal{D}$, we re-run the greedy process on the nonatree locally, between the cells $c_1 \preceq c \prec c_2$ that contain disks $s_1, s_2 \in S$. If any of the new disks added to S intersects s_2, then we remove

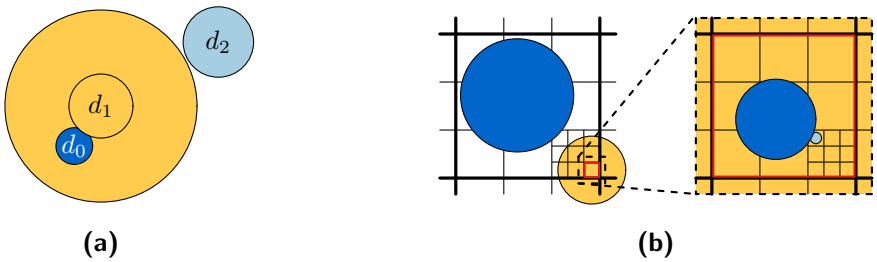

Figure 4 Constructions for Lemmata 16 and 17, respectively: **(a)** The light yellow disk representing $3d_1$ is disjoint from d_2 because of 3-clearance. **(b)** The light blue disk can intersect only a disk of S_k in the red cell c_o; larger disks in S_k are disjoint from the yellow obstacle disk defined by c_o.

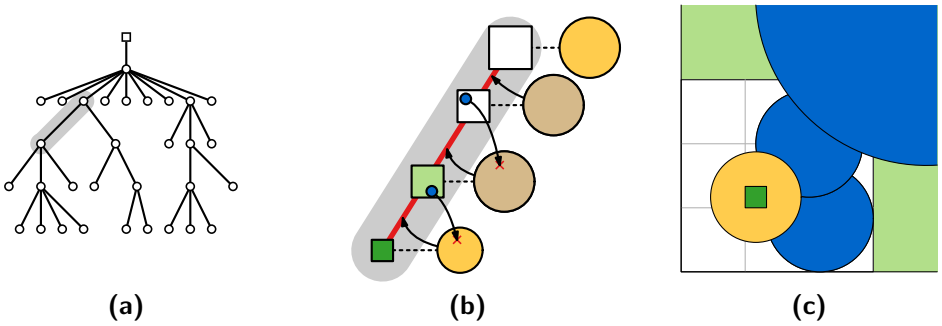

Figure 5 (a) Decomposition of a nonatree into ascending paths between merge obstacle cells. Only relevant leaves are drawn and hence all leaves are obstacle cells (disks) as well. The (square) root is not necessarily an obstacle cell. One ascending path between merge nodes is highlighted in grey. The structure of the highlighted ascending path is shown **(b)** abstractly and **(c)** geometrically: The merge obstacle cells at the top and bottom (with yellow obstacle disks) each have no disk of S_k associated with them. Every other obstacle cell on the path also defines a brown obstacle disk. Each such cell contains a (dark blue) disk of S_k, which is disjoint from the (closest) obstacle disk below it (indicated by red crosses). All (light blue) disks on the (red) ascending path above an obstacle cell are intersected by the obstacle below. Green colors identify cells between **(b)** and **(c)**.

s_2 from S, and add it to B as a *barrier disk*. Such a barrier disk defines a *barrier clearance disk* $o(c_b) = 3d_b$, where d_b is the smallest enclosing disk of the *barrier cell* c_b containing s_2. This obstacle disk also implements the clearance (defined above), to guarantee that the new disks added to S in this process do not intersect any disk in S larger than s_2. Importantly, we maintain the properties that (i) the obstacle disks, for all obstacle cells and barrier cells, form a dominating set for $\mathcal{D}$, that is, all disks in $\mathcal{D}$ intersect an obstacle disk of some obstacle cell or the barrier clearance disk of a barrier cell; and (ii) on any ascending path there is always an obstacle cell between two barrier cells.

The latter property ensures that $|B| \leq 2\,|S|$ and is maintained as follows. We maintain an assignment β between barrier disks and the closest obstacle cells below them. Each barrier disk $\beta(c_1)$ lies in one of the cells of the nonatree along an ascending path between two obstacle cells $c_1 \prec c_2$. Each path contains at most one barrier disk.

In the full paper, we introduce six invariants that guarantee that the largest of the eight independent sets, $S_1, \ldots, S_8$, is a constant-factor approximate MIS of $\mathcal{D}$. It then suffices to show that the invariants can be efficiently maintained under dynamic changes.

3.3 Dynamic Maintenance Using Farthest Neighbor Data Structures

On a high level, for a dynamic set of disks $\mathcal{D}$, we maintain eight nonatrees $N_1, \ldots N_8$, and for each $k \in \{1, \ldots, 8\}$ two sets of disks: an independent set S_k and a set of barrier disks B_k. In this section, we sketch how to maintain these data structures with polylogarithmic update times. For that, we use the dynamic farthest neighbor (DFN) data structure (Lemma 6) to efficiently find disks that are disjoint from obstacle disks in ascending paths of our nonatrees.

More specifically, when a disk d associated with a cell $c \in N_k$ is inserted or deleted, then c lies in an ascending path $P(d)$ between two obstacle cells, say $c_1 \preceq c \prec c_2$. To update the independent set S_k and the barrier disks B_k, in general we run the greedy algorithm in this path. The greedy process queries the DFN data structure to find disks that are disjoint from any smaller disk in S_k. Now we distinguish between three cases (see Figure 6): (a) If c_2 is a merge obstacle cell, then it does not contain a disk in S_k, and hence we are done. (b)

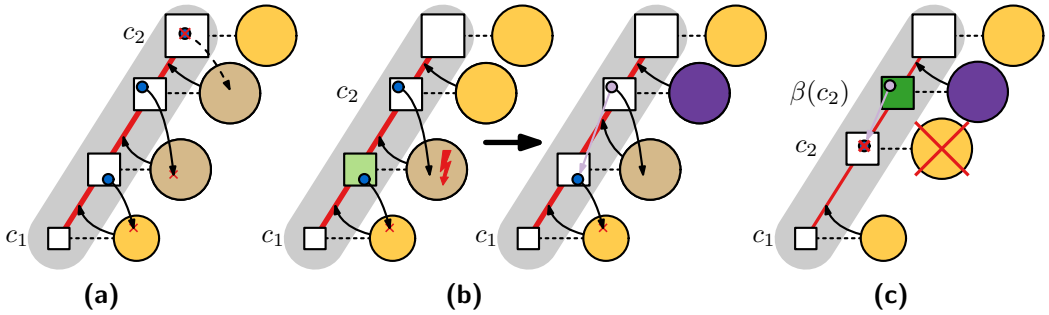

Figure 6 Greedy updates in an ascending path: **(a)** There is no disk $s_2 \in S_k$ in c_2 that can intersect the new (brown) obstacle disks in the gray ascending path. **(b)** The disk $s_2 \in S_k$ in c_2 is turned into a barrier if it overlaps the obstacle disk of the highest new disk in the light green cell. **(c)** If $\beta(c_2)$ exists, remove c_2 from S_k and run the greedy algorithm up to the dark green cell.

However, if c_2 is a true obstacle cell, then the last disk added to S_k may intersect the disk $s_2 \in S_k$ associated with c_2. If this is the case, we delete s_2 from S_k, insert it into B_k, and assign it to the highest disk in S_k in $P(d)$ below s_2; this highest disk in $P(d)$ is necessarily the disk added last to S_k, causing the intersection with s_2. (c) Finally, if s_2 was already associated with a barrier disk, $\beta(c_2)$, then adding s_2 to B_k would result in two barrier disks between consecutive obstacle cells, which is not allowed. For this reason, if $\beta(c_2)$ exists, we remove s_2 from S_k, run the greedy algorithm on a longer path, up to the cell associated with $\beta(c_2)$, and then reassign $\beta(c_2)$ to the largest disk in S_k found by the greedy algorithm.

Update Time Analysis. For maintaining our invariants, we show that it suffices to re-run the greedy algorithm on $O(1)$ ascending paths; and in each such path, the greedy algorithm terminates after $O(1)$ iterations, each of which adds one new disk to S_k that is disjoint from obstacle disks below. Thus each dynamic update involves only $O(1)$ queries to the DFN data structure (Lemma 6); in fact, we use a hierarchical version of this data structure incurring extra logarithmic factors (see the full version [12]). As these queries dominate the update time, our algorithm achieves polylogarithmic amortized expected update time.

By Lemma 5, we can smoothly transition from one independent set to another using the MIX algorithm, with amortized $O(1)$ changes in the ultimate independent set per update in $\mathcal{D}$, and conclude the following theorem.

▶ **Theorem 3.** *For a fully dynamic set of disks of arbitrary radii in the plane, an $O(1)$-approximate MIS can be maintained in polylogarithmic expected amortized update time.*

4 Conclusions

We studied the dynamic geometric independent set problem for a collection of disks in the plane and presented the first fully dynamic algorithm with polylogarithmic update time. First, we showed that for a fully dynamic set of unit disks in the plane, a constant factor approximate maximum independent set can be maintained in polylogarithmic update time. Moreover, we showed that this result generalizes to fat objects in any fixed dimension. Our main result was a dynamic data structure that maintains a constant factor approximate maximum independent set in polylogarithmic amortized update time. One bottleneck in our framework is the nearest/farthest neighbor data structure [28, 32] (as discussed in Section 1), which provides only *expected amortized* polylogarithmic update time. This is the only reason

why our algorithm does not guarantee deterministic worst-case update time, and it does not extend to balls in $\mathbb{R}^d$ for $d \geq 3$, or to arbitrary fat objects in $\mathbb{R}^2$. It remains open whether there is a dynamic nearest/farthest neighbor data structure in constant dimensions $d \geq 2$ with worst-case polylogarithmic update and query time: Any such result would immediately carry over to a fully dynamic algorithm for an approximate MIS for balls in higher dimensions.

References

1. Pankaj K Agarwal, Marc Van Kreveld, and Subhash Suri. Label placement by maximum independent set in rectangles. *Comput. Geom.*, 11(3-4):209–218, 1998. doi:10.1016/S0925-7721(98)00028-5.
2. Jochen Alber and Jirí Fiala. Geometric separation and exact solutions for the parameterized independent set problem on disk graphs. *J. Algorithms*, 52(2):134–151, 2004. doi:10.1016/j.jalgor.2003.10.001.
3. Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific, 2013. doi:10.1142/8685.
4. Reuven Bar-Yehuda, Magnús M Halldórsson, Joseph Naor, Hadas Shachnai, and Irina Shapira. Scheduling split intervals. *SIAM J. Computing*, 36(1):1–15, 2006. doi:10.1137/S0097539703437843.
5. Sarita de Berg and Frank Staals. Dynamic data structures for k-nearest neighbor queries. *Comput. Geom.*, 111:101976, 2023. doi:10.1016/j.comgeo.2022.101976.
6. Piotr Berman, Bhaskar DasGupta, S. Muthukrishnan, and Suneeta Ramaswami. Efficient approximation algorithms for tiling and packing problems with rectangles. *J. Algorithms*, 41(2):443–470, 2001. doi:10.1006/jagm.2001.1188.
7. Piotr Berman and Toshihiro Fujito. On approximation properties of the independent set problem for low degree graphs. *Theory Comput. Syst.*, 32:115–132, 1999. doi:10.1007/s002240000113.
8. Sujoy Bhore, Jean Cardinal, John Iacono, and Grigorios Koumoutsos. Dynamic geometric independent set. In *Abst. of 23rd Thailand-Japan Conference on Discrete and Computational Geometry, Graphs, and Games (TJCDCG'21)*, 2021. doi:10.48550/ARXIV.2007.08643.
9. Sujoy Bhore and Timothy M. Chan. Fully dynamic geometric vertex cover and matching. *CoRR*, abs/2402.07441, 2024. doi:10.48550/ARXIV.2402.07441.
10. Sujoy Bhore, Fabian Klute, and Jelle J. Oostveen. On streaming algorithms for geometric independent set and clique. In *Proc. 20th Workshop on Approximation and Online Algorithms (WAOA'22)*, volume 13538 of *LNCS*, pages 211–224. Springer, 2022. doi:10.1007/978-3-031-18367-6_11.
11. Sujoy Bhore, Guangping Li, and Martin Nöllenburg. An algorithmic study of fully dynamic independent sets for map labeling. *ACM J. Exp. Algorithmics*, 27(1):1–36, 2022. doi:10.1145/3514240.
12. Sujoy Bhore, Martin Nöllenburg, Csaba D. Tóth, and Jules Wulms. Fully dynamic maximum independent sets of disks in polylogarithmic update time. *CoRR*, abs/2308.00979, 2023. doi:10.48550/ARXIV.2308.00979.
13. Jean Cardinal, John Iacono, and Grigorios Koumoutsos. Worst-case efficient dynamic geometric independent set. In *Proc. 29th European Symposium on Algorithms (ESA'21)*, volume 204 of *LIPIcs*, pages 25:1–25:15, 2021. See also arXiv:2108.08050. doi:10.4230/LIPIcs.ESA.2021.25.
14. Timothy M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, 46(2):178–189, 2003. doi:10.1016/S0196-6774(02)00294-8.
15. Timothy M. Chan. A dynamic data structure for 3-D convex hulls and 2-D nearest neighbor queries. *J. ACM*, 57(3):16:1–16:15, 2010. doi:10.1145/1706591.1706596.
16. Timothy M. Chan. Dynamic geometric data structures via shallow cuttings. *Discret. Comput. Geom.*, 64(4):1235–1252, 2020. doi:10.1007/s00454-020-00229-5.

17 Timothy M. Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discret. Comput. Geom.*, 48(2):373–392, 2012. doi:10.1007/s00454-012-9417-5.

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

19 Spencer Compton, Slobodan Mitrovic, and Ronitt Rubinfeld. New partitioning techniques and faster algorithms for approximate interval scheduling. In *Proc. 50th International Colloquium on Automata, Languages, and Programming (ICALP'23)*, volume 261 of *LIPIcs*, pages 45:1–45:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.45.

20 Alon Efrat, Matthew J. Katz, Frank Nielsen, and Micha Sharir. Dynamic data structures for fat objects and their applications. *Comput. Geom.*, 15(4):215–227, 2000. doi:10.1016/S0925-7721(99)00059-0.

21 Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM J. Computing*, 34(6):1302–1323, 2005. doi:10.1137/S0097539702402676.

22 Waldo Gálvez, Arindam Khan, Mathieu Mari, Tobias Mömke, Madhusudhan Reddy Pittu, and Andreas Wiese. A 3-approximation algorithm for maximum independent set of rectangles. In *Proc. 33rd Symposium on Discrete Algorithms (SODA'22)*, pages 894–905, 2022. doi:10.1137/1.9781611977073.38.

23 Alexander Gavruskin, Bakhadyr Khoussainov, Mikhail Kokho, and Jiamou Liu. Dynamic algorithms for monotonic interval scheduling problem. *Theor. Comput. Sci.*, 562:227–242, 2015. doi:10.1016/j.tcs.2014.09.046.

24 Sariel Har-Peled. *Geometric Approximation Algorithms*, volume 173 of *Mathematical Surveys and Monographs*. AMS, 2011. URL: https://bookstore.ams.org/surv-173/.

25 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Dynamic approximate maximum independent set of intervals, hypercubes and hyperrectangles. In *Proc. 36th Symposium on Computational Geometry (SoCG'20)*, volume 164 of *LIPIcs*, pages 51:1–51:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SoCG.2020.51.

26 Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, 32(1):130–136, 1985. doi:10.1145/2455.214106.

27 Harry B. Hunt III, Madhav V. Marathe, Venkatesh Radhakrishnan, S. S. Ravi, Daniel J. Rosenkrantz, and Richard Edwin Stearns. NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. *J. Algorithms*, 26(2):238–274, 1998. doi:10.1006/jagm.1997.0903.

28 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. *Discret. Comput. Geom.*, 64(3):838–904, 2020. doi:10.1007/s00454-020-00243-7.

29 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer, Boston, MA, 1972. doi:10.1007/978-1-4684-2001-2_9.

30 Sanjeev Khanna, Shan Muthukrishnan, and Mike Paterson. On approximating rectangle tiling and packing. In *Proc. 9th Symposium on Discrete Algorithms (SODA'98)*, pages 384–393, 1998. doi:10.5555/314613.314768.

31 Kerstin Kirchner and Gerhard Wengerodt. Die dichteste Packung von 36 Kreisen in einem Quadrat. *Beiträge zur Algebra und Geometrie / Contributions to Algebra and Geometry*, 25:147–160, 1987.

32 Chih-Hung Liu. Nearly optimal planar k nearest neighbors queries under general distance functions. *SIAM J. Comput.*, 51(3):723–765, 2022. doi:10.1137/20m1388371.

595 **33** Madhav V. Marathe, Heinz Breu, Harry B. Hunt III, Sekharipuram S. Ravi, and Daniel J.
596 Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25(2):59–68, 1995. doi:
597 10.1002/net.3230250205.
598 **34** Dániel Marx. Efficient approximation schemes for geometric problems? In *Proc. 13th European*
599 *Symposium on Algorithms (ESA'05)*, volume 3669 of *LNCS*, pages 448–459. Springer, 2005.
600 doi:10.1007/11561071_41.
601 **35** Dániel Marx. On the optimality of planar and geometric approximation schemes. In *Proc.*
602 *48th Symposium on Foundations of Computer Science (FOCS'07)*, pages 338–348, 2007.
603 doi:10.1109/FOCS.2007.26.
604 **36** Joseph S.B. Mitchell. Approximating maximum independent set for rectangles in the plane.
605 In *Proc. 62nd Symposium on Foundations of Computer Science (FOCS'21)*, pages 339–350,
606 2022. doi:10.1109/FOCS52979.2021.00042.
607 **37** Micha Sharir and Pankaj K. Agarwal. *Davenport-Schinzel Sequences and their Geometric*
608 *Applications*. Cambridge University Press, 1995.
609 **38** Péter Gábor Szabó and Eckard Specht. Packing up to 200 equal circles in a square. In
610 *Models and Algorithms for Global Optimization: Essays Dedicated to Antanas Žilinskas on*
611 *the Occasion of His 60th Birthday*, pages 141–156. Springer, Boston, 2007. doi:10.1007/
612 978-0-387-36721-7_9.
613 **39** David Zuckerman. Linear degree extractors and the inapproximability of max clique and
614 chromatic number. *Theory Comput.*, 3(1):103–128, 2007. doi:10.4086/toc.2007.v003a006.