Fully Dynamic Maximum Independent Sets of Disks in Polylogarithmic Update Time

Sujoy Bhore[1], Martin Nöllenburg[2], Csaba D. Tóth[3], and Jules Wulms[4]

1 **Indian Institute of Technology Bombay**
`sujoy@cse.iitb.ac.in`

2 **Algorithms and Complexity Group, TU Wien**
`noellenburg@ac.tuwien.ac.at`

3 **California State University Northridge; Tufts University**
`csaba.toth@csun.edu`

4 **TU Eindhoven**
`j.j.h.m.wulms@tue.nl`

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.

Related Version When details are missing, refer to the full version at `arxiv.org/abs/2308.00979`.

1 Introduction

The maximum independent set (MIS) problem is a fundamental problem in theoretical computer science, and it is one of Karp's 21 classical NP-complete problems [18]. In the MIS problem, we are given a graph $G = (V, E)$, and the objective is to choose a subset $S \subseteq V$ 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 an approximation factor $n^{1-\varepsilon}$ (for $|V| = n$ and a constant $\varepsilon > 0$) unless P=ZPP [23]. In fact, even if the maximum degree of G is bounded by 3, no polynomial-time approximation scheme (PTAS) is possible [4].

Geometric Independent Set. In geometric settings, the input 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: Each vertex in G corresponds to an object in $\mathcal{L}$, and each edge connects the vertices of two intersecting objects. Thus a MIS of G corresponds to a maximum cardinality subset $\mathcal{L}' \subseteq \mathcal{L}$ of pairwise disjoint objects. A large body of work has been devoted to geometric MIS problems, due to their wide applicability, for example in scheduling [2], VLSI design [16], map labeling [1], and data mining [19, 3].

Objects	Approx. Ratio	Update time	Reference
Intervals	$1+\varepsilon$	$O(\varepsilon^{-1}\log n)$	[12]
Squares	$O(1)$	$O(\log^5 n)$ amortized	[5]
Arbitrary radii disks	$O(1)$	$(\log n)^{O(1)}$ expec. amortized	Theorem 1.2
Unit disks	$O(1)$	$O(\log n)$ worst-case	Theorem 1.1
	$1+\varepsilon$	$n^{(1/\varepsilon)^{\Omega(1)}}$	Full version [8]
f-fat objects in $\mathbb{R}^d$	$O_f(1)$	$O_f(\log n)$ worst-case	Full version [8]
d-dimensional hypercubes	$(1+\varepsilon)\cdot 2^d$	$O_{d,\varepsilon}(\log^{2d+1} n \cdot \log^{2d+1} U)$	[15]

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

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 this the *Dynamic Geometric Maximum Independent Set* problem (for short, DGMIS).

Henzinger et al. [15] 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 [22] 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 (therefore their aspect ratio is also bounded by N). Bhore et al. [5] 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. [12] achieved a faster update time for intervals, by using a new partitioning scheme. Recently, Bhore et al. [6] studied the MIS problem for intervals in the streaming settings, and obtained lower bounds.

Moreover, Bhore et al. [7] 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. [9] designed dynamic algorithms for fat objects in fixed dimension d with sublinear worst-case update time. 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 1.1–1.2); see Table 1. As a first step, we address the case of unit disks in the plane.

▶ **Theorem 1.1.** *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 1.1 in the full version [8]. Similarly to classical approximation algorithms for the static problem [16], 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

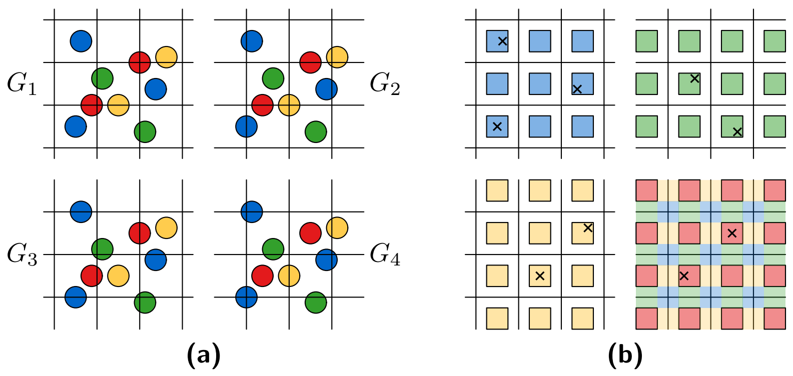

Figure 1 (a) The four shifted grids $G_1, \ldots, G_4$, which respectively do not intersect the blue, green, yellow, and red disks. **(b)** The radius-1 squares inside grid cells, along with the center points of the disks that lie completely inside grid cells, as crosses. In the bottom right, besides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

independent sets $S_1, \ldots, S_4$ at all times, where the largest is a constant-factor approximation of the MIS. Using the MIX algorithm [9], 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 [8]).

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

▶ **Theorem 1.2.** *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 1.2 in Section 2, we extend the core ideas developed for unit disks.

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 [22], resembling the same result for hypercubes by Henzinger et al. [15].

2 Disks of Arbitrary Radii in the Plane

Summary of our data structures and update algorithms. When considering 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 used in the unit disk case, where the grid cells now have side length 3^i, are shifted by $\frac{3^i}{2}$, and store disks with radius r, where $\frac{3^{i-1}}{4} < r \leq \frac{3^i}{4}$, for $i \in \mathbb{Z}$. The resulting hierarchies of recursively 3×3 subdivided grid cells form so-called *nonatrees.*

The main algorithmic ideas in using these nonatrees revolve around a bottom-up traversal of the nonatree using a well-known greedy strategy [21, 13]. In the static case, greedily considering fat objects in ascending order of size allows us to find a constant-factor approximate MIS. In the dynamic case, we want to mimic this idea by traversing paths in a nonatree towards the root. However, the height of such a nonatree (even compressed) may be $\Theta(n)$ for n disks (see Figure 2). Thus, for dynamic updates we cannot afford to traverse ascending paths in their entirety with polylogarithmic update time.

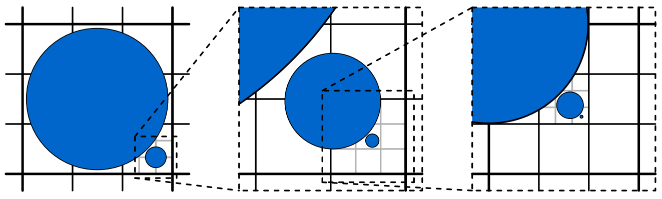

Figure 2 A nonatree with height linear in the number of stored disks, whose radii decay exponentially. A compressed nonatree (with compressed nodes) has linear height.

Approximating a Maximum Independent Set. Regardless of the above observation, we intend to traverse the nonatrees in bottom-up fashion, computing an $O(1)$-approximate MIS. In the dynamic setting, we then ensure that we only have to update the nonatrees locally.

We start by explaining how we compute an $O(1)$ approximation with our nonatrees. We refer to the data structures G_k^i associated with each value $i \in \mathbb{Z}$ as a *bucket*, and 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. Furthermore, to prevent computational overhead, our nonatrees are *compressed*, similar to compressed quadtrees [14, Chapter 2]. Figure 3 illustrates the concepts in the previous and upcoming paragraph.

Two crucial high-level steps can be distinguished in our approach:

1. For each cell $c \in G_k^i$ in our compressed nonatrees, we communicate upwards which disks have been included in our independent set. To do so, we use *obstacle disks*, and only input disks disjoint from obstacle disks can be chosen in the independent set. Once all cells are handled, we output the largest independent sets computed for the nonatrees $N_1, \ldots, N_4$, to get an $O(1)$-approximation (Lemmata 6-8 in the full version [8]).
2. We want an obstacle disk for a cell c to cover the disks in the independent set selected in the subtree rooted at c, to prevent overlaps. The obstacle disks for a cell c is hence defined as the smallest enclosing disk of c. Additionally, if the independent set of the children originates from more than one child, we do not add a disk from c, even if possible. We still obtain an $O(1)$ approximation under these constraints (Lemmata 9 and 10 in [8]).

▶ **Lemma 2.1.** *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.*

Modifications to Support Dynamic Maintenance. We now highlight two changes in the above data structures, to support efficient updates, while maintaining an $O(1)$-approximation. Dynamic updates trigger bottom-up traversals through a nonatree, and these changes are aimed at keeping updates local, leading to expected amortized polylogarithmic update time.

Obstacle cells. To support dynamic updates, we use slightly enlarged obstacle disks to prevent cascading effects during updates. Obstacle disks are associated with so-called obstacle cells of the nonatree N_k. Those cells that contribute to independent set S_k, are called *true obstacles.* Cells of the nonatree with two or more children are also considered as obstacle cells and are *merge obstacles.* The obstacle cells decompose the nonatree into ascending paths in which each cell has relevant descendants in only one subtree (see Figure 4a). 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 4b and 4c).

▶ **Lemma 2.2.** *A disk d in cell $c \in N_k$ added to S_k can intersect only the disk $d_o \in S_k$ in the next obstacle cell c_o on the ascending path $P(d)$ from c towards the root, if d_o even exists.*

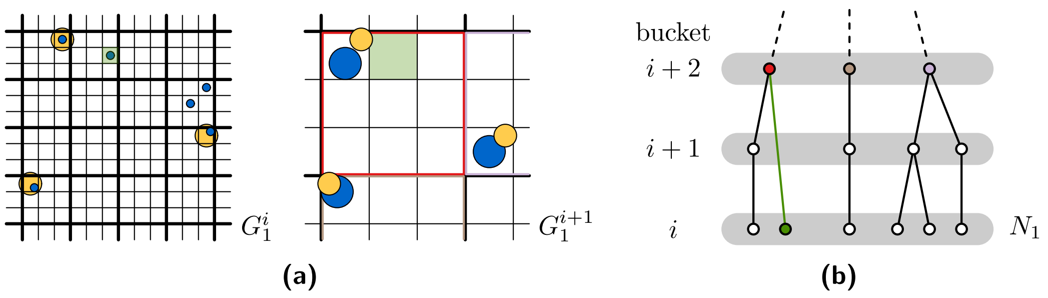

Figure 3 **(a)** Two compatible grids in buckets i and $i+1$, with (blue) disks of D in relevant cells. In particular, the green cell in G_1^i is relevant, but its (green) parent cell in G_1^{i+1} is not. Three (yellow) obstacle disks of G_1^i are drawn in both grids. Only one blue disk in G_1^{i+1} 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).

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 to repair the greedy bottom-up property. Unfortunately, we cannot afford this (recall Fig. 2). 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. Here, $\prec$ denotes that c_1 is a descendant of c_2. The greedy process guarantees that new disks added to S are disjoint from any smaller disk in S, including s_1, but they 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 $S \cup B$ is a dominating set of $\mathcal{D}$. This is one of the invariants that we maintain to show that the described changes still result in a $O(1)$-approximate MIS (Lemma 16 in [8]). Furthermore, we maintain an assignment β between barrier disks and the closest obstacle cells below them. Each barrier disk $\beta(c_1)$ lies along an ascending path between two obstacle cells $c_1 \prec c_2$. Importantly, another of our invariants states that each ascending path contains at most one barrier disk.

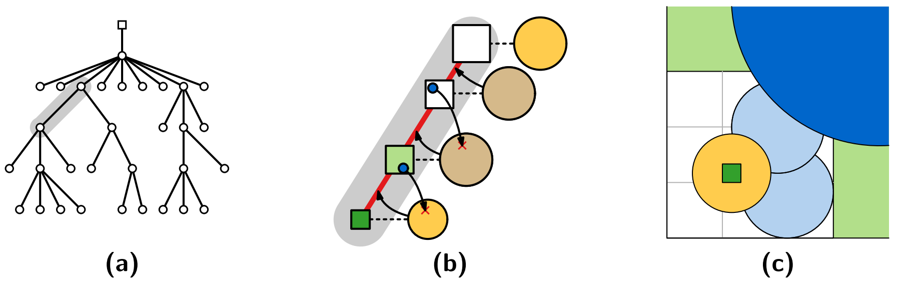

Figure 4 **(a)** Decomposition of a nonatree into ascending paths between merge obstacle cells. Only relevant leaves are drawn. One ascending path between merge nodes is highlighted in grey. This path is shown **(b)** abstractly and **(c)** geometrically: The merge obstacle cells at the top and bottom (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, disjoint from the obstacle disk below it (indicated by red crosses). All (light blue) disks on the (red) ascending path intersect the obstacle below. Green colors identify cells in **(b)** and **(c)**.

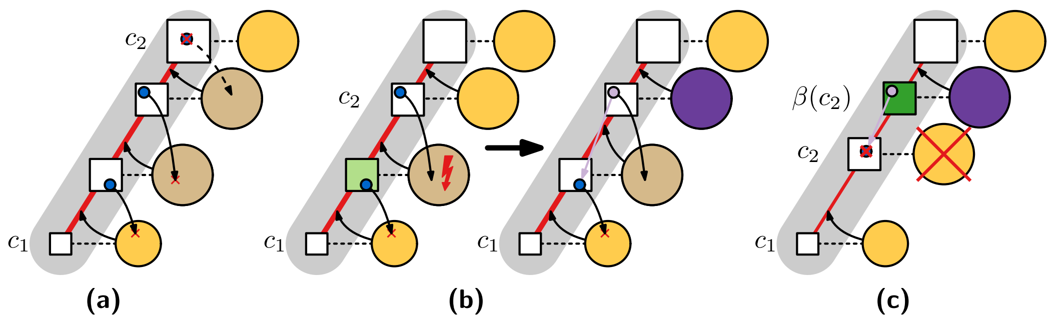

Figure 5 Greedy updates in an ascending path: **(a)** There is no disk $s_2 \in S_k$ in c_2 that can intersect the new (brown) obstacle disks in the gray ascending path. **(b)** The disk $s_2 \in S_k$ in c_2 is turned into a barrier if it overlaps the obstacle disk of the highest new disk in the light 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.

Dynamic maintenance using farthest neighbor data structures. We now sketch how to maintain our data structures with polylogarithmic update times. One key component is the use of the dynamic farthest neighbor (DFN) data structure by Kaplan et al. [17] (generalizing Chan's famous dynamic convex hull data structure [10, 11]). We adapt this data structure to efficiently find disks that are disjoint from obstacle disks in ascending paths of our nonatrees in polylogarithmic time, and with polylogarithmic expected amortized update time.

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 5): (a) If c_2 is a merge obstacle cell, then we are done. (b) 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 (and only s_2, by Lemma 2.2). If so, 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.(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 up to the cell associated with $\beta(c_2)$, and then reassign $\beta(c_2)$ to the highest disk added to S_k.

3 Conclusions

One bottleneck in our framework is the nearest/farthest neighbor data structure [17, 20], 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 a worst-case polylogarithmic update and query time: Such a 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. *Computational Geometry*, 11(3-4):209–218, 1998. `doi:10.1016/S0925-7721(98)00028-5`.

2 Reuven Bar-Yehuda, Magnús M Halldórsson, Joseph Naor, Hadas Shachnai, and Irina Shapira. Scheduling split intervals. *SIAM Journal on Computing*, 36(1):1–15, 2006. doi:10.1137/S0097539703437843.

3 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.

4 Piotr Berman and Toshihiro Fujito. On approximation properties of the independent set problem for low degree graphs. *Theory of Computing Systems*, 32:115–132, 1999. doi:10.1007/s002240000113.

5 Sujoy Bhore, Jean Cardinal, John Iacono, and Grigorios Koumoutsos. Dynamic geometric independent set. In *Abstracts of 23rd Thailand-Japan Conference on Discrete and Computational Geometry, Graphs, and Games (TJDCG)*, 2021. arXiv:2007.08643.

6 Sujoy Bhore, Fabian Klute, and Jelle J. Oostveen. On streaming algorithms for geometric independent set and clique. In *Proc. 20th International Workshop on Approximation and Online Algorithms (WAOA)*, volume 13538 of *LNCS*, pages 211–224. Springer, 2022. doi:10.1007/978-3-031-18367-6_11.

7 Sujoy Bhore, Guangping Li, and Martin Nöllenburg. An algorithmic study of fully dynamic independent sets for map labeling. *ACM Journal of Experimental Algorithmics (JEA)*, 27(1):1–36, 2022. doi:10.1145/3514240.

8 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. arXiv:2308.00979, doi:10.48550/ARXIV.2308.00979.

9 Jean Cardinal, John Iacono, and Grigorios Koumoutsos. Worst-case efficient dynamic geometric independent set. In *Proc. 29th European Symposium on Algorithms (ESA)*, volume 204 of *LIPIcs*, pages 25:1–25:15, 2021. See also arXiv:2108.08050. arXiv:arXiv:2108.08050.

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

11 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.

12 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)*, volume 261 of *LIPIcs*, pages 45:1–45:16. Schloss Dagstuhl, 2023. doi:10.4230/LIPIcs.ICALP.2023.45.

13 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.

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

15 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Dynamic approximate maximum independent set of intervals, hypercubes and hyperrectangles. In *Proc. 36th International Symposium on Computational Geometry (SoCG)*, volume 164 of *LIPIcs*, pages 51:1–51:14, 2020. doi:10.4230/LIPIcs.SoCG.2020.51.

16 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.

17 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.

18 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

19 Sanjeev Khanna, Shan Muthukrishnan, and Mike Paterson. On approximating rectangle tiling and packing. In *Proc. 9th ACM-SIAM Symposium on Discrete algorithms (SODA)*, volume 98, pages 384–393, 1998. URL: https://dl.acm.org/doi/10.5555/314613.314768.

20 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.

21 Madhav V. Marathe, Heinz Breu, Harry B. Hunt III, Sekharipuram S. Ravi, and Daniel J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25(2):59–68, 1995. doi:10.1002/net.3230250205.

22 Dániel Marx. On the optimality of planar and geometric approximation schemes. In *Proc. 48th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 338–348, 2007. doi:10.1109/FOCS.2007.26.

23 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory Comput.*, 3(1):103–128, 2007. doi:10.4086/toc.2007.v003a006.