arXiv:2407.20659v1 [cs.CG] 30 Jul 2024

Fast Static and Dynamic Approximation Algorithms for Geometric Optimization Problems: Piercing, Independent Set, Vertex Cover, and Matching

Sujoy Bhore* Timothy M. Chan†

Abstract

We develop simple and general techniques to obtain faster (near-linear time) static approximation algorithms, as well as efficient dynamic data structures, for four fundamental geometric optimization problems: *minimum piercing set* (MPS), *maximum independent set* (MIS), *minimum vertex cover* (MVC), and *maximum-cardinality matching* (MCM). Highlights of our results include the following:

- For n axis-aligned boxes in any constant dimension d, we give an $O(\log\log n)$-approximation algorithm for MPS that runs in $O(n^{1+\delta})$ time for an arbitrarily small constant $\delta > 0$. This significantly improves the previous $O(\log\log n)$-approximation algorithm by Agarwal, Har-Peled, Raychaudhury, and Sintos (SODA 2024), which ran in $O(n^{d/2} \operatorname{polylog} n)$ time.
- Furthermore, we show that our algorithm can be made fully dynamic with $O(n^{\delta})$ amortized update time. Previously, Agarwal et al. (SODA 2024) obtained dynamic results only in $\mathbb{R}^2$ and achieved only $O(\sqrt{n} \operatorname{polylog} n)$ amortized expected update time.
- For n axis-aligned rectangles in $\mathbb{R}^2$, we give an $O(1)$-approximation algorithm for MIS that runs in $O(n^{1+\delta})$ time. Our result significantly improves the running time of the celebrated algorithm by Mitchell (FOCS 2021) (which was about $O(n^{21})$), and answers one of his open questions. Our algorithm can also be made fully dynamic with $O(n^{\delta})$ amortized update time.
- For n (unweighted or weighted) fat objects in any constant dimension, we give a dynamic $O(1)$-approximation algorithm for MIS with $O(n^{\delta})$ amortized update time. Previously, Bhore, Nöllenburg, Tóth, and Wulms (SoCG 2024) obtained efficient dynamic $O(1)$-approximation algorithms only for disks in $\mathbb{R}^2$ and only in the unweighted setting.
- For n axis-aligned rectangles in $\mathbb{R}^2$, we give a dynamic $(\frac{3}{2} + \varepsilon)$-approximation algorithm for MVC with $O(\operatorname{polylog} n)$ amortized update time for any constant $\varepsilon > 0$. Our static result improves the running time of Bar-Yehuda, Hermelin, and Rawitz (2011). For disks in $\mathbb{R}^2$ or hypercubes in any constant dimension, we give the first fully dynamic $(1 + \varepsilon)$-approximation algorithm for MVC with $O(\operatorname{polylog} n)$ amortized update time.
- For (monochromatic or bichromatic) disks in $\mathbb{R}^2$ or hypercubes in any constant dimension, we give the first fully dynamic $(1 + \varepsilon)$-approximation algorithm for MCM with $O(\operatorname{polylog} n)$ amortized update time.

*Department of Computer Science & Engineering, Indian Institute of Technology Bombay, Mumbai, India. Email: sujoy@cse.iitb.ac.in

†Department of Computer Science, University of Illinois at Urbana-Champaign. Email: tmc@illinois.edu.
Work supported in part by NSF Grant CCF-2224271.

Contents

1 Introduction

In this work, we study geometric versions of four fundamental optimization problems: *minimum piercing set* (MPS), *maximum independent set* (MIS), *minimum vertex cover* (MVC), and *maximum-cardinality matching* (MCM). The first three problems are NP-hard for most types of geometric objects, and polynomial-time approximation algorithms for these problems have been extensively studied in the computational geometry literature. Recently, researchers have started exploring techniques to improve the running time of such approximation algorithms, e.g., for geometric set cover [AP20, CHQ20, CH20], MPS [AHRS24], and MCM [HY22]. Given the advent of large-scale datasets in today's world, algorithms with *near-linear running time* are especially desirable. Furthermore, many real-world problems require efficient processing of geometric data which undergo updates. This has also prompted researchers to explore efficient *dynamic* approximation algorithms for these problems [HNW20, ACS+22, CH21, CHSX22, AHRS24, BNTW24]. (Note that the existence of efficient static algorithms with subquadratic running time is a prerequisite to the existence of efficient dynamic algorithms with sublinear update time.) In this paper, we continue investigating these two research directions.

Minimum piercing set (MPS). Given a set S of n geometric objects in $\mathbb{R}^d$, a subset $P \subset \mathbb{R}^d$ is a *piercing set* of S if every object of S contains at least one point of P. The *minimum piercing set* (MPS) problem asks for a piercing set P of the smallest size. The problem has numerous applications, in facility location, wireless sensor networks, etc. [SW96, HRS02, BKS00, KNS03]. The problem may be viewed as a "continuous" version of geometric hitting set (where P is constrained to be a subset of a given discrete point set rather than $\mathbb{R}^d$), and geometric hitting set in turn corresponds to geometric set cover in the dual range space. In particular, by the standard greedy algorithm for set cover, one can compute an $O(\log n)$-approximation to the minimum piercing set in polynomial time for any family of piercing set with constant description complexity (since it suffices to work with a discrete set of $O(n^d)$ candidate points).

For unit squares/hypercubes, unit disks/balls, or more generally, near-equal-sized *fat* objects in $\mathbb{R}^d$ for any constant d, there are simple $O(n)$-time $O(1)$-approximation algorithms, whereas the well-known shifted grid strategy of Hochbaum and Maass [HM85] gives a PTAS, computing a $(1+\varepsilon)$-approximation in $n^{O(1/\varepsilon^d)}$ time. For fat objects of arbitrary sizes, a simple greedy algorithm yields an $O(1)$-approximation; the running time is naively quadratic, but can be improved to $O(n \operatorname{polylog} n)$ in the case of fat objects in $\mathbb{R}^2$ by using range-searching data structures [EKNS00]. Chan [Cha03] gave a separator-based PTAS for arbitrary fat objects (in particular, arbitrary hypercubes and balls), running in $n^{O(1/\varepsilon^d)}$ time (see also [CM05] for another PTAS for the case of unit-height rectangles in $\mathbb{R}^2$).

For arbitrary boxes[1] in $\mathbb{R}^d$ (which may not be fat), the current best polynomial-time approximation algorithm achieves approximation ratio[2] $O(\log\log \mathrm{OPT})$. The approach is to solve the standard linear program (LP) relaxation for piercing/hitting set (either exactly, or approximately by a multiplicative weight update method [BG95]), and then round the LP solution via *ε-nets*—the $\log\log$ approximation ratio comes from combinatorial bounds by Aronov, Ezra, and Sharir [AES10] on ε-nets for $d \in \{2, 3\}$ and by Ezra [Ezr10] on weak ε-nets for $d \geq 4$.

All these methods have high polynomial running time, prompting the following questions:

[1]Throughout this paper, all rectangles, boxes, squares, and hypercubes are axis-aligned by default.
[2]Throughout this paper, OPT denotes the optimal value to an optimization problem.

Do there exist near-linear-time sublogarithmic-approximation algorithms for MPS *for various families of geometric objects? Do there exist similar dynamic algorithms with sublinear update time?*

In SODA 2024, Agarwal, Har-Peled, Raychaudhury, and Sintos [AHRS24] presented a faster randomized $O(\log\log \mathrm{OPT})$-approximation algorithm with expected running time[3] $\widetilde{O}(n^{d/2})$ for boxes in $\mathbb{R}^d$. Moreover, they showed the expected running time can be improved to near-linear but only when OPT is smaller than $n^{1/(d-1)}$. Furthermore, they studied the problem in the dynamic setting. For rectangles in $\mathbb{R}^2$ they obtained $\widetilde{O}(\sqrt{n})$ amortized expected time per insertion/deletion. There has been no other prior work on dynamic MPS (ignoring the easy case of near-equal-sized fat objects, where a straightforward grid strategy yields $O(1)$-approximation in $O(1)$ update time).

Our contributions to MPS.

- For boxes in $\mathbb{R}^d$ for any constant d, we present an $O(\log\log \mathrm{OPT})$-approximation algorithm running in $O(n^{1+\delta})$ time (Theorem 2.2) for an arbitrarily small constant $\delta > 0$. The running time is a dramatic improvement over Agarwal et al.'s previous $\widetilde{O}(n^{d/2})$ bound [AHRS24] for all $d \geq 3$.
- Furthermore, our $O(\log\log \mathrm{OPT})$-approximation algorithm for boxes can be made dynamic with $O(n^\delta)$ amortized update time (Theorem 2.2) for any $d \geq 2$. This is a significant improvement over Agarwal et al.'s previous $\widetilde{O}(\sqrt{n})$ bound, which addressed only the $\mathbb{R}^2$ case.
- For fat objects in $\mathbb{R}^d$ for any constant d (assuming constant description complexity), we present an $O(1)$-approximation algorithm running in $O(n^{1+\delta})$ time (Theorem 2.11). Recall that a near-linear-time implementation of the $O(1)$-approximation greedy algorithm [EKNS00] was known only in $\mathbb{R}^2$; the exponent in the previous time bound converges to 2 as d increases, due to the use of range searching.
- Furthermore, our $O(1)$-approximation algorithm for fat objects can be made dynamic with $O(n^\delta)$ amortized update time (Theorem 2.11) for any constant d. No previous dynamic algorithms for fat objects were known even for the case of disks in $\mathbb{R}^2$ (Agarwal et al. [AHRS24] did consider the case of squares in $\mathbb{R}^2$ but obtained a weaker $\widetilde{O}(n^{1/3})$ update time bound).

Maximum independent set (MIS). Given a set S of objects in $\mathbb{R}^d$, the geometric MIS problem is to choose a maximum-cardinality subset $I \subseteq S$ of *independent* (i.e., pairwise-disjoint) objects. The problem is among the most popular geometric optimization problems studied. It is related to MPS: the size of the MIS is always at most the size of the MPS; in fact, a standard LP for MIS is dually equivalent to the standard LP for MPS.

For near-equal-sized fat objects for any constant d, Hochbaum and Maass's shifted grid method yields a PTAS [HM85]. For fat objects of arbitrary sizes, a simple greedy algorithm yields an $O(1)$-approximation [EKNS00], but several PTASs with running time $n^{O(1/\varepsilon^d)}$ or $n^{O(1/\varepsilon^{d-1})}$ have been found, e.g., via shifted quadtrees, geometric separators, or local search [EJS05, Cha03, CH12].

The case of arbitrary rectangles in $\mathbb{R}^2$ has especially garnered considerable attention. A $(\log n)$-approximation $O(n \log n)$-time algorithm via straightforward binary divide-and-conquer has long

[3]Throughout this paper, the $\widetilde{O}$ notation hides polylogarithmic factors in n.

been known [AvKS98] (see also [KMP98, Nie00]); by increasing the branching factor, the approximation ratio can be lowered to $\varepsilon \log n$ with running time $n^{O(1/\varepsilon)}$ [Cha04, BDMR01]. The first substantial progress was made by Chalermsook and Chuzhoy [CC09] in SODA 2009, who obtained an $O(\log \log n)$-approximation polynomial-time algorithm for rectangles, by rounding the LP solution using an intricate analysis. In a different direction, Adamaszek and Wiese [AHW19] in SODA 2014 obtained a quasi-PTAS, i.e., a $(1+\varepsilon)$-approximation algorithm running in $n^{\text{polylog}\, n}$ time, by using separators and dynamic programming; their approach works more generally for polygons, in particular, arbitrary line segments (see also [FP11, CPW24] for other related results). For rectangles, Chuzhoy and Ene [CE16] improved this to a "quasi-quasi-PTAS", running in $n^{\text{polyloglog}\, n}$ time, with a more complicated algorithm. In a remarkable breakthrough, Mitchell [Mit22] obtained the first polynomial-time $O(1)$-approximation algorithm for rectangles, by proving a variant of a combinatorial conjecture of Pach and Tardos [PT00] and applying straightforward dynamic programming; the approximation ratio was 10 in the original paper, but was subsequently lowered to $2+\varepsilon$ by Gálvez, Khan, Mari, Mömke, Pittu, and Wiese [GKM+22, GKM+21].

The running time of Mitchell's algorithm is high: the original paper stated a (loose) upper bound of $O(n^{21})$, and although the exponent is likely improvable somewhat with more effort, it is not clear how to get a more practical polynomial bound. For example, even if the original version of Pach and Tardos's conjecture were proven, the dynamic program would still require at least n^4 table entries. (The running time of Gálvez et al.'s algorithm is even higher.) At the end of his paper, Mitchell [Mit22] specifically asked the following question:

"Can the running time of a constant-factor approximation algorithm be improved significantly?"

The dynamic version of geometric MIS was first studied by Henzinger, Neumann, and Wiese [HNW20], who gave dynamic $O(1)$-approximation algorithms for squares/hypercubes with amortized polylogarithmic update time (see also [BCIK21]). For arbitrary boxes, Henzinger et al.'s algorithm requires approximation ratio $O(\log^{d-1} n)$. Cardinal, Iacono, and Koumoutsos [CIK21] designed dynamic algorithms for fat objects with sublinear worst-case update time, using range searching data structures, but the exponent in the update bound converges to 1 as d increases (even for squares or disks in $\mathbb{R}^2$, their update bound is large, near $\widetilde{O}(n^{3/4})$). Very recently, Bhore, Nöllenburg, Tóth, and Wulms [BNTW24] showed that for disks in $\mathbb{R}^2$, an $O(1)$-approximation can be maintained in expected polylogarithmic update time. Their algorithm for disks used certain dynamic data structures for range/intersection searching (requiring generalizations of dynamic 3-dimensional convex hulls [Cha10, Cha20a, KMR+20]); even if it could be extended to balls and fat objects in higher dimensions, the exponent would also converge to 1 for larger d (since convex hulls have much larger combinatorial complexity as dimension exceeds 3).

Do there exist dynamic $O(1)$-approximation algorithms with sublinear update time for rectangles in $\mathbb{R}^2$, or with (say) $O(n^{0.1})$ update time for fat objects in $\mathbb{R}^d$ for $d \geq 3$?

Our contributions to MIS.

- For rectangles in $\mathbb{R}^2$, we present an $O(1)$-approximation algorithm running in $O(n^{1+\delta})$ time (Theorem 3.2) for an arbitrarily small constant $\delta > 0$. The running time is a dramatic improvement over Mitchell's previous algorithm [Mit22].

- Furthermore, our $O(1)$-approximation algorithms for rectangles can be made dynamic with $O(n^\delta)$ amortized update time (Theorem 3.2). In contrast, Henzinger et al.'s previous dynamic algorithm [HNW20] had $O(\log n)$ approximation ratio.

- For fat objects in $\mathbb{R}^d$ for any constant d (assuming constant description complexity), we present an $O(1)$-approximation algorithm running in $O(n^{1+\delta})$ time (Theorem 3.6).

- Furthermore, our $O(1)$-approximation algorithm for fat objects can be made dynamic with $O(n^\delta)$ amortized update time (Theorem 3.6). The update time is a significant improvement over Cardinal et al.'s [CIK21]. Surprisingly, range searching structures are not needed.

- Our approach extends to the *weighted* case (computing the maximum-weight independent set for a given set of weighted objects). For weighted rectangles in $\mathbb{R}^2$, our static and dynamic algorithms have the same running time but with $O(\log\log n)$ approximation ratio—this matches the current best approximation ratio for polynomial-time algorithms due to Chalermsook and Walczak [CW21] (a generalization of the LP-rounding approach by Chalermsook and Chuzhoy [CC09]). The previous dynamic algorithm for weighted rectangles was due to Henzinger et al. [HNW20] and had $O(\log n)$ approximation ratio. For weighted fat objects in $\mathbb{R}^d$, we obtain the same result with $O(1)$ approximation ratio. In contrast, Cardinal et al.'s and Bhore et al.'s previous dynamic algorithms [CIK21, BNTW24] inherently do not work in the weighted setting. We obtain the *first* efficient data structures for MIS for weighted disks in $\mathbb{R}^2$ and other types of weighted fat objects.

Minimum vertex cover (MVC). Given an undirected graph $G = (V, E)$, a subset of vertices $X \subseteq V$ is a *vertex cover* if each edge has at least one endpoint in X. The *minimum vertex cover* (MVC) problem asks for a vertex cover with the minimum cardinality. In the geometric version of the problem, we are given a set S of n geometric objects in $\mathbb{R}^d$ and want the MVC of the intersection graph of S. In the exact setting, MVC is equivalent to MIS, as the complement of an MIS is an MVC. However, from the approximation perspective, a sharp dichotomy exists between the two problems; for instance, MIS on general graphs cannot be approximated with ratio $n^{1-\varepsilon}$ under standard hypotheses [Zuc07], but there is a simple greedy 2-approximation algorithm for MVC: namely, just take a maximal matching[4] and output its vertices. MVC has not received as much attention as MPS and MIS in the geometry literature, but is just as natural to study in geometric settings: if we are given a set of geometric objects that are almost non-overlapping and we want to remove the fewest number of objects to eliminate all the intersections, this is precisely the MVC problem in the intersection graph.

Erlebach, Jansen, and Seidel [EJS05] gave the first PTAS for MVC for fat objects in $\mathbb{R}^d$ running in $n^{O(1/\varepsilon^d)}$ time. For rectangles in the plane, Bar-Yehuda, Hermelin, and Rawitz [BHR11] obtained a $(\frac{3}{2} + \varepsilon)$-approximation algorithm. Bar-Yehuda et al.'s work made use of Nemhauser and Trotter's standard LP-based kernelization for MVC [NTJ75], which allows us to approximate the MVC by flipping to an MIS instance. Following the same kernelization approach, Har-Peled [Har23] noted that the known quasi-PTAS for MIS for polygons [AHW19] and quasi-quasi-PTAS for MIS for rectangles [CE16] imply a quasi-PTAS for MVC for polygons and quasi-quasi-PTAS for MVC for rectangles. Recently, Lokshtanov, Panloan, Saurabh, Xue, and Zehavi [LPS+24] gave the first

[4]A *matching* in a graph $G = (V, E)$ is a subset of edges $M \subseteq E$ such that no two edges in M share a common endpoint. A matching M is *maximal* if for every edge $uv \in E \setminus M$, either u or v is matched in M.

polynomial-time algorithm for strings (which include arbitrary line segments and polygons) that achieves a constant approximation ratio strictly below 2, using the Nemhauser–Trotter kernel and a number of new ideas.

We are interested in improving the running time of static algorithms as well as the dynamic version of geometric MVC. Dynamic MVC is a well-studied problem in the dynamic graph algorithms literature, under edge updates; e.g., see [OR10, BHI18, BK19]. However, these graph results are not directly applicable to the geometric setting, since the insertion/deletion of a single object may require as many as $\Omega(n)$ edge updates in the intersection graph in the worst case. There has been no prior work on dynamic geometric MVC (ignoring the case of monochromatic, nearly-equal-sized fat objects, where it is not difficult to adapt the standard shifted grid strategy [HM85] to maintain a $(1+\varepsilon)$-approximation with $O(1)$ update time).

Do there exist near-linear-time better-than-2-approximation algorithms for MVC *for various families of geometric objects? Do there exist similar dynamic algorithms with sublinear update time?*

Our contributions to MVC.

- For rectangles in $\mathbb{R}^2$, we speed up Bar-Yehuda et al.'s $(\frac{3}{2}+\varepsilon)$-approximation polynomial-time algorithm [BHR11] to run in $O(n \operatorname{polylog} n)$ time, and at the same time obtain a dynamic $(\frac{3}{2}+\varepsilon)$-approximation algorithm with $O(\operatorname{polylog} n)$ amortized update time (Corollary 4.6).
- For disks in $\mathbb{R}^2$ and fat boxes (e.g., hypercubes) in $\mathbb{R}^d$ for any constant d, we speed up the previous PTAS [EJS05] to run in $O(n \operatorname{polylog} n)$ time (ignoring dependence on ε), and at the same time obtain a dynamic $(1+\varepsilon)$-approximation algorithm with $O(\operatorname{polylog} n)$ amortized update time (Corollaries 4.5 and 4.7). The fact that the approximation ratio is $1+\varepsilon$ is notable and interesting: none of the known dynamic algorithms has approximation ratio $1+\varepsilon$ for the other geometric optimization problems such as MPS and MIS[5], except in 1-dimensional special cases for intervals [HNW20, BCIK21, CMR23].
- Similar results hold for bichromatic disks and fat boxes, for MVC in their *bipartite* intersection graph (Corollaries 4.8 and 4.9).

Our results on MVC can be generalized to other types of fat objects in $\mathbb{R}^d$ (e.g., balls in $\mathbb{R}^3$) but with a larger time bound, dependent on range searching (with exponent converging to 1 as d increases). However, this is unavoidable for MVC (in contrast to our results on MIS): any dynamic approximation algorithm for MVC needs to recognize whether the MVC size is zero, and so must know whether the intersection graph is empty. Dynamic range emptiness (maintaining a dynamic set of input points so that we can quickly decide whether a query object contains any input point) can be reduced to this problem, by inserting all the input points, and repeatedly inserting a query object and deleting it.

Maximum-cardinality matching (MCM). Another closely related classical optimization problem on graphs is maximum-cardinality matching (MCM), where the objective is to find a *matching*

[5]This is with good reason: for MIS, it is not possible to maintain a $(1+\varepsilon)$-approximation in sublinear time for a sufficiently small ε, even in the case of unit squares in $\mathbb{R}^2$, since the static problem has a lower bound of $n^{\Omega(1/\varepsilon)}$ under ETH [Mar07].

with the largest number of edges in a given undirected graph G. The problem is related to MVC: the size of an MVC is always at least the size of an MCM; for bipartite graphs, they are well-known to be equal. In fact, the standard LP for MVC is the dual to the LP for MCM. MCM is polynomial-time solvable: the classical algorithm by Hopcroft and Karp [HK73] runs in $O(m\sqrt{n})$ time for bipartite graphs with n vertices and m edges, and Vazirani's algorithm [Vaz94] achieves the same run time for general graphs. By recent breakthrough results [CKL+22], MCM can be solved in $m^{1+o(1)}$ time for bipartite graphs. Earlier, Duan and Pettie [DP14] obtained $O(m)$-time $(1+\varepsilon)$-approximation algorithms for general graphs.

MCM on geometric intersection graphs has received some attention. Efrat, Itai, and Katz [EIK01] showed how to compute the (exact) MCM in bipartite unit disk graphs in $O(n^{3/2}\log n)$ time. Their algorithm works for other geometric objects; for example, it runs in $O(n^{3/2}\operatorname{polylog} n)$ time for bipartite intersection graphs of arbitrary disks, by using known dynamic data structures for disk intersection searching [KKK+22]. Cabello, Cheng, Cheong, and Knauer [CCCK24] improved the time bound to $O(n^{4/3+\varepsilon})$ for unit disks, and also gave further exact subquadratic-time algorithms for other types of objects, using biclique covers in combination with the recent graph results [CKL+22]. See also [BCM23] for other special-case results. Recently, Har-Peled and Yang [HY22] presented near-linear time $(1+\varepsilon)$-approximation algorithms for MCM in (bipartite or non-bipartite) intersection graphs of arbitrary disks, among other things.

Dynamic MCM is a well-studied problem in the dynamic graph algorithms literature, under edge updates; e.g., see [OR10, GP13, BS15, BS16, PS16, Sol16, BHI18, Beh23, BKSW23, ABR24]. However, these graph results are not directly applicable to the geometric setting, again since the insertion/deletion of a single object may cause many edge updates. There has been no prior work on dynamic geometric MCM (again ignoring the easier case of monochromatic, nearly-equal-sized fat objects).

Do there exist dynamic $(1+\varepsilon)$-approximation algorithm for MCM for various families of geometric objects with sublinear update time?

Our contributions to MCM.

- For disks in $\mathbb{R}^2$ and fat boxes (e.g., hypercubes) in $\mathbb{R}^d$ for any constant d, we obtain a dynamic $(1+\varepsilon)$-approximation algorithm with $O(\operatorname{polylog} n)$ amortized update time (Corollaries 5.3, 5.4, 6.4, and 6.5), in both the monochromatic and bichromatic (bipartite) cases. This can be viewed as a dynamization of Har-Peled and Yang's static algorithms [HY22].

Our techniques for MPS and MIS. A natural approach to get faster static algorithms or efficient dynamic algorithms is to take a known polynomial-time static algorithm and modify it. This was indeed the approach taken originally by Chan and He [CH21] on dynamic geometric set cover, and more recently by Agarwal et al. [AHRS24] on dynamic MPS. In these works, the previous algorithms were LP-based, and the bottleneck was in solving the LP, which was done using a multiplicative weight update (MWU) method. The idea was to apply geometric data structuring (range searching) techniques to speed up each iteration of the MWU; if OPT is small, the number of iterations is small, but if OPT is large, we can switch to a different strategy (since we can tolerate a larger additive error).

We started our research by following the above strategy but end up discovering a different, better, and *simpler* approach: namely, we directly reduce our problem to smaller instances, and just solve these subproblems by invoking the known polynomial-time static algorithm *as a black box*! This way, we do not even need to know how the previous static algorithm internally works. (This is advantageous for MIS for rectangles, for example, since Mitchell's previous algorithm was not based on LP/MWU, and it is not clear how it can be sped up with data structures.)

More precisely: for MPS/MIS for rectangles, we first use a standard divide-and-conquer (similar to [AvKS98]) to reduce the problem to less complex instances where the rectangles are stabbable by a small number of horizontal lines and by a small number of vertical lines. The divide-and-conquer causes the approximation ratio to increase, but by using a larger branching factor n^{δ}, the increase is only by a constant factor. For each such instance, we "round" the input rectangles to reduce the number of rectangles to $n^{O(\delta)}$ (more formally, we form $n^{O(\delta)}$ "classes" and map each rectangle to a "representative" element in its class); we can then solve each such subproblem in $n^{O(\delta)}$ time by the black box. The key step is to show that input rounding increases the approximation ratio by only a constant factor; this combinatorial fact has simple proofs. This idea of reducing the input size by rounding is somewhat reminiscent to the familiar notion of *coresets* [AHV05], though we have not seen coresets used in the context of geometric MPS/MIS before.

For fat objects, we proceed similarly, except that we use a divide-and-conquer based on shifted quadtrees [Cha98].

Surprisingly, this (embarrassingly) simple approach is sufficient to yield all our new results by MPS and MIS—for example, the description of our method for MPS for rectangles fits in under two pages, in contrast to the much lengthier solution by Agarwal et al. [AHRS24]. A virtue of this approach is that dynamization now becomes almost trivial.

Our techniques for MVC and MCM. For MVC, we return to the approach of speeding up MWU using geometric data structures. There have been previous works [CHQ20, CH21] on speeding up MWU for static and dynamic geometric set cover, but thus far not for geometric MVC. We show that geometric MVC is well-suited to this approach (in some ways, even more so than geometric set cover): interestingly, the right data structure for an efficient implementation of MWU turns out to involve a type of a *dynamic generalized closest pair* problem, which Eppstein [Epp95] (see also [Cha20a]) has conveniently developed a technique for. For MVC, the purpose of using MWU to solve the LP (approximately) is in computing a Nemhauser–Trotter-style kernel [NTJ75], which allows us to reduce n to $\leq 2\,\mathrm{OPT}$ (roughly), after which we can flip to a MIS instance. As another technical ingredient, we show that an approximate LP solution is sufficient for the kernelization. To solve geometric MVC in the dynamic setting, we additionally use a standard trick: *periodic rebuilding*. As mentioned, when OPT is small, the number of iterations of the MWU is small, but when OPT is large, we only need to rebuild the solution after a long stretch of $\varepsilon\,\mathrm{OPT}$ updates.

For MCM, kernels for matching seem harder to compute. Instead, in the bipartite case, we adapt an approach based on Hopcroft and Karp's classical matching algorithm [HK73], which is known to yield good approximation after a constant number of iterations. We show how to implement the approximate version of Hopcroft and Karp's algorithm using range searching data structures. Previously, Efrat, Itai, and Katz [EIK01] have already applied geometric data structures to speed up Hopcroft and Karp's exact algorithm, but their work was on the static case. The dynamic setting is trickier and requires a more delicate approach. In the general non-bipartite case, we need one more idea by Lotker, Patt-Shamir, and Pettie [LPP15] (also used in [HY22]) to reduce the non-bipartite to the bipartite case in the approximate setting; we give a reinterpretation of this technique in terms of *color-coding* [AYZ95], of independent interest.

Both our methods for geometric MVC and MCM are quite general, and work for any family of objects satisfying certain requirements (see Theorems 4.4 and 5.2 for the general framework).

2 Minimum Piercing Set (MPS)

In this section, we present our static and dynamic approximation algorithms for MPS for boxes and fat objects.

2.1 Boxes

To solve the MPS problem for boxes, we first consider a special case that can be solved by "rounding" the input boxes—this simple idea will be the key:

Lemma 2.1. *Let d be a constant. Let Γ be a set of $O(b)$ axis-aligned hyperplanes in $\mathbb{R}^d$. Let S be a set of n axis-aligned boxes in $\mathbb{R}^d$ with the property that each box in S is stabbed by at least one hyperplane in Γ orthogonal to the k-th axis for every $k \in \{1, \dots, d\}$. We can compute an $O(\log\log \mathrm{OPT})$-approximation to the minimum piercing set for S in $\widetilde{O}(n + b^{O(1)})$ time. Furthermore, we can support insertions and deletions in S (assuming the property) in $\widetilde{O}(b^{O(1)})$ time.*

Proof. The hyperplanes in Γ form a (non-uniform) grid with $O(b^d)$ grid cells. Place two boxes of S in the same *class* if they intersect the same subset of grid cells (see Figure 1(a) for a depiction of one class). There are $O(b^{2d})$ classes (as each class can be specified by $2d$ hyperplanes in Γ). Let $\hat{S}$ be a subset of S where we keep one "representative" element from each class. Then $|\hat{S}| = O(b^{2d})$. We apply the known result by Aronov, Ezra, and Sharir [AES10] for $d \in \{2, 3\}$ or Ezra [Ezr10] for $d \geq 4$ (see also [AHRS24]) to compute an $O(\log\log \mathrm{OPT})$-approximation to the minimum piercing set for the boxes in $\hat{S}$. This takes time polynomial in $|\hat{S}|$, i.e., $b^{O(1)}$ time. Let P be the returned piercing set for $\hat{S}$. For each point $p \in P$, add the 2^d corners of the grid cell containing p to a set P'. Then $|P'| \leq 2^d |P| \leq O(\log\log \mathrm{OPT}) \cdot \mathrm{OPT}$. We output P'.

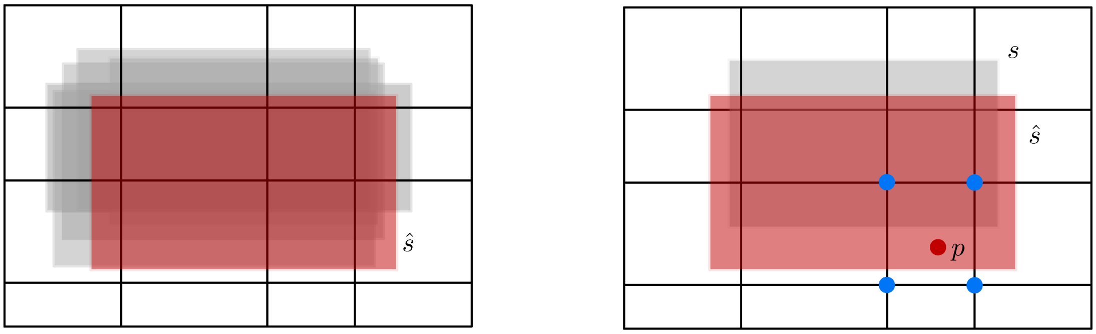
To show correctness, it suffices to show that P' is a piercing set for S. This follows because if $\hat{s}$ is the representative element of s's class, and $\hat{s}$ is pierced by p, then s intersects the grid cell containing p and so s must be pierced by one of the corners of the grid cell (because of the stated property), as illustrated in Figure 1(b).

Insertions and deletions are straightforward, by just maintaining a linked list per class, and re-running Agarwal et al. [AHRS24]'s algorithm on $\hat{S}$ from scratch each time. □

By combining the lemma with (a b-ary version of) a standard divide-and-conquer method [AvKS98], we obtain our main result for boxes:

Theorem 2.2. *Let d be a constant, and $\delta > 0$ be a parameter. Given a set S of n axis-aligned boxes in $\mathbb{R}^d$, we can compute an $O((1/\delta^d) \log\log \mathrm{OPT})$-approximation to the minimum piercing set for S in $O(n^{1+O(\delta)})$ time. Furthermore, we can support insertions and deletions in S in $O(n^{O(\delta)})$ amortized time.*

Proof. In a *type-j* problem ($j \in \{0, \dots, d\}$), we assume that the given set S is stabbable by $O(b)$ hyperplanes orthogonal to the k-th axis for every $k \in \{1, \dots, j\}$. The original problem is a type-0 problem. A type-d problem can be solved directly by Lemma 2.1.



(a) (b)

Figure 1: Proof of Lemma 2.1.

To solve a type-j problem with $j < d$, we build a b-ary tree[6] for S as follows. Pick $b-1$ hyperplanes orthogonal to the $(j+1)$-th axis to divide space into b slabs, each containing $O(n/b)$ corner points. Store the boxes in S that are stabbed by the $b-1$ dividing hyperplanes at the root. Recursively build subtrees for the subset of the $O(n/b)$ boxes in S that are contained in each of the b slabs.

For each of the $O(\log_b n)$ levels of the tree, we compute a piercing set for the subset of all boxes stored at that level; we then return the union of these $O(\log_b n)$ piercing sets. The approximation ratio consequently increases by an $O(\log_b n)$ factor. To compute a piercing set at one level, since the boxes at different nodes lie in disjoint slabs, we can just compute a piercing set for the boxes at each node separately and return the union. Computing a piercing set at each node reduces to a type-$(j+1)$ problem.

The approximation ratio for a type-j problem satisfies the recurrence $f_j \le O(\log_b n) \cdot f_{j+1}$, with $f_d = O(\log\log \mathrm{OPT})$. Thus, the overall approximation ratio is $f_0 = O((\log_b n)^d \log\log \mathrm{OPT})$.

To analyze the running time, observe that in a type-j problem, each box is stored in one node of the tree and is thus assigned to one type-$(j+1)$ problem. Since the running time for a type-d problem is $\widetilde{O}(b^{O(1)})$ per box by Lemma 2.1, the total running time is $\widetilde{O}(b^{O(1)}n)$.

When we insert/delete a box in a type-j problem, we insert/delete the box in one type-$(j+1)$ problem. The update time for a type-d problem is $\widetilde{O}(b^{O(1)})$ by Lemma 2.1. The update time for the original type-0 problem is thus $\widetilde{O}(b^{O(1)})$. One technical issue is tree balancing: for each a subtree with n boxes, each child's slab should have $O(n/b)$ corner points. We can use a standard weight-balancing scheme, rebuilding the subtree after it encounters n/b updates. The amortized cost for rebuilding is still $\widetilde{O}(\frac{b^{O(1)}n}{n/b}) = \widetilde{O}(b^{O(1)})$ per level.

Finally, we set $b = n^\delta$ to get the bounds in the theorem. □

Remark 2.3. In Lemma 2.1, we know that $\mathrm{OPT} \le O(b^d)$ under the stated property, and an alternative proof of the lemma is to apply the MWU-based method by Agarwal et al. [AHRS24], which is known to be efficient in the small OPT case. However, our input rounding approach is more general and powerful (not limited to LP/MWU-based algorithms), and will be essential later, for our results on fat objects and on MIS.

[6] This is basically a b-ary version of the *interval tree* [PS85, dBCvKO08] applied to the projections of the boxes along the $(j+1)$-th axis.

Remark 2.4. Any improvement to the $O(\log\log \mathrm{OPT})$ approximation ratio for polynomial-time algorithms for piercing boxes would automatically improve the approximation ratio for our static and dynamic algorithms.

For the static algorithm, the $1/\delta^d$ factor in the running time can be lowered to $\log^d(1/\delta)$ by setting b differently (as a function of the local input size n). In particular, when setting $\delta = 1/\log n$, we can obtain $O(n \operatorname{polylog} n)$ running time with approximation ratio $O(\log\log \mathrm{OPT} \cdot (\log\log n)^d)$ for boxes in $\mathbb{R}^d$. In Appendix A, we note another variant of the static algorithm with $O(n \operatorname{polylog} n)$ running time, while keeping the approximation ratio at $O(\log\log \mathrm{OPT})$; however, this variant uses Monte Carlo randomization, and works only in the setting when we want to output an approximation to the optimal value rather than an actual piercing set.

For the dynamic algorithm, we have stated amortized bounds for simplicity; worst-case bounds seem plausible by standard deamortization techniques for weight-balanced trees [Ove83].

2.2 Fat objects

We next turn to the case of fat objects. We begin with some definitions and preliminary facts. In what follows, the *diameter* of an object s, denoted $\mathrm{diam}(s)$, refers to its L_∞-diameter (i.e., the side length of its smallest enclosing axis-aligned hypercube). We use the following definition of fatness [Cha03], which is the most convenient here:

Definition 2.5. A collection $\mathcal{C}$ of objects in $\mathbb{R}^d$ is *c-fat* if the following property holds: for every hypercube B, there exist c points piercing all objects in $\mathcal{C}$ that intersect B and have diameter at least $\mathrm{diam}(B)$.

Definition 2.6. A *quadtree box* is a hypercube of the form $[\frac{i_1}{2^\ell}, \frac{i_1+1}{2^\ell}) \times \cdots \times [\frac{i_d}{2^\ell}, \frac{i_d+1}{2^\ell})$ for some integers $i_1, \ldots, i_d, \ell \in \mathbb{Z}$. (We also consider $\mathbb{R}^d$ to be a quadtree box.)

The fact below follows by applying standard tree partitioning schemes, e.g., Frederickson [Fre85],[7] to the (compressed) quadtree.

Fact 2.7. *Let d be a constant and b be a parameter. Let P be a set of n points in $\mathbb{R}^d$. In $O(n)$ time, we can partition of $\mathbb{R}^d$ into b interior-disjoint cells, which each cell is either a quadtree box or the difference of two quadtree boxes, such that each cell contains at most $O(n/b)$ points of P.*

Definition 2.8. An object s is *c_0-good* if it is contained in a quadtree box B with $\mathrm{diam}(B) \le c_0 \cdot \mathrm{diam}(s)$.

Fact 2.9 (Shifting Lemma [Cha98, Cha03])**.** *Suppose d is even. Let $v_j = (\frac{j}{d+1}, \ldots, \frac{j}{d+1}) \in \mathbb{R}^d$. For every object $s \subset [0,1)^d$, there exists $j \in \{0, \ldots, d\}$ such that $s + v_j$ is $O(d)$-good.*

We present our key lemma addressing a special case that can be solved by "rounding" the input objects:

Lemma 2.10. *Let d, c, c_0 be constants. Let Γ be a partition of $\mathbb{R}^d$ into b disjoint cells, where each cell is either a quadtree box or the difference of two quadtree boxes. Let S be a set of n c_0-good objects in $\mathbb{R}^d$ of*

[7] Any constant-degree tree with n nodes can be partitioned into b connected pieces of $O(n/b)$ nodes each, such that each non-singleton piece is adjacent to at most two other pieces [Fre85]. When applied to the quadtree, each piece corresponds to a quadtree box or the difference between two quadtree boxes.

constant description complexity from a c-fat collection $\mathcal{C}$, with the property that each object in S intersects the boundary of at least one cell of Γ. We can compute an $O(1)$-approximation to the minimum piercing set for S in $\widetilde{O}(n + b^{O(1)})$ time. Furthermore, we can support insertions and deletions in S (assuming the property) in $\widetilde{O}(b^{O(1)})$ time.

Proof. For each quadtree box B, define $\Lambda(B)$ to be a set of points piercing all c_0-good objects in $\mathcal{C}$ that intersect ∂B. We can ensure that $|\Lambda(B)| = O(1)$ by fatness, since all c_0-good objects s intersecting ∂B have diameter more than $\text{diam}(B)/c_0$, and we can cover ∂B by $O(1)$ hypercubes with diameter $\text{diam}(B)/c_0$.

For each cell γ which is the difference of an outer quadtree box B^+ with an inner quadtree box B^-, define $\Lambda(\gamma)$ to be $\Lambda(B^+) \cup \Lambda(B^-)$. Then $|\Lambda(\gamma)| = O(1)$.

It follows that $\text{OPT} \le \sum_{\gamma\in\Gamma} |\Lambda(\gamma)| \le O(b)$. In the static case, we could use the known greedy algorithm (e.g., see [Cha03, EKNS00]) to compute an $O(1)$-approximation to the minimum piercing set for the fat objects in S, which runs in time $\widetilde{O}(n \cdot \text{OPT}) = \widetilde{O}(bn)$. We propose a better approach which is dynamizable.

Place two objects of S in the same *class* if they intersect the same subset of cells in Γ. There are at most $b^{O(1)}$ classes: since the objects have constant description complexity, each object maps to a point in a constant-dimensional space; the objects intersecting a cell map to a semialgebraic set in this space; a class corresponds to a cell in the arrangement of these b semialgebraic sets; there are $b^{O(1)}$ cells in the arrangement. We can determine the class of an object in $\widetilde{O}(1)$ time by point location in the arrangement [AS00], after preprocessing in $b^{O(1)}$ time. Let $\hat{S}$ be a subset of S where we keep one "representative" element from each class. Then $|\hat{S}| \le b^{O(1)}$. We apply the known greedy algorithm to compute an $O(1)$-approximation to the minimum piercing set for the fat objects in $\hat{S}$. This takes time polynomial in $|\hat{S}|$, i.e., $b^{O(1)}$ time. Let P be the returned piercing set for $\hat{S}$. For each point $p \in P$, find the cell $\gamma \in \Gamma$ containing p and add $\Lambda(\gamma)$ to a set P'. Then $|P'| \le O(1) \cdot |P| \le O(1) \cdot \text{OPT}$. We output P'.

To show correctness, it suffices to show that P' is a piercing set for S. This follows because if $\hat{s}$ is the representative element of s's class, and $\hat{s}$ is pierced by p, then s intersects the cell $\gamma \in \Gamma$ containing p, and thus s intersect $\partial\gamma$ (because of the stated property), and so s must be pierced by one of the points in $\Lambda(\gamma)$.

Insertions and deletions are now straightforward, by just maintaining a linked list per class, and re-running Agarwal et al.'s algorithm on $\hat{S}$ from scratch each time. □

Combining with a quadtree-based divide-and-conquer, we obtain our main result for fat objects:

Theorem 2.11. *Let d and c be constants, and $\delta > 0$ be a parameter. Given a set S of n objects in $\mathbb{R}^d$ of constant description complexity from a c-fat collection $\mathcal{C}$, we can compute an $O(1/\delta)$-approximation to the minimum piercing set for S in $O(n^{1+O(\delta)})$ time. Furthermore, we can support insertions and deletions in S in $O(n^{O(\delta)})$ amortized time.*

Proof. We assume that all objects of S are in $[0,1)^d$ and are $O(d)$-good. This is without loss of generality by the shifting lemma (Fact 2.9): for each of the $d+1$ shifts v_j ($j \in \{0,\ldots,d\}$), we can solve the problem for the good objects of $S + v_j$, and return the union of the piercing sets found (after shifting back by $-v_j$). The approximation ratio increases by a factor of $d+1 = O(1)$.

We build a b-ary tree[8] for S as follows. Arbitrarily pick one "center" point from each object of

[8]This is basically a b-ary variant of Arya et al.'s *balanced box decomposition (BBD) tree* [AMN+98].

S and apply Fact 2.7 to the n center points of S, to obtain a partition Γ into b cells. Store the objects that intersect the boundaries of these cells at the root. Recursively build subtrees for the subset of the $O(n/b)$ boxes that are contained in each of the b cells.

For each of the $O(\log_b n)$ levels of the tree, we compute a piercing set for the subset of all objects stored at that level; we then return the union of these $O(\log_b n)$ piercing sets. The approximation ratio consequently increases by an $O(\log_b n)$ factor. To compute a piercing set at one level, since the objects at different nodes lie in disjoint cells, we can just compute a piercing set for the boxes at each node separately and return the union. Computing a piercing set at each node reduces to the case handled by Lemma 2.10. The overall approximation ratio is thus $O(\log_b n)$.

To analyze the running time, observe that each object is stored in one node of the tree and is thus assigned to one subproblem handled by Lemma 2.10. The total running time is $\widetilde{O}(b^{O(1)}n)$.

When we insert/delete an object, we insert/delete the object in one subproblem handled by Lemma 2.10. The update time is $\widetilde{O}(b^{O(1)})$. One technical issue is tree balancing: for each subtree with n objects, each child's cell should have $O(n/b)$ center points. We can use a standard weight-balancing scheme, rebuilding the subtree after encountering n/b updates. The amortized cost for rebuilding is still $\widetilde{O}(\frac{b^{O(1)}n}{n/b}) = \widetilde{O}(b^{O(1)})$ per level.

Finally, we set $b = n^{\delta}$ to get the bounds in the theorem. □

Observations similar to Remark 2.4 hold here as well.

3 Maximum Independent Set (MIS)

In this section, we present our static and dynamic approximation algorithms for MIS for (unweighted or weighted) rectangles and fat objects. The approach is very similar to our algorithms for MPS in the previous section. The main difference is in the justification that rounding the input objects increases the approximation ratio by at most an $O(1)$ factor: the proofs are trickier, but still short.

3.1 Rectangles

Lemma 3.1. *Let Γ be a set of $O(b)$ horizontal/vertical lines in $\mathbb{R}^2$. Let S be a set of n axis-aligned rectangles in $\mathbb{R}^2$ with the property that each rectangle in S is stabbed by at least one horizontal line and at least one vertical line in Γ. We can compute an $O(1)$-approximation to the maximum independent set for S in $\widetilde{O}(n + b^{O(1)})$ time. Furthermore, we can support insertions and deletions in S (assuming the property) in $\widetilde{O}(b^{O(1)})$ time.*

If the rectangles in S are weighted, we can do the same for an $O(\log\log b)$-approximation to the maximum-weight independent set.

Proof. Define classes as in the proof of Lemma 2.1. Let $\hat{S}$ be a subset of S where we keep one "representative" element from each class—in the weighted case, we keep the largest-weight element of the class. Then $|\hat{S}| = O(b^4)$. We apply Mitchell's result [Mit22] (or its subsequent improvement [GKM+22, GKM+21]) to compute an $O(1)$-approximation to the maximum independent set for the rectangles in $\hat{S}$ in the unweighted case, or Chalermsook and Walczak's result [CW21] to compute an $O(\log\log|\hat{S}|)$-approximation in the weighted case. This takes time polynomial in $|\hat{S}|$, i.e., $b^{O(1)}$ time. We output the returned independent set $\hat{I}$ for $\hat{S}$.

To analyze the approximation ratio, let I^* be the optimal independent set. For each rectangle s, let $\hat{s}$ denote the representative element of s's class. We claim that $\{\hat{s} : s \in I^*\}$ contains an independent set of cardinality $\Omega(1) \cdot |I^*|$ in the unweighted case, or of weight $\Omega(1)$ times the weight of I^* in the weighted case. From the claim, it would follow that the overall approximation ratio is $O(1)$ in the unweighted case or $O(\log \log b)$ in the weighted case.

To prove the claim, we first show that $\{\hat{s} : s \in I^*\}$ has maximum depth[9] $\Delta \le 4$. To see this, observe that if a point p lies inside $\hat{s}$, then s intersects the grid cell containing p, and so s contains one of the 4 corners of this cell (because of the stated property), as illustrated in Figure 1(b), but there are at most 4 rectangles $s \in I^*$ satisfying this condition for a fixed p (because of disjointness of I^*). A classical result of Asplund and Grünbaum [AG60] states that every arrangement of axis-aligned rectangles with maximum depth Δ is $O(\Delta^2)$-colorable.[10] Thus, $\{\hat{s} : s \in I^*\}$ can be $O(1)$-colored, and the largest-cardinality/weight color class, which is an independent set of $\hat{S}$, must have an $\Omega(1)$ fraction of the cardinality/weight of $\{\hat{s} : s \in I^*\}$.

Insertions and deletions are straightforward, by just maintaining a linked list per class in the unweighted case, or a priority queue (to maintain the largest-weight element) per class in the weighted case, and re-running Mitchell's or Chalermsook and Walczak's algorithm on $\hat{S}$ from scratch each time. □

Alternative Proof. We describe an interesting, alternative proof of the above claim, which does not rely on the known coloring results. Consider the grid formed by Γ. For each rectangle s, let $\xi^-(s)$ and $\xi^+(s)$ be the grid columns containing the left and right edge of s respectively, and let $\eta^-(s)$ and $\eta^+(s)$ be the grid rows containing the bottom and top edge of s respectively. Observe that there exists a subset Z of the grid columns and rows, such that $I^*_Z = \{s \in I^* : \xi^-(s) \in Z \;\wedge\; \xi^+(s) \notin Z \;\wedge\; \eta^-(s) \in Z \;\wedge\; \eta^+(s) \notin Z\}$ has at least $\frac{1}{16}$ of the cardinality/weight of I^*. This can be proved in several ways[11]; for example, a standard, simple probabilistic argument is to pick Z randomly and just note that the expected cardinality/weight of I^*_Z is equal to $\frac{1}{16}$ times that of I^*.

To finish, observe that $\{\hat{s} : s \in I^*_Z\}$ is independent: if s and s' do not intersect but $\hat{s}$ and $\hat{s'}$ intersect, then $\xi^+(s) = \xi^-(s')$ or $\xi^+(s') = \xi^-(s)$ or $\eta^+(s) = \eta^-(s')$ or $\eta^+(s') = \eta^-(s)$; but this can't happen when $s, s' \in I^*_Z$ by our definition of I^*_Z. □

Theorem 3.2. *Let $\delta > 0$ be a parameter. Given a set S of n axis-aligned rectangles in $\mathbb{R}^2$, we can compute an $O(1/\delta^2)$-approximation to the maximum independent set for S in $O(n^{1+O(\delta)})$ time. Furthermore, we can support insertions and deletions in S in $O(n^{O(\delta)})$ amortized time.*

If the rectangles in S are weighted, we can do the same for an $O((1/\delta^2) \log \log n)$-approximation to the maximum-weight independent set.

Proof. We proceed as in the proof of Theorem 2.2. A type-d problem is now solved by Lemma 3.1 with $d = 2$.

To solve a type-j problem with $j < d$, we build the same b-ary tree as in the proof of Theorem 2.2. For each of the $O(\log_b n)$ levels of the tree, we compute an independent set for the

[9]The *depth* of a point is the number of objects containing the point. For rectangles/boxes (but not necessarily other objects), the maximum depth (also called *ply*) is equal to the maximum clique size in the intersection graph.

[10]This has been improved to $O(\Delta \log \Delta)$ by Chalermsook and Walczak [CW21]. In our case, the rectangles are pseudo-disks, and the bound can be improved further to $O(\Delta)$. But all this is not too important, since $\Delta = O(1)$ in our application.

[11]This is similar to the well-known fact that in any undirected graph, the *maximum cut* contains at least half of the edges (this has multiple proofs, including the simple probabilistic one).

subset of all rectangles stored at that level; we then return the largest-cardinality/weight of these $O(\log_b n)$ independent sets. The approximation ratio consequently increases by an $O(\log_b n)$ factor. To compute an independent set at a level, since the rectangles at different nodes lie in disjoint slabs, we can just compute an independent set for the rectangles at each node separately and return the union. Computing an independent set at each node reduces to a type-$(j+1)$ problem.

The approximation ratio for a type-j problem satisfies the recurrence $f_j \le O(\log_b n) \cdot f_{j+1}$, with $f_d = O(1)$ in the unweighted case or $f_d = O(\log\log b)$ in the weighted case. Thus, the overall approximation ratio is $f_0 = O((\log_b n)^d)$ in the unweighted case or $f_0 = O((\log_b n)^d \log\log b)$ with $d = 2$.

The analysis of the running time and update time is as before. □

By standard binary divide-and-conquer [AvKS98], we can extend the result to higher-dimensional boxes, with the approximation ratio increased by one logarithmic factor per dimension:

Corollary 3.3. *Let d be a constant and $\delta > 0$ be a parameter. Given a set S of n axis-aligned boxes in $\mathbb{R}^d$, we can compute an $O((1/\delta^2)\log^{d-2} n)$-approximation to the maximum independent set for S in $O(n^{1+O(\delta)})$ time. Furthermore, we can support insertions and deletions in S in $O(n^{O(\delta)})$ amortized time.*

If the boxes in S are weighted, we can do the same for an $O((1/\delta^2)\log^{d-2} n \log\log n)$-approximation to the maximum-weight independent set.

Remark 3.4. Observations similar to Remark 2.4 hold here as well. For example, in Appendix B.2, we note a randomized variant of the static algorithm in $\mathbb{R}^2$ with $O(n \operatorname{polylog} n)$ running time and approximation ratio $O(1)$ (an absolute constant), when we only want an approximation to the optimal value but not an independent set.

Any improvement to the approximation ratio for polynomial-time algorithms for unweighted or weighted case would automatically imply analogous improvements to the approximation ratio for our static and dynamic algorithms for any constant d. (Although the first proof of Lemma 3.1 relies on a coloring result that holds only in $\mathbb{R}^2$, the alternative proof of the lemma straightforwardly extends to higher dimensions.)

3.2 Fat objects

Lemma 3.5. *Let d, c, c_0 be constants. Let Γ be a partition of $\mathbb{R}^d$ into b disjoint cells, where each cell is either a quadtree box or the difference of two quadtree boxes. Let S be a set of n c_0-good weighted objects in $\mathbb{R}^d$ of constant description complexity from a c-fat collection $\mathcal{C}$, with the property that each object in S intersects the boundary of at least one cell of Γ. We can compute an $O(1)$-approximation to the maximum-weight independent set for S in $\widetilde{O}(n + b^{O(1)})$ time. Furthermore, we can support insertions and deletions in S (assuming the property) in $\widetilde{O}(b^{O(1)})$ time.*

Proof. Define $\Lambda(\cdot)$, classes, and $\hat{S}$ as in the proof of Lemma 2.10. We apply a known algorithm (e.g., see [Cha03]) to compute an $O(1)$-approximation to the maximum-weight independent set for the fat objects in $\hat{S}$. This takes time polynomial in $|\hat{S}|$, i.e., $b^{O(1)}$ time. We output the returned independent set $\hat{I}$ for $\hat{S}$.

To analyze the approximation ratio, let I^* be the optimal independent set. For each object s, let $\hat{s}$ denote the representative element of s's class. We first show that $\{\hat{s} : s \in I^*\}$ has maximum depth $\Delta = O(1)$. To see this, observe that if a point p lies inside $\hat{s}$, then s intersects the cell $\gamma \in \Gamma$ containing p, and thus s intersects $\partial\gamma$ (because of the stated property), and so s must contain at least one of the

$O(1)$ points in $\Lambda(\gamma)$, but there are at most $O(1)$ objects $s \in I^*$ satisfying this condition (because of disjointness of I^*). The intersection graph of any collection of c-fat objects with maximum depth Δ is $(c\Delta)$-degenerate[12] and is therefore $(c\Delta + 1)$-colorable. Thus, $\{\hat{s} : s \in I^*\}$ can be $O(1)$-colored, and the largest-weight color class, which is an independent set of $\hat{S}$, must have $\Omega(1)$ fraction of the weight of $\{\hat{s} : s \in I^*\}$. It follows that the weight of $\hat{I}$ is at least $\Omega(1)$ times the weight of I^*.

Insertions and deletions are straightforward as before. □

Theorem 3.6. *Let d and c be constants, and $\delta > 0$ be a parameter. Given a set S of n weighted objects in $\mathbb{R}^d$ of constant description complexity from a c-fat collection $\mathcal{C}$, we can compute an $O(1/\delta)$-approximation to the maximum-weight independent set for S in $O(n^{1+O(\delta)})$ time. Furthermore, we can support insertions and deletions in S in $O(n^{O(\delta)})$ amortized time.*

Proof. We proceed as in the proof of Theorem 2.11. As before, we assume that all objects of S are in $[0,1)^d$ and are $O(d)$-good. This is without loss of generality by the shifting lemma (Fact 2.9): for each of the $d+1$ shifts v_j ($j \in \{0,\ldots,d\}$), we can solve the problem for the good objects of $S + v_j$, and return the largest-weight of the independent sets found. The approximation ratio increases by a factor of $d + 1 = O(1)$.

We build the same b-ary tree as in the proof of Theorem 2.11. For each of the $O(\log_b n)$ levels of the tree, we compute an independent set for the subset of all objects stored at that level; we then return the largest-weight of these $O(\log_b n)$ independent sets. The approximation ratio consequently increases by an $O(\log_b n)$ factor. To compute an independent set at a level, since the objects at different nodes lie in disjoint cells, we can just compute an independent set for the boxes at each node separately and return the union. Computing an independent set at each node reduces to the case handled by Lemma 3.5. The overall approximation ratio is thus $O(\log_b n)$.

The analysis of the running time and update time is as before. □

4 Minimum Vertex Cover (MVC)

In this section, we study efficient static and dynamic algorithms for the MVC problem for intersection graphs of geometric objects.

4.1 Approximating the LP via MWU

For a graph $G = (V, E)$, a *fractional vertex cover* is a vector $(x_v)_{v \in V}$ such that $x_u + x_v \le 1$ for all $uv \in E$ and $x_v \in [0,1]$ for all $v \in V$. Its *size* is defined as $\sum_{v \in V} x_v$. Finding a minimum-size fractional vertex cover corresponds to solving an LP, namely, the standard LP relaxation of the MVC problem.

It is known that solving this LP is equivalent to computing the MVC in a related bipartite graph, and thus can be done by known bipartite MCM algorithms—in fact, in time almost linear in the number of edges by recent breakthrough results [CKL+22]. However, there are two issues that prevent us from applying such algorithms. First of all, we are considering geometric intersection graphs, which may have $\Omega(n^2)$ number of edges; this issue could potentially be fixed by using known techniques involving *biclique covers* to sparsify the graph (maximum matching in a bipartite graph then reduces to maximum flow in a sparser 3-layer graph [FM95]). Second, for dynamic

[12] Recall that a graph is *k-degenerate* if every induced subgraph has a vertex of degree at most k. To see why the intersection graph is $(c\Delta)$-degenerate, pick the object s in the subgraph with the smallest diameter. From the definition of c-fatness, the objects intersecting s can be pierced by c points; so there can be at most $c\Delta$ objects intersecting s.

MVC, we will need efficient data structures that can solve the LP still faster, in *sublinear* time when OPT is small.

For our purposes, we only need to solve the LP approximately. Our idea is to use a different well-known technique: *multiplicative weight update* (MWU). The key lemma is stated below. The MWU algorithm and analysis here are not new (the description is short enough that we choose to include it to be self-contained), and MWU algorithms have been used before for static and dynamic geometric set cover and other geometric optimization problems (the application to vertex cover turns out to be a little simpler). However, our contribution is not in the proof of the lemma, but in the realization that MWU reduces the problem to designing a dynamic data structure (for finding min-weight edges subject to vertex-weight updates), which geometric intersection graphs happen to possess, as we will see.

Lemma 4.1. *We are given a graph $G = (V, E)$. Suppose there is a data structure $\mathcal{DS}$ for storing a vector $(w_v)_{v\in V}$ that can support the following two operations in τ time: (i) find an edge $uv \in E$ minimizing $w_u + w_v$, and (ii) update a number w_v.*

Given a data structure $\mathcal{DS}$ for the vector that currently has $w_v = 1$ for all $v \in V$, we can compute a $(1 + O(\delta))$-approximation to the minimum fractional vertex cover in $\widetilde{O}((1/\delta^2)\,\mathrm{OPT}\cdot\tau)$ time. Here, OPT *denotes the minimum vertex cover size.*

Proof. Given a number z, the following algorithm attempts to find a fractional vertex cover of size at most z (below, W denotes $\sum_{v\in V} w_v$):

let $w_v = 1$ for all $v \in V$, and $W = n$
while there exists $uv \in E$ with $w_u + w_v < W/z$ do
    let uv be such an edge
    $W \leftarrow W + \delta(w_u + w_v)$, $w_u \leftarrow (1+\delta)w_u$, $w_v \leftarrow (1+\delta)w_v$

If and when the algorithm terminates, we have $w_u + w_v \geq W/z$ for all $uv \in E$. Thus, defining $x_v := \min\{zw_v/W, 1\}$, we have $x_u + x_v \geq 1$ for all $uv \in E$, and $\sum_{v\in V} x_v \leq z$, i.e., $(x_v)_{v\in V}$ is a fractional vertex cover of size at most z.

We now bound the number of iterations t. In each iteration, W increases by at most a factor of $1 + \delta/z$. Thus, at the end,

$$W \leq (1 + \delta/z)^t n.$$

Write each w_v as $(1+\delta)^{c_v}$ for some integer c_v. Let $(x_v^*)_{v\in V}$ be an optimal fractional vertex cover of size z^*. In each iteration, $\sum_{v\in V} c_v x_v^*$ increases by at least 1 (since we increment c_u and c_v for the chosen edge uv, and we know $x_u^* + x_v^* \geq 1$). Thus, at the end, $\sum_{v\in V} c_v x_v^* \geq t$. Since $\sum_{v\in V} x_v^* = z^*$, it follows that $\max_{v\in V} c_v \geq t/z^*$. Thus,

$$W \geq (1+\delta)^{t/z^*}.$$

Therefore, $(1+\delta)^{t/z^*} \leq (1+\delta/z)^t n \leq e^{\delta t/z} n$, implying $(t/z^*)\ln(1+\delta) \leq \delta t/z + \ln n$. So, if $z \geq (1+\delta)z^*$, then $t \leq \frac{z^* \ln n}{\ln(1+\delta) - \delta/(1+\delta)} = O((1/\delta^2) z^* \log n)$.

Note that only $O(t) = O((1/\delta^2) z^* \log n)$ of the numbers w_v are not equal to 1, so the vector $(x_v)_{v\in V}$ can be encoded in $\widetilde{O}(z^*)$ space.

We can try different z values by binary or exponential search till the algorithm terminates in $O((1/\delta^2) z^* \log n)$ iterations. Each run can be implemented with $O((1/\delta^2) z^* \log n)$ operations in $\mathcal{DS}$. After each run, we reset all the modified values w_v back to 1 by $O((1/\delta^2) z^* \log n)$ update operations in $\mathcal{DS}$. □

4.2 Kernel via Approximate LP

Our approach for solving the MVC problem is to use a standard *kernel* by Nemhauser and Trotter [NTJ75], which allows us to reduce the problem to an instance where the number of vertices is at most 2 OPT. Nemhauser and Trotter's construction is obtained from the LP solution: the kernel is simply the subset of all vertices $v \in V$ with $x_v = \frac{1}{2}$.

In our scenario, we are only able to solve the LP approximately. We observe that this is still enough to give a kernel of approximately the same size. We adapt the standard analysis of Nemhauser and Trotter, but some extra ideas are needed. The proof below will be self-contained.

Lemma 4.2. *Let $c \geq 1$ and $0 \leq \delta < \gamma < \frac{1}{4}$. Given a $(1 + O(\delta))$-approximation to the minimum fractional vertex cover in a graph $G = (V, E)$, we can compute a subset $K \subseteq V$ of size at most $(2 + O(\gamma))\,\mathrm{OPT}$, in $O(\mathrm{OPT})$ time, such that a $c(1 + O(\sqrt{\delta/\gamma}))$-approximation to the minimum vertex cover of G can be obtained from a c-approximation to the minimum vertex cover of $G[K]$ (the subgraph of G induced by K). Here,* OPT *denotes the minimum vertex cover size of G.*

Proof. Let $(x_v)_{v \in V}$ be the given fractional vertex cover. Let $\lambda < \gamma$ be a parameter to be set later. Pick a value $\alpha \in [\frac{1}{2} - \gamma - \lambda, \frac{1}{2} - \lambda]$ which is an integer multiple of λ, minimizing $|\{v \in V : \alpha \leq x_v < \alpha + \lambda\}|$. This can be found in $O(|\{v \in V : x_v \geq \frac{1}{2} - \gamma - \lambda\}|) \leq O(\mathrm{OPT})$ time.[13]

Partition V into 3 subsets: $L = \{v \in V : x_v < \alpha\}$, $H = \{v \in V : x_v > 1 - \alpha\}$, and $K = \{v \in V : \alpha \leq x_v \leq 1 - \alpha\}$. Note that $|K| \leq \frac{1}{\alpha} \sum_{v \in V} x_v \leq (2 + O(\gamma))(1 + \delta)\mathrm{OPT} = (2 + O(\gamma))\mathrm{OPT}$.

Let X_K be a c-approximate minimum vertex cover of $G[K]$. We claim that $X := X_K \cup H$ is a $c(1 + O(\sqrt{\delta/\gamma}))$-approximation to the minimum vertex cover of G.

First, $X_K \cup H$ is a vertex cover of G, since vertices in L can only be adjacent to vertices in H.

Let X^* be a minimum vertex cover of G, with $|X^*| = \mathrm{OPT}$. Since $K \cap X^*$ is a vertex cover of $G[K]$, we have $|X_K| \leq c|K \cap X^*|$, and so $|X| \leq c(|K \cap X^*| + |H|) = c(|X^*| + |H \setminus X^*| - |L \cap X^*|)$.

We will upper-bound $|H \setminus X^*| - |L \cap X^*|$. To this end, let $L' = \{v \in V : \alpha \leq x_v < \alpha + \lambda\}$, and define the following modified vector $(x'_v)_{v \in V}$:

$$x'_v = \begin{cases} x_v - \lambda & \text{if } v \in H \setminus X^* \\ x_v + \lambda & \text{if } v \in (L \cup L') \cap X^* \\ x_v & \text{otherwise.} \end{cases}$$

Note that $(x'_v)_{v \in V}$ is still a fractional vertex cover, since for each edge $uv \in E$ with $u \in H \setminus X^*$ and $v \in (L \cup L') \cap X^*$, we have $x'_u + x'_v = (x_u - \lambda) + (x_v + \lambda) \geq 1$; on the other hand, for each edge $uv \in E$ with $u \in H \setminus X^*$ and $v \notin (L \cup L') \cap X^*$, we have $v \in X^*$ (since X^* is a vertex cover) and so $v \notin L \cup L'$, implying that $x'_u + x'_v \geq (1 - \alpha - \lambda) + (\alpha + \lambda) \geq 1$. Now, $\sum_{v \in V} x_v - \sum_{v \in V} x'_v = \lambda(|H \setminus X^*| - |L \cap X^*| - |L' \cap X^*|)$. On the other hand, $\sum_{v \in V} x_v - \sum_{v \in V} x'_v \leq (1 - \frac{1}{1 + O(\delta)}) \sum_{v \in V} x_v \leq O(\delta)|X^*|$. It follows that $|H \setminus X^*| - |L \cap X^*| \leq O(\frac{\delta}{\lambda})|X^*| + |L' \cap X^*|$.

By our choice of α, we have $|L'| \leq O(\frac{1}{\gamma/\lambda}) \cdot |\{v \in V : x_v \geq \frac{1}{2} - \gamma - \lambda\}| \leq O(\frac{\lambda}{\gamma})|X^*|$. We conclude that $|X| \leq c(|X^*| + |H \setminus X^*| - |L \cap X^*|) \leq c(1 + O(\frac{\delta}{\lambda} + \frac{\lambda}{\gamma}))|X^*|$. Choose $\lambda = \sqrt{\gamma\delta}$. □

[13] This assumes an appropriate encoding of the vector $(x_v)_{v \in V}$, for example, the encoding from the proof of Lemma 4.1.

4.3 Dynamic Geometric MVC via Kernels

We now use kernels to reduce the dynamic vertex cover for geometric intersection graphs to a special case of static vertex cover where the number of objects is approximately at most $2\,\mathrm{OPT}$. We use a simple, standard idea for dynamization: be lazy, and periodically recompute the solution only after every $\varepsilon\,\mathrm{OPT}$ updates (since the optimal size changes by at most 1 per update). This idea is commonly used in dynamic algorithms, e.g., in various previous works on dynamic geometric MIS [CIK21], dynamic MPS [AHRS24], dynamic matching in graphs [GP13], etc.

We observe that the data structure subproblem of dynamic min-weight intersecting pair, needed in Lemma 4.1, is reducible to dynamic intersection detection by known techniques.

Lemma 4.3. *Let $\mathcal{C}$ be a class of geometric objects, where there is a dynamic data structure $\mathcal{DS}_0$ for n objects in $\mathcal{C}$ that can detect whether there is an object intersecting a query object, and supports insertions and deletions of objects, with $O(\tau_0(n))$ query and update time.*

Then there is a dynamic data structure $\mathcal{DS}$ for n weighted objects in $\mathcal{C}$ that maintains an intersecting pair of objects minimizing the sum of the weights, under insertions and deletions of weighted objects, with $\widetilde{O}(\tau_0(n))$ amortized time.

Proof. Eppstein [Epp95] gave a general technique to reduce the problem of dynamic closest pair to the problem of dynamic nearest neighbor search, for arbitrary distance functions, while increasing the time per operation by at most two logarithmic factors (with amortization). Chan [Cha20a] gave an alternative method achieving a similar result. The $\mathcal{DS}$ problem can be viewed as a dynamic closest pair problem, where the distance between objects u and v is $w_u + w_v$ if they intersect, and ∞ otherwise. Thus, our problem reduces to the corresponding dynamic nearest neighbor search problem, namely, designing a data structure that can find a min-weight object intersecting a query object, subject to insertions and deletions of objects (*dynamic min-weight intersection searching*).

We can further reduce this to the $\mathcal{DS}_0$ problem (*dynamic intersection detection*) by a standard *multi-level* data structuring technique [AE99], where the primary data structure is a 1D search tree over the weights, and each node of the tree stores a secondary data structure for dynamic intersection detection. Query and update time increase by one more logarithmic factor. □

We are now ready to state our general framework for solving MVC problems for geometric objects: the following theorem allows us to convert any efficient static approximation algorithm for the special case when n is roughly less than $2\,\mathrm{OPT}$ to not only an efficient static approximation algorithm for the general case, but an efficient dynamic approximation algorithm at the same time, under the assumption that there is an efficient dynamic data structure for intersection detection (as noted in the introduction, some form of range searching is unavoidable for MVC).

Theorem 4.4. *Let $c \geq 1$, $\varepsilon > 0$, and $0 \leq \delta < \gamma < \frac{1}{4}$. Let $\mathcal{C}$ be a class of geometric objects with the following oracles:*

(i) a dynamic data structure $\mathcal{DS}_0$ for n objects in $\mathcal{C}$ that can detect whether there is an object intersecting a query object, and supports insertions and deletions of objects, with $O(\tau_0(n))$ query and update time;

(ii) a static algorithm $\mathcal{A}$ for computing a c-approximation of the minimum vertex cover of the intersection graph of n objects in $\mathcal{C}$ in $T(n)$ time, under the promise that $n \leq (2 + O(\gamma))\,\mathrm{OPT}$, where OPT is the optimal vertex cover size.

Then there is a dynamic data structure for n objects in $\mathcal{C}$ that maintains a $c(1 + O(\sqrt{\delta/\gamma} + \varepsilon))$-approximation of the minimum vertex cover of the intersection graph, under insertions and deletions of objects, in $\widetilde{O}((1/(\delta^2\varepsilon))\tau_0(n) + (1/\varepsilon)T(n)/n)$ amortized time, assuming that $T(n)/n$ is monotonically increasing and $T(2n) = O(T(n))$.

Proof. Assume that $b \le \mathrm{OPT} < 2b$ for a given parameter b. Divide into phases with εb updates each. At the beginning of each phase:

1. Compute a $(1 + \delta)$-approximation to the minimum fractional vertex cover by Lemma 4.1 in $\widetilde{O}((1/\delta^2)b \cdot \tau_0(n))$ amortized time using the data structure $\mathcal{DS}$ from Lemma 4.3. (A weight change can be done by a deletion and an insertion. It is important to note that we don't rebuild $\mathcal{DS}$ at the beginning of each phase; we continue using the same structure $\mathcal{DS}$ in the next phase, after resetting the modified weights back to 1.)

2. Generate a kernel K with size at most $(2 + O(\gamma))\,\mathrm{OPT}$ by Lemma 4.2 in $O(b)$ time.

3. Compute a c-approximation to the minimum vertex cover of the intersection graph of K by the static algorithm $\mathcal{A}$ in $O(T((2 + O(\gamma))\,\mathrm{OPT})) = O(T(b))$ time, from which we obtain a $c(1+O(\sqrt{\delta/\gamma}))$-approximation of the minimum vertex cover of the entire intersection graph.

Since the above is done only at the beginning of each phase, the amortized cost per update is $\widetilde{O}(\frac{(1/\delta^2)b\cdot\tau_0(n)+T(b)}{\varepsilon b})$.

During a phase, we handle an object insertion simply by inserting it to the current cover, and we handle an object deletion simply by removing it from the current cover. This incurs an additive error at most $O(\varepsilon b) = O(\varepsilon\,\mathrm{OPT})$. We also perform the insertion/deletion of the object in $\mathcal{DS}$, with initial weight 1, in $\widetilde{O}(\tau_0(n))$ amortized time.

How do we obtain a correct guess b? We build the above data structure for each b that is a power of 2, and run the algorithm simultaneously for each b (with appropriate cap on the run time based on b). □

4.4 Specific Results

We now apply our framework to solve the dynamic geometric vertex cover problem for various specific families of geometric objects. By Theorem 4.4, it suffices to provide (i) a dynamic data structure $\mathcal{DS}_0$ for intersection detection queries, and (ii) a static algorithm $\mathcal{A}$ for solving the special case of the vertex cover problem when $n \le (2 + O(\gamma))\mathrm{OPT}$.

Disks in $\mathbb{R}^2$. Intersection detection queries for disks in $\mathbb{R}^2$ reduce to additively weighted Euclidean nearest neighbor search, where the weights (different from the weights from MWU) are the radii of the disks. Kaplan et al. [KKK+22] adapted Chan's data structure [Cha10, Cha20b] for dynamic Euclidean nearest neighbor search in $\mathbb{R}^2$ (reducible to dynamic convex hull in $\mathbb{R}^3$) and obtained a data structure for additively weighted Euclidean nearest neighbor search with polylogarithmic amortized update time and query time. Thus, $\mathcal{DS}_0$ can be implemented in $\tau_0(n) = O(\log^{O(1)} n)$ amortized time for disks in $\mathbb{R}^2$.

In Appendix B.1, we give a static $(1 + O(\varepsilon))$-approximation algorithm $\mathcal{A}$ for MVC for disks, under the promise that $n = O(\mathrm{OPT})$, with running time $T(n) = \widetilde{O}(2^{O(1/\varepsilon^2)}n)$. The algorithm is obtained by modifying a known PTAS for MIS for disks [Cha03].

Applying Theorem 4.4 with $c = 1 + O(\varepsilon)$, $\gamma = \Theta(1)$, and $\delta = \varepsilon^2$, we obtain our main result for disks:

Corollary 4.5. *There is a dynamic data structure for n disks in $\mathbb{R}^2$ that maintains a $(1+O(\varepsilon))$-approximation of the minimum vertex cover of the intersection graph, under insertions and deletions, in $O(2^{O(1/\varepsilon^2)} \log^{O(1)} n)$ amortized time. (In particular, there is a static $(1+O(\varepsilon))$-approximation algorithm running in $O(2^{O(1/\varepsilon^2)} n \log^{O(1)} n)$ time.)*

Rectangles in $\mathbb{R}^2$. For rectangles in $\mathbb{R}^2$, dynamic intersection detection requires $\tau_0(n) = O(\log^{O(1)} n)$ query and update time, by standard orthogonal range searching techniques (range trees) [PS85, AE99].

(Note: There is an alternative approach that bypasses Eppstein's technique and directly solves the $\mathcal{DS}$ data structure problem: We dynamically maintain a *biclique cover*, which can be done in polylogarithmic time for rectangles [Cha06]. It is then easy to maintain the minimum weight of each biclique, by maintaining the minimum weight of each of the two sides of the biclique with priority queues.)

In Appendix B.2, we give a static $(\frac{3}{2} + O(\varepsilon))$-approximation algorithm for MVC for rectangles, with running time $T(n) = \widetilde{O}(2^{O(1/\varepsilon^2)} n)$. The algorithm is obtained by modifying the method by Bar-Yehuda, Hermelin, and Rawitz [BHR11], and combining with our efficient kernelization method.

Applying Theorem 4.4 with $c = \frac{3}{2} + O(\varepsilon)$, $\gamma = \Theta(1)$, and $\delta = \varepsilon^2$, we obtain our main result for rectangles:

Corollary 4.6. *There is a dynamic data structure for n axis-aligned rectangles in $\mathbb{R}^2$ that maintains a $(\frac{3}{2}+O(\varepsilon))$-approximation of the minimum vertex cover of the intersection graph, under insertions and deletions, in $O(2^{O(1/\varepsilon^2)} \log^{O(1)} n)$ amortized time. (In particular, there is a static $(\frac{3}{2} + O(\varepsilon))$-approximation algorithm running in $O(2^{O(1/\varepsilon^2)} n \log^{O(1)} n)$ time.)*

Fat boxes (e.g., hypercubes) in $\mathbb{R}^d$. For the case of fat axis-aligned boxes (e.g., hypercubes) in a constant dimension d, dynamic intersection detection can again be solved with $\tau_0(n) = O(\log^{O(1)} n)$ amortized query and update time by orthogonal range searching [PS85, AE99]. We can design the static algorithm $\mathcal{A}$ in exactly the same way as in the case of disks (since the method in Appendix B.1 holds for fat objects in $\mathbb{R}^d$), achieving $T(n) = \widetilde{O}(2^{O(1/\varepsilon^d)} n)$.

Corollary 4.7. *There is a dynamic data structure for n fat axis-aligned boxes in $\mathbb{R}^d$ for any constant d that maintains a $(1 + O(\varepsilon))$-approximation of the minimum vertex cover of the intersection graph, under insertions and deletions, in $O(2^{O(1/\varepsilon^d)} \log^{O(1)} n)$ amortized time.*

We can also obtain results for balls or other types of fat objects in $\mathbb{R}^d$, but because intersection detection data structures have higher complexity, the update time would be sublinear rather than polylogarithmic.

Bipartite disks in $\mathbb{R}^2$. For the case of a bipartite intersection graph between two sets of disks in $\mathbb{R}^2$, we have $\tau_0(n) = O(\log^{O(1)} n)$ as already noted. In the bipartite case, the static algorithm $\mathcal{A}$ is trivial: we just return the smaller of the two parts in the bipartition, which yields a vertex cover of size at most $n/2 \le (1 + O(\varepsilon))\,\mathrm{OPT}$ under the promise that $n \le (2 + O(\varepsilon))\,\mathrm{OPT}$.

Applying Theorem 4.4 with $c = 1 + O(\varepsilon)$, $\gamma = \varepsilon$, and $\delta = \varepsilon^3$, we obtain:

Corollary 4.8. *There is a dynamic data structure for two sets of $O(n)$ disks in $\mathbb{R}^2$ that maintains a $(1 + O(\varepsilon))$-approximation of the minimum vertex cover of the bipartite intersection graph, under insertions and deletions, in $O((1/\varepsilon^7)\log^{O(1)} n)$ amortized time.*

Bipartite boxes in $\mathbb{R}^d$. For the case of a bipartite intersection graph between two sets of (not necessarily fat) boxes in $\mathbb{R}^d$, we have $\tau_0(n) = O(\log^{O(1)} n)$ by orthogonal range searching. As already noted, in bipartite cases, the static algorithm $\mathcal{A}$ is trivial.

Corollary 4.9. *There is a dynamic data structure for two sets of $O(n)$ axis-aligned boxes in $\mathbb{R}^d$ for any constant d that maintains a $(1+O(\varepsilon))$-approximation of the minimum vertex cover of the bipartite intersection graph, under insertions and deletions, in $O((1/\varepsilon^7)\log^{O(1)} n)$ amortized time.*

We did not write out the number of logarithmic factors in our results, as we have not attempted to optimize them, but it is upper-bounded by 3 plus the number of logarithmic factors in $\tau_0(n)$.

5 Bipartite Maximum-Cardinality Matching (MCM)

The approach in the previous section finds a $(1 + O(\varepsilon))$-approximation of the MVC of bipartite geometric intersection graphs, and so it allows us to approximate the size of the MCM in such bipartite graphs. However, it does not compute a matching. In this section, we give a different approach to maintain a $(1+O(\varepsilon))$-approximation of the MCM in bipartite intersection graphs. The first thought that comes to mind is to compute a kernel, as we have done for MVC, but for MCM, known approaches seem to yield only a kernel of $O(\mathrm{OPT}^2)$ size (e.g., see [GP13]). Instead, we will bypass kernels and construct an approximate MCM directly.

5.1 Approximate Bipartite MCM via Modified Hopcroft–Karp

It is well known that Hopcroft and Karp's $O(m\sqrt{n})$-time algorithm for exact MCM in bipartite graphs [HK73] can be modified to give a $(1 + \varepsilon)$-approximation algorithm that runs in near-linear time, simply by terminating early after $O(1/\varepsilon)$ iterations (for example, see the introduction in [DP14]). We describe a way to reimplement the algorithm in sublinear time when OPT is small by using appropriate data structures, which correspond to dynamic intersection detection in the case of geometric intersection graphs. Note that earlier work by Efrat, Itai, and Katz [EIK01] has already combined Hopcroft and Karp's algorithm with geometric data structures to obtain static exact algorithm [EIK01] for maximum matching in bipartite geometric intersection graphs (see also Har-Peled and Yang's paper [HY22] on static approximation algorithms). However, to achieve bounds sensitive to OPT, our algorithm will work differently (in particular, it will be DFS-based instead of BFS-based).

Lemma 5.1. *We are given an unweighted bipartite graph $G = (V, E)$. Suppose there is a data structure $\mathcal{DS}_0$ for storing a subset $X \subseteq V$ of vertices, initially with $X = \emptyset$, that supports the following two operations in τ_0 time: given a vertex $u \in V$, find a neighbor of u that is in X (if exists), and insert/delete a vertex to/from X.*

Given a data structure $\mathcal{DS}_0$ that currently has $X = V$, and given a maximal matching M_0, we can compute a $(1 + O(\varepsilon))$-approximation to the maximum-cardinality matching in $\widetilde{O}((1/\varepsilon^2)\,\mathrm{OPT} \cdot \tau_0)$ time. Here, OPT *denotes the maximum matching size.*

Proof. The algorithm proceeds iteratively. We maintain a matching M. At the beginning of the ℓ-th iteration, we know that the current matching M does not have augmenting paths of length $\le 2\ell - 1$. We find a maximal collection Γ of vertex-disjoint augmenting paths of length $2\ell + 1$. We then augment M along the paths in Γ. As shown by Hopcroft and Karp [HK73], the new matching M will then not have augmenting paths of length $\le 2\ell + 1$.

As shown by Hopcroft and Karp [HK73], there are at least $\mathrm{OPT} - |M|$ vertex-disjoint augmenting paths, and so $|M| \ge (\ell + 1)(\mathrm{OPT} - |M|)$, i.e., $\mathrm{OPT} \le (1 + \frac{1}{\ell+1})|M|$. Thus, once ℓ reaches $\Theta(1/\varepsilon)$, we may terminate the algorithm. Initially, we can set $M = M_0$ before the first iteration.

It suffices to describe how to find a maximal collection Γ of vertex-disjoint augmenting paths of length $2\ell + 1$ in the ℓ-th iteration, under the assumption that there are no shorter augmenting paths. Hopcroft and Karp originally proposed a BFS approach, starting at the "exposed" vertices not covered by the current matching M. Unfortunately, this approach does not work in our setting: because we want the running time to be near OPT, the searches need to start at vertices of M. We end up adopting a DFS approach, but the vertices of M need to be duplicated ℓ times in ℓ "layers" (this increases the running time by a factor of ℓ, but fortunately, ℓ is small in our setting).

Let V_M be the $2|M|$ vertices of the current matching M. As is well known, $\mathrm{OPT} \le 2|M|$. A walk $v_0u_1v_1\cdots u_\ell v_\ell u_{\ell+1}$ in G is an *augmenting walk* of length $2\ell + 1$ if $v_0 \notin V_M$, $v_0u_1 \notin M$, $u_1v_1 \in M$, $\ldots$, $v_\ell u_{\ell+1} \notin M$, and $u_{\ell+1} \notin V_M$. In such an augmenting walk of length ℓ, we automatically have $v_0 \ne u_{\ell+1}$ (because G is bipartite) and the walk must automatically be a simple path (because otherwise we could short-cut and obtain an augmenting path of length $\le 2\ell - 1$). In the procedure EXTEND$(v_0u_1v_1\cdots u_i, \ell)$ below, the input is a walk $v_0u_1v_1\cdots u_i$ with $v_0u_1 \notin M$, $u_1v_1 \in M$, $\ldots$, $v_{i-1}u_i \notin M$, and the output is true if it is possible to extend it to an augmenting walk $v_0u_1v_1\cdots u_\ell v_\ell u_{\ell+1}$ of length $2\ell + 1$ that is vertex-disjoint from the augmenting walks generated so far.

MAXIMAL-AUG-PATHS(ℓ):
let $X = V \setminus V_M$ and $X_1 = \cdots = X_\ell = V_M$
for each $u_1 \in X_1$ do
    let v_0 be a neighbor of u_1 with $v_0 \in X$
    if v_0 does not exist then delete u_1 from X_1
    else EXTEND(v_0u_1, ℓ)

EXTEND$(v_0u_1v_1\cdots u_i, \ell)$:
let v_i be the partner of u_i in M
if $i = \ell$ then
    let $u_{\ell+1}$ be a neighbor of v_ℓ with $u_{\ell+1} \in X$
    if $u_{\ell+1}$ does not exist then delete u_ℓ from X_ℓ and return false
    output the augmenting path $v_0u_1v_1\cdots u_\ell v_\ell u_{\ell+1}$
    delete v_0 and $u_{\ell+1}$ from X, and $u_1, \ldots, u_\ell$ from all of $X_1, \ldots, X_\ell$, and return true
for each neighbor u_{i+1} of v_i with $u_{i+1} \in X_{i+1} \setminus \{u_i\}$ do
    if EXTEND$(v_0u_1v_1\cdots u_{i+1}, \ell)$ = true then return true
delete u_i from X_i and return false

Note that if it is not possible to extend the walk $v_0u_1v_1\cdots u_i$ to an augmenting walk of length

$2\ell+1$, then it is not possible to extend any other walk $v'_0u'_1v'_1\cdots u'_i$ of the same length with $u'_i = u_i$. This justifies why we may delete u_i from X_i in line 9 of EXTEND (and similarly why we may delete u_ℓ from X_ℓ in line 4 of EXTEND, and why we may delete u_1 from X_1 in line 4 of MAXIMAL-AUG-PATHS).

For the running time analysis, note that each vertex may be a candidate for u_i in only one call to EXTEND per i, because we delete u_i from X_i, either in line 9 if false is returned, or in line 6 if true is returned. Thus, the number of calls to EXTEND is at most $O(\ell|M|)$.

We use the given data structure $\mathcal{DS}_0$ to maintain X. This allows us to do line 3 of MAXIMAL-AUG-PATHS. Line 1 of MAXIMAL-AUG-PATHS requires $O(|M|)$ initial deletions from X. At the end, we reset X to V by performing $O(|M|)$ insertions of the deleted elements.

We also maintain $X_1, \ldots, X_\ell$ in ℓ new instances of the data structure $\mathcal{DS}_0$. This allows us to do lines 3 and 7 of EXTEND. Line 1 of MAXIMAL-AUG-PATHS requires $O(\ell|M|)$ initial insertions to $X_1, \ldots, X_\ell$. We conclude that MAXIMAL-AUG-PATHS takes $O(\ell|M| \cdot \tau_0)$ time. Hence, the overall running time of all $\ell = \Theta(1/\varepsilon)$ iterations is $O((1/\varepsilon^2)|M| \cdot \tau_0)$. ☐

5.2 Dynamic Geometric Bipartite MCM

To solve dynamic bipartite MCM for geometric intersection graphs, we again use a standard idea for dynamization: be lazy, and periodically recompute the solution only after every $\varepsilon\,\mathrm{OPT}$ updates (since the optimal size changes by at most 1 per update). The following theorem states our general framework:

Theorem 5.2. *Let $\mathcal{C}$ be a class of geometric objects, where there is a dynamic data structure $\mathcal{DS}_0$ for n objects in $\mathcal{C}$ that can find an object intersecting a query object (if exists), and supports insertions and deletions of objects, with $O(\tau_0(n))$ query and update time.*

Then there is a dynamic data structure for two sets of $O(n)$ objects in $\mathcal{C}$ that maintains a $(1+O(\varepsilon))$-approximation of the maximum-cardinality matching in the bipartite intersection graph, under insertions and deletions of objects, in $\widetilde{O}((1/\varepsilon^3)\tau_0(n))$ amortized time.

Proof. First, observe that we can maintain a maximal matching M_0 with $O(\tau_0(n))$ update time: We maintain the subset S of all vertices not in M_0 in a data structure $\mathcal{DS}_0$. When we insert a new object u, we match it with an object in S intersecting u (if exists) by querying $\mathcal{DS}_0$. When we delete an object u, we delete u and its partner v in M_0, and reinsert v.

Assume that $b \le \mathrm{OPT} < 2b$ for a given parameter b. Divide into phases with εb updates each. At the beginning of each phase, compute a $(1+O(\varepsilon))$-approximation of the maximum-cardinality matching by Lemma 5.1 in $\widetilde{O}((1/\varepsilon^2)b\cdot\tau_0(n))$ time. (It is important to note that we don't rebuild the data structure $\mathcal{DS}$ for $X = V$ at the beginning of each phase; we continue using the same structure $\mathcal{DS}$ in the next phase, after resetting the modified X back to V.)

Since the above is done only at the beginning of each phase, the amortized cost per update is $\widetilde{O}(\frac{(1/\varepsilon^2)b\cdot\tau_0(n)}{\varepsilon b})$.

During a phase, we handle an object insertion simply by doing nothing, and we handle an object deletion simply by removing its incident edge (if exists) from the current matching. This incurs an additive error at most $O(\varepsilon b) = O(\varepsilon\,\mathrm{OPT})$. We also perform the insertion/deletion of the object in $\mathcal{DS}_0$ for $X = V$, in $\widetilde{O}(\tau_0(n))$ time.

How do we obtain a correct guess b? We build the above data structure for each b that is a power of 2, and run the algorithm simultaneously for each b. ☐

5.3 Specific Results

Recall that for disks in $\mathbb{R}^2$ as well as boxes in $\mathbb{R}^d$, we have $\tau_0(n) = O(\log^{O(1)} n)$. (Note that most data structures for intersection detection can be modified to report a witness object intersecting the query object if the answer is true.) Thus, we immediately obtain:

Corollary 5.3. *There is a dynamic data structure for two sets of $O(n)$ disks in $\mathbb{R}^2$ that maintains a $(1+O(\varepsilon))$-approximation of the maximum-cardinality matching in the bipartite intersection graph, under insertions and deletions, in $O((1/\varepsilon^3)\log^{O(1)} n)$ amortized time.*

Corollary 5.4. *There is a dynamic data structure for two sets of $O(n)$ axis-aligned boxes in $\mathbb{R}^d$ for any constant d that maintains a $(1+O(\varepsilon))$-approximation of the maximum-cardinality matching in the bipartite intersection graph, under insertions and deletions, in $O((1/\varepsilon^3)\log^{O(1)} n)$ amortized time.*

6 General MCM

In this section, we adapt our results for bipartite MCM to general MCM. We use a simple idea by Lotker, Patt-Shamir, and Pettie [LPP15] to reduce the problem for general graphs (in the approximate setting) to finding maximal collections of short augmenting paths in *bipartite* graphs, which we already know how to solve. (See also Har-Peled and Yang's paper [HY22], which applied Lotker et al.'s idea to obtain static approximation algorithms for geometric intersection graphs.) The reduction has exponential dependence in the path length ℓ (which is fine since ℓ is small), and is originally randomized. We reinterpret their idea in terms of *color-coding* [AYZ95], which allows for efficient derandomization, and also simplifies the analysis (bypassing Chernoff-bound calculations). With this reinterpretation, it is easy to show that the idea carries over to the dynamic setting.

We begin with a lemma, which is a consequence of the standard color-coding technique:

Lemma 6.1. *For any n and ℓ, there exists a collection $\mathcal{Z}^{(n,\ell)}$ of $O(2^{O(\ell)}\log n)$ subsets of $[n] := \{1,\ldots,n\}$ such that for any two disjoint sets $A, B \subseteq [n]$ of total size at most ℓ, we have $A \subseteq Z$ and $B \subseteq [n] \setminus Z$ for some $Z \in \mathcal{Z}$. Furthermore, $\mathcal{Z}^{(n,\ell)}$ can be constructed in $O(2^{O(\ell)} n \log n)$ time.*

Proof. As shown by Alon, Yuster, and Zwick [AYZ95], there exists a collection $\mathcal{H}^{(n,\ell)}$ of $O(2^{O(\ell)}\log n)$ mappings $h : [n] \to [\ell]$, such that for any set $X \subseteq [n]$ of size at most ℓ, the elements in $\{h(v) : v \in X\}$ are all distinct. Furthermore, $\mathcal{H}^{(n,\ell)}$ can be constructed in $O(2^{O(\ell)} n \log n)$ time. (This is related to the notion of "ℓ-perfect hash family".)

For each $h \in \mathcal{H}^{(n,\ell)}$ and each subset $I \subseteq [\ell]$, add the subset $Z_{h,I} = \{v \in [n] : h(v) \in I\}$ to $\mathcal{Z}^{(n,\ell)}$. The number of subsets is $|\mathcal{H}^{(n,\ell)}| \cdot 2^\ell \le 2^{O(\ell)}\log n$. For any two disjoint sets $A, B \subseteq [n]$ of total size at most ℓ, let $h \in \mathcal{H}^{(n,\ell)}$ be such that the elements in $\{h(a) : a \in A\}$ and $\{h(b) : b \in B\}$ are all distinct, and let $I = \{h(a) : a \in A\}$; then $A \subseteq Z_{h,I}$ and $B \subseteq [n] \setminus Z_{h,I}$. □

We now present a non-bipartite analog of Lemma 5.1:

Lemma 6.2. *We are given an unweighted graph $G = (V, E)$, with $V = [n]$. Let $\mathcal{Z}^{(n,1/\varepsilon)}$ be as in Lemma 6.1. Suppose there is a data structure $\mathcal{DS}_0^*$ for storing a subset $X \subseteq V$ of vertices, initially with $X = \emptyset$, that supports the following two operations in τ_0 time: given a vertex $u \in V$ and $Z \in \mathcal{Z}^{(n,1/\varepsilon)}$, find a neighbor of u that is in $X \cap Z$ (if exists) and a neighbor of u that is in $X \setminus Z$ (if exists); and insert/delete a vertex to/from X.*

Given a data structure $\mathcal{DS}_0$ that currently has $X = V$, and given a maximal matching M_0, we can compute a $(1+O(\varepsilon))$-approximation to the maximum-cardinality matching in $\widetilde{O}(2^{O(1/\varepsilon)}\,\mathrm{OPT}\cdot\tau_0)$ time. Here, OPT *denotes the maximum matching size.*

Proof. As in the proof of Lemma 5.1, we iteratively maintain a current matching M, and it suffices to describe how to find a maximal collection Γ of vertex-disjoint augmenting paths of length $2\ell+1$ in the ℓ-th iteration, under the assumption that there are no augmenting paths of length $\le 2\ell-1$. However, the presence of odd-length cycles complicates the computation of Γ.

Initialize $\Gamma = \varnothing$. We loop through each $Z \in \mathcal{Z}^{(n,1/\varepsilon)}$ one by one and do the following. Let G_Z be the subgraph of G with edges $\{uv \in E : u \in Z, v \notin Z\}$. We find a maximal collection of vertex-disjoint augmenting paths of length $2\ell+1$ in G_Z that are vertex-disjoint from paths already selected to be in Γ; we then add this new collection to Γ. Since G_Z is bipartite, this step can be done using the MAXIMAL-AUG-PATHS procedure from the proof of Lemma 5.1. Since we are working with G_Z instead of G, when we find neighbors of a given vertex, they are now restricted to be in Z if the given vertex is in $[n] \setminus Z$, or vice versa; the data structure $\mathcal{DS}_0^*$ allows for such queries. The only other change is that when we initialize $X, X_1, \ldots, X_\ell$, we should remove vertices that have appeared in paths already selected to be in Γ.

Assume $2\ell+2 \le 1/\varepsilon$. We claim that after looping through all $Z \in \mathcal{Z}^{(n,1/\varepsilon)}$, the resulting collection Γ of vertex-disjoint length-$(2\ell+1)$ augmenting paths is maximal in G. To see this, consider any length-$(2\ell+1)$ augmenting path $v_0u_1v_1\cdots u_\ell v_\ell u_{\ell+1}$ in G. There exists $Z \in \mathcal{Z}^{(n,1/\varepsilon)}$ such that $u_1, \ldots, u_{\ell+1} \in Z$ and $v_0, \ldots, v_\ell \notin Z$. Thus, the path must intersect some path in Γ during the iteration when we consider Z. □

We can now obtain a non-bipartite analog of Theorem 5.2:

Theorem 6.3. *Let $\mathcal{C}$ be a class of geometric objects, where there is a dynamic data structure $\mathcal{DS}_0$ for n objects in $\mathcal{C}$ that can find an object intersecting a query object (if exists), and supports insertions and deletions of objects, with $O(\tau_0(n))$ query and update time.*

Then there is a dynamic data structure for $O(n)$ objects in $\mathcal{C}$ that maintains a $(1+O(\varepsilon))$-approximation of the maximum-cardinality matching in the intersection graph, under insertions and deletions of objects, in $\widetilde{O}(2^{O(1/\varepsilon)}\tau_0(n))$ amortized time.

Proof. This is similar to the proof of Theorem 5.2, with Lemma 6.2 replacing Lemma 5.1. The only difference is that to support the data structure $\mathcal{DS}_0^*$, we maintain $O(2^{O(1/\varepsilon)}\log n)$ parallel instances of the data structure $\mathcal{DS}_0$ for $X \cap Z$ and $X \setminus Z$, for every $Z \in \mathcal{Z}^{(n,1/\varepsilon)}$. This increases the update time by a factor of $O(2^{O(1/\varepsilon)}\log n)$.

We have assumed that the input objects are labeled by integers in $[n]$. When a new object is inserted, we can just assign it the next available label in $[n]$. When the number of objects exceeds n, we double n and rebuild the entire data structure from scratch. Similarly, when the number of objects is below $n/4$, we halve n and rebuild. □

Corollary 6.4. *There is a dynamic data structure for n disks in $\mathbb{R}^2$ that maintains a $(1+O(\varepsilon))$-approximation of the maximum-cardinality matching in the intersection graph, under insertions and deletions, in $O(2^{O(1/\varepsilon)}\log^{O(1)} n)$ amortized time.*

Corollary 6.5. *There is a dynamic data structure for n axis-aligned boxes in $\mathbb{R}^d$ for any constant d that maintains a $(1+O(\varepsilon))$-approximation of the maximum-cardinality matching in the intersection graph, under insertions and deletions, in $O(2^{O(1/\varepsilon)}\log^{O(1)} n)$ amortized time.*

7 Conclusion

In this paper, we have presented a plethora of results on efficient static and dynamic approximation algorithms for four fundamental geometric optimization problems, all obtained from two simple and general approaches (one for piercing and independent set, and the other for vertex cover and matching). We hope that our techniques will find many more applications in future work on these and other fundamental geometric optimization problems.

Many interesting open questions remain in this area. We list some below:

- Is there an $O(1)$-approximation polynomial-time algorithm for the piercing (MPS) problem for rectangles in $\mathbb{R}^2$ (like MIS for rectangles [Mit22])? Is there an $O(1)$-approximation polynomial-time algorithm for the weighted version of independent set (MIS) for rectangles in $\mathbb{R}^2$? Is there a sublogarithmic-approximation polynomial-time algorithm for (unweighted) MIS for boxes in $\mathbb{R}^3$? If the answer is yes to any of these questions, our approach could automatically convert such an algorithm into near-linear-time static algorithms and efficient dynamic algorithms.
- Is there an $O(n^\varepsilon)$-approximation algorithm for MIS for arbitrary line segments or polygons [FP11] with near linear running time? Our input rounding approach does not seem to work for arbitrary line segments or non-fat polygons.
- Is there a $(2-\varepsilon)$-approximation algorithm for vertex cover (MVC) for arbitrary line segments or strings [LPS+24] with near linear running time?
- Are there efficient dynamic algorithms for the weighted version of geometric MVC similar to our unweighted MVC results?
- Can we avoid the exponential dependence on ε in our results (in Section 6) on non-bipartite geometric maximum-cardinality matching (MCM)?

References

[ABR24] Amir Azarmehr, Soheil Behnezhad, and Mohammad Roghani. Fully dynamic matching: $(2 - \sqrt{2})$-approximation in polylog update time. In *Proceedings of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3040–3061, 2024. doi:10.1137/1.9781611977912.109. 8

[ACS+22] Pankaj K. Agarwal, Hsien-Chih Chang, Subhash Suri, Allen Xiao, and Jie Xue. Dynamic geometric set cover and hitting set. *ACM Trans. Algorithms*, 18(4):40:1–40:37, 2022. doi:10.1145/3551639. 3

[AE99] Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. In *Advances in Discrete and Computational Geometry*, pages 1–56. AMS Press, 1999. 20, 22

[AES10] Boris Aronov, Esther Ezra, and Micha Sharir. Small-size ε-nets for axis-parallel rectangles and boxes. *SIAM J. Comput.*, 39(7):3248–3282, 2010. doi:10.1137/090762968. 3, 10

[AG60] Edgar Asplund and Branko Grünbaum. On a coloring problem. *Mathematica Scandinavica*, 8(1):181–188, 1960. doi:10.7146/math.scand.a-10607. 15

[AHRS24] Pankaj K. Agarwal, Sariel Har-Peled, Rahul Raychaudhury, and Stavros Sintos. Fast approximation algorithms for piercing boxes by points. In *Proceedings of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4892–4908, 2024. doi:10.1137/1.9781611977912.174. 3, 4, 8, 9, 10, 11, 20

[AHV05] Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Geometric approximation via coresets. In *Combinatorial and Computational Geometry*, pages 1–30. Cambridge University Press, 2005. 9

[AHW19] Anna Adamaszek, Sariel Har-Peled, and Andreas Wiese. Approximation schemes for independent set and sparse subsets of polygons. *J. ACM*, 66(4):29:1–29:40, 2019. doi:10.1145/3326122. 5, 6

[AMN+98] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998. doi:10.1145/293347.293348. 13

[AP20] Pankaj K. Agarwal and Jiangwei Pan. Near-linear algorithms for geometric hitting sets and set covers. *Discret. Comput. Geom.*, 63(2):460–482, 2020. doi:10.1007/S00454-019-00099-6. 3

[AS00] Pankaj K. Agarwal and Micha Sharir. Arrangements and their applications. In *Handbook of Computational Geometry*, pages 49–119. North Holland / Elsevier, 2000. doi:10.1016/B978-044482537-7/50003-6. 13

[AvKS98] Pankaj K. Agarwal, Marc J. 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. 5, 9, 10, 16

[AYZ95] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337. 9, 26

[BCIK21] 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. 5, 7

[BCM23] Édouard Bonnet, Sergio Cabello, and Wolfgang Mulzer. Maximum matchings in geometric intersection graphs. *Discrete & Computational Geometry*, pages 1–30, 2023. doi:10.1007/S00454-023-00564-3. 8

[BDMR01] Piotr Berman, Bhaskar DasGupta, S. Muthukrishnan, and Suneeta Ramaswami. Improved approximation algorithms for rectangle tiling and packing. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 427–436, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365496. 5

[Beh23] Soheil Behnezhad. Dynamic algorithms for maximum matching size. In *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 129–162, 2023. doi:10.1137/1.9781611977554.CH6. 8

[BG95] Hervé Brönnimann and Michael T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discret. Comput. Geom.*, 14(4):463–479, 1995. doi:10.1007/BF02570718. 3

[BHI18] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. *SIAM J. Comput.*, 47(3):859–887, 2018. doi:10.1137/140998925. 7, 8

[BHR11] Reuven Bar-Yehuda, Danny Hermelin, and Dror Rawitz. Minimum vertex cover in rectangle graphs. *Computational Geometry*, 44(6-7):356–364, 2011. doi:10.1016/J.COMGEO.2011.03.002. 6, 7, 22, 39, 40

[BK19] Sayan Bhattacharya and Janardhan Kulkarni. Deterministically maintaining a $(2+\varepsilon)$-approximate minimum vertex cover in $O(1/\varepsilon^2)$ amortized update time. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1872–1885, 2019. doi:10.1137/1.9781611975482.113. 7

[BKS00] Boaz Ben-Moshe, Matthew J. Katz, and Michael Segal. Obnoxious facility location: Complete service with minimal harm. *International Journal of Computational Geometry & Applications*, 10(06):581–592, 2000. doi:10.1142/S0218195900000322. 3

[BKSW23] Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. Dynamic matching with better-than-2 approximation in polylogarithmic update time. In *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 100–128, 2023. doi:10.1137/1.9781611977554.CH5. 8

[BNTW24] Sujoy Bhore, Martin Nöllenburg, Csaba D. Tóth, and Jules Wulms. Fully dynamic maximum independent sets of disks in polylogarithmic update time. In *40th International Symposium on Computational Geometry, SoCG*, volume 293 of *LIPIcs*, pages 19:1–19:16, 2024. doi:10.4230/LIPICS.SOCG.2024.19. 3, 5, 6

[BS15] Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP), Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 167–179. Springer, 2015. doi:10.1007/978-3-662-47672-7_14. 8

[BS16] Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 692–711, 2016. doi:10.1137/1.9781611974331.CH50. 8

[CC09] Parinya Chalermsook and Julia Chuzhoy. Maximum independent set of rectangles. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 892–901. SIAM, 2009. doi:10.1137/1.9781611973068.97. 5, 6

[CCCK24] Sergio Cabello, Siu-Wing Cheng, Otfried Cheong, and Christian Knauer. Geometric matching and bottleneck problems. In *Proceedings of the 40th International Symposium on Computational Geometry (SoCG)*, volume 293 of *LIPIcs*, pages 31:1–31:15, 2024. doi:10.4230/LIPICS.SOCG.2024.31. 8

[CE16] Julia Chuzhoy and Alina Ene. On approximating maximum independent set of rectangles. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 820–829, 2016. doi:10.1109/FOCS.2016.92. 5, 6

[CH12] 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. 4

[CH20] Timothy M. Chan and Qizheng He. Faster approximation algorithms for geometric set cover. In *Proceedings of the 36th International Symposium on Computational Geometry (SoCG)*, volume 164 of *LIPIcs*, pages 27:1–27:14, 2020. doi:10.4230/LIPICS.SOCG.2020.27. 3

[CH21] Timothy M. Chan and Qizheng He. More dynamic data structures for geometric set cover with sublinear update time. In *Proceedings of the 37th International Symposium on Computational Geometry (SoCG)*, volume 189 of *LIPIcs*, pages 25:1–25:14, 2021. doi:10.4230/LIPICS.SOCG.2021.25. 3, 8, 9, 37

[CH24] Timothy M. Chan and Zhengcheng Huang. Dynamic geometric connectivity in the plane with constant query time. In *Proceedings of the 40th International Symposium on Computational Geometry (SoCG)*, volume 293 of *LIPIcs*, pages 36:1–36:13, 2024. doi:10.4230/LIPICS.SOCG.2024.36. 38

[Cha98] Timothy M. Chan. Approximate nearest neighbor queries revisited. *Discret. Comput. Geom.*, 20(3):359–373, 1998. doi:10.1007/PL00009390. 9, 12

[Cha03] 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. 3, 4, 12, 13, 16, 21, 38

[Cha04] Timothy M. Chan. A note on maximum independent sets in rectangle intersection graphs. *Inf. Process. Lett.*, 89(1):19–23, 2004. doi:10.1016/J.IPL.2003.09.019. 5

[Cha06] Timothy M. Chan. Dynamic subgraph connectivity with geometric applications. *SIAM J. Comput.*, 36(3):681–694, 2006. doi:10.1137/S009753970343912X. 22

[Cha10] 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. 5, 21

[Cha20a] Timothy M. Chan. Dynamic generalized closest pair: Revisiting Eppstein's technique. In *Proceedings of the 3rd SIAM Symposium on Simplicity in Algorithms (SOSA)*, pages 33–37, 2020. doi:10.1137/1.9781611976014.6. 5, 9, 20

[Cha20b] 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. 21

[CHQ20] Chandra Chekuri, Sariel Har-Peled, and Kent Quanrud. Fast LP-based approximations for geometric packing and covering problems. In *Proceedings of the 31st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1019–1038, 2020. doi:10.1137/1.9781611975994.62. 3, 9

[CHSX22] Timothy M. Chan, Qizheng He, Subhash Suri, and Jie Xue. Dynamic geometric set cover, revisited. In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3496–3528, 2022. doi:10.1137/1.9781611977073.139. 3

[CIK21] Jean Cardinal, John Iacono, and Grigorios Koumoutsos. Worst-case efficient dynamic geometric independent set. In *Proceedings of the 29th Annual European Symposium on Algorithms (ESA)*, volume 204 of *LIPIcs*, pages 25:1–25:15, 2021. doi:10.4230/LIPICS.ESA.2021.25. 5, 6, 20

[CKL+22] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *Proceedings of the 63rd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623, 2022. doi:10.1109/FOCS54457.2022.00064. 8, 17

[CLP11] Timothy M. Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proceedings of the 27th ACM Symposium on Computational Geometry (SoCG)*, pages 1–10, 2011. doi:10.1145/1998196.1998198. 40

[CM05] Timothy M. Chan and Abdullah-Al Mahmood. Approximating the piercing number for unit-height rectangles. In *Proceedings of the 17th Annual Canadian Conference on Computational Geometry (CCCG)*, pages 15–18, 2005. 3

[CMR23] Spencer Compton, Slobodan Mitrovic, and Ronitt Rubinfeld. New partitioning techniques and faster algorithms for approximate interval scheduling. In *Proceedings of the 50th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 261 of *LIPIcs*, pages 45:1–45:16, 2023. doi:https://doi.org/10.4230/LIPIcs.ICALP.2023.45. 7

[CPW24] Jana Cslovjecsek, Michał Pilipczuk, and Karol Wegrzycki. A polynomial-time $\mathrm{OPT}^{\varepsilon}$-approximation algorithm for maximum independent set of connected subgraphs in a planar graph. In *Proceedings of the 35th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 625–638, 2024. doi:10.1137/1.9781611977912.23. 5

[CW21] Parinya Chalermsook and Bartosz Walczak. Coloring and maximum weight independent set of rectangles. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 860–868, 2021. doi:10.1137/1.9781611976465.54. 6, 14, 15

[dBCvKO08] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008. URL: https://www.worldcat.org/oclc/227584184. 11

[DP14] Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1):1:1–1:23, 2014. doi:10.1145/2529989. 8, 23

[EIK01] Alon Efrat, Alon Itai, and Matthew J Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31:1–28, 2001. doi:10.1007/S00453-001-0016-8. 8, 9, 23

[EJS05] Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM Journal on Computing*, 34(6):1302–1323, 2005. doi:10.1137/S0097539702402676. 4, 6, 7

[EKNS00] Alon Efrat, Matthew J. Katz, Frank Nielsen, and Micha Sharir. Dynamic data structures for fat objects and their applications. *Computational Geometry*, 15(4):215–227, 2000. doi:10.1016/S0925-7721(99)00059-0. 3, 4, 13

[Epp95] David Eppstein. Dynamic Euclidean minimum spanning trees and extrema of binary functions. *Discret. Comput. Geom.*, 13:111–122, 1995. doi:10.1007/BF02574030. 9, 20

[Ezr10] Esther Ezra. A note about weak ε-nets for axis-parallel boxes in d-space. *Information Processing Letters*, 110(18-19):835–840, 2010. doi:10.1016/J.IPL.2010.06.005. 3, 10

[FM95] Tomás Feder and Rajeev Motwani. Clique partitions, graph compression and speeding-up algorithms. *J. Comput. Syst. Sci.*, 51(2):261–272, 1995. doi:10.1006/JCSS.1995.1065. 17

[FP11] Jacob Fox and János Pach. Computing the independence number of intersection graphs. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1161–1165, 2011. doi:10.1137/1.9781611973082.87. 5, 28

[Fre85] 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

[GKM+21] Waldo Gálvez, Arindam Khan, Mathieu Mari, Tobias Mömke, Madhusudhan Reddy Pittu, and Andreas Wiese. A $(2+\varepsilon)$-approximation algorithm for maximum independent set of rectangles. *CoRR*, abs/2106.00623, 2021. arXiv:2106.00623. 5, 14

[GKM+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 *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 894–905, 2022. doi:10.1137/1.9781611977073.38. 5, 14

[GP13] Manoj Gupta and Richard Peng. Fully dynamic $(1+\varepsilon)$-approximate matchings. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 548–557, 2013. doi:10.1109/FOCS.2013.65. 8, 20, 23

[Har23] Sariel Har-Peled. Approximately: Independence implies vertex cover. *CoRR*, abs/2308.00840, 2023. arXiv:2308.00840. 6

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. doi:10.1137/0202019. 8, 9, 23, 24

[HM85] Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32(1):130–136, 1985. doi:10.1145/2455.214106. 3, 4, 7

[HNW20] Monika Henzinger, Stefan Neumann, and Andreas Wiese. Dynamic approximate maximum independent set of intervals, hypercubes and hyperrectangles. In *Proceedings of the 36th International Symposium on Computational Geometry (SoCG)*, volume 164 of *LIPIcs*, pages 51:1–51:14, 2020. doi:10.4230/LIPICS.SOCG.2020.51. 3, 5, 6, 7

[HRS02] Hai Huang, Andréa W. Richa, and Michael Segal. Approximation algorithms for the mobile piercing set problem with applications to clustering in ad-hoc networks. In *Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 52–61, 2002. doi:10.1023/B:MONE.0000013626.53247.1C. 3

[HY22] Sariel Har-Peled and Everett Yang. Approximation algorithms for maximum matchings in geometric intersection graphs. In *Proceedings of the 38th International Symposium on Computational Geometry (SoCG)*, volume 224 of *LIPIcs*, pages 47:1–47:13, 2022. doi:10.4230/LIPICS.SOCG.2022.47. 3, 8, 9, 23, 26

[Ind07] Piotr Indyk. A near linear time constant factor approximation for Euclidean bichromatic matching (cost). In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 39–42. SIAM, 2007. 37

[KKK+22] Haim Kaplan, Alexander Kauer, Katharina Klost, Kristin Knorr, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Dynamic connectivity in disk graphs. In *Proceedings of the 38th International Symposium on Computational Geometry (SoCG)*, volume 224 of *LIPIcs*, pages 49:1–49:17, 2022. doi:10.4230/LIPICS.SOCG.2022.49. 8, 21

[KLPS86] Klara Kedem, Ron Livne, János Pach, and Micha Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discret. Comput. Geom.*, 1:59–70, 1986. doi:10.1007/BF02187683. 40

[KMP98] Sanjeev Khanna, S. Muthukrishnan, and Mike Paterson. On approximating rectangle tiling and packing. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 384–393, 1998. URL: http://dl.acm.org/citation.cfm?id=314613.314768. 5

[KMR+20] 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. 5

[KNS03] Matthew J. Katz, Frank Nielsen, and Michael Segal. Maintenance of a piercing set for intervals with applications. *Algorithmica*, 36(1):59–73, 2003. doi:10.1007/S00453-002-1006-1. 3

[LPP15] Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. Improved distributed approximate matching. *J. ACM*, 62(5):38:1–38:17, 2015. doi:10.1145/2786753. 9, 26

[LPS+24] Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, Jie Xue, and Meirav Zehavi. A 1.9999-approximation algorithm for vertex cover on string graphs. In *Proceedings of the 40th International Symposium on Computational Geometry (SoCG)*, volume 293 of *LIPIcs*, pages 72:1–72:11, 2024. doi:10.4230/LIPICS.SOCG.2024.72. 6, 28

[LT80] Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980. doi:10.1137/0209046. 39, 40

[Mar07] Dániel Marx. On the optimality of planar and geometric approximation schemes. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 338–348, 2007. doi:10.1109/FOCS.2007.26. 7

[Mit22] Joseph S. B. Mitchell. Approximating maximum independent set for rectangles in the plane. In *Proceedings of the 62nd Annual IEEE Symposium on Foundations of

Computer Science (FOCS), pages 339–350, 2022. doi:10.1109/FOCS52979.2021.00042. 5, 14, 28

[Nie00] Frank Nielsen. Fast stabbing of boxes in high dimensions. *Theor. Comput. Sci.*, 246(1-2):53–72, 2000. doi:10.1016/S0304-3975(98)00336-3. 5

[NTJ75] George L. Nemhauser and Leslie E. Trotter Jr. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8(1):232–248, 1975. doi:10.1007/BF01580444. 6, 9, 19

[OR10] Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC)*, pages 457–464, 2010. doi:10.1145/1806689.1806753. 7, 8

[Ove83] Mark H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes in Computer Science*. Springer, 1983. doi:10.1007/BFB0014927. 12

[PS85] Franco P. Preparata and Michael I. Shamos. *Computational Geometry: An Introduction*. Springer, 1985. 11, 22, 40

[PS16] David Peleg and Shay Solomon. Dynamic $(1+\varepsilon)$-approximate matchings: A density-sensitive approach. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 712–729, 2016. doi:10.1137/1.9781611974331.CH51. 8

[PT00] János Pach and Gábor Tardos. Cutting glass. *Discret. Comput. Geom.*, 24(2-3):481–496, 2000. doi:10.1007/S004540010050. 5

[Sol16] Shay Solomon. Fully dynamic maximal matching in constant update time. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 325–334, 2016. doi:10.1109/FOCS.2016.43. 8

[SW96] Micha Sharir and Emo Welzl. Rectilinear and polygonal p-piercing and p-center problems. In *Proceedings of the 12th Annual Symposium on Computational Geometry (SoCG)*, pages 122–132, 1996. doi:10.1145/237218.237255. 3

[SW98] Warren D. Smith and Nicholas C. Wormald. Geometric separator theorems & applications. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 232–243, 1998. doi:10.1109/SFCS.1998.743449. 38

[Vaz94] Vijay V. Vazirani. A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{V}E)$ general graph maximum matching algorithm. *Comb.*, 14(1):71–109, 1994. doi:10.1007/BF01305952. 8

[Zuc07] 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. 6

A MPS and MIS: Speedup via Sampling

In this appendix, we note that the $O(n^{1+O(\delta)})$ running time of our static algorithms for MPS and MIS in Sections 2–3 can be lowered to $O(n \operatorname{polylog} n)$ if we are only required to output an approximation to the optimal value. The idea is to approximate a sum by random sampling. This idea has been used before in dynamic geometric algorithms, e.g., in [CH21] on dynamic set cover. In the weighted case, we use *importance sampling* (which has also been used before in geometric optimization, e.g., in [Ind07]).

Fact A.1.

(Basic random sampling) *Let $a_1, \ldots, a_m \in [1, B]$. Select a random multiset $R \subseteq \{a_1, \ldots, a_m\}$ of size $\ell = \lceil 4B/\varepsilon^2 \rceil$, where each of its ℓ elements is chosen independently and is a_i with probability $1/m$. Then $X := \sum_{a_i \in R} a_i \cdot m/\ell$ is a $(1 \pm \varepsilon)$-approximation of $S := \sum_{i=1}^m a_i$ with probability at least $3/4$.*

(Importance sampling) *More generally: Let $a_i \in [w_i, Bw_i]$ for each $i = 1, \ldots, m$, with $\sum_{i=1}^m w_i = W$. Select a random multiset $R \subseteq \{a_1, \ldots, a_m\}$ of size $\ell = \lceil 4B/\varepsilon^2 \rceil$, where each of its ℓ elements is chosen independently and is a_i with probability w_i/W. Then $X := \sum_{a_i \in R} a_i \cdot W/(w_i \ell)$ is a $(1 \pm \varepsilon)$-approximation of $S := \sum_{i=1}^m a_i$ with probability at least $3/4$.*

Proof. We will prove the more general statement. First note that $\mathbb{E}[X] = \ell \sum_{i=1}^m (a_i \cdot W/(w_i \ell)) \cdot w_i/W = S$. Also, $\operatorname{Var}[X] \le \ell \sum_{i=1}^m (a_i \cdot W/(w_i \ell))^2 \cdot w_i/W = (1/\ell) \sum_{i=1}^m a_i^2 W/w_i \le (1/\ell) \sum_{i=1}^m B a_i W = BSW/\ell \le BS^2/\ell \le (\varepsilon S/2)^2$. By Chebyshev's inequality, $\Pr[|X - S| \ge \varepsilon S] \le 1/4$. □

Theorem A.2. *Let d be a constant. Given a set S of n axis-aligned boxes in $\mathbb{R}^d$, we can compute an $O(\log \log \mathrm{OPT})$-approximation to the size of the minimum piercing set for S in $\widetilde{O}(n)$ time w.h.p.*[14] *by a static Monte Carlo randomized algorithm.*

Proof. In the algorithm in the proof of Theorem 2.2, the size of the returned piercing set can be expressed as a sum of the sizes of piercing sets for a number of type-d subproblems handled by Lemma 2.1. The size of the piercing set for each type-d subproblem lies in the range from 1 to $B := O(b^d)$ (after removing empty subproblems). We can approximate the sum to within a constant factor by summing over a random sample of $O(B)$ terms (see Fact A.1), with error probability at most $1/4$ (which can be lowered by by repeating for logarithmically many trials and returning the median). Thus, the number of calls to Lemma 2.1 is $\widetilde{O}(b^d)$. The total running time is now $\widetilde{O}(n + b^{O(1)})$, which is $\widetilde{O}(n)$ by setting $b = n^\delta$ for a sufficiently small constant δ. □

Theorem A.3. *Let d and c be constants. Given a set S of n objects in $\mathbb{R}^d$ of constant description complexity from a c-fat collection $\mathcal{C}$, we can compute an $O(1)$-approximation to the size of the minimum piercing set for S in $\widetilde{O}(n)$ time w.h.p. by a static Monte Carlo randomized algorithm.*

Proof. This follows by a similar modification to the proof of Theorem 2.11 with random sampling. □

Theorem A.4. *Given a set S of n axis-aligned rectangles in $\mathbb{R}^2$, we can compute an $O(1)$-approximation to the size of the maximum independent set for S in $\widetilde{O}(n)$ time w.h.p. by a static Monte Carlo randomized algorithm.*

If the rectangles in S are weighted, we can do the same for an $O(\log \log n)$-approximation to the weight of the maximum-weight independent set.

[14] With high probability, i.e., with probability $1 - O(1/n^c)$ for an arbitrarily large constant c.

Proof. In the algorithm in the proof of Theorem 3.2 (after minor modification), the size/weight of returned independent set can be expressed as the maximum of the sizes/weights of $O((\log_b n)^d)$ independent sets, each of which is a sum of the sizes/weights of independent sets for a number of type-d subproblems handled by Lemma 3.1. In the unweighted case, the size of the independent set for each type-d subproblem lies in the range from 1 to $B := O(b^d)$ (after removing empty subproblems). In the weighted case, the weight of the independent set for the i-th type-d subproblem lies between w_i and Bw_i where w_i is the largest rectangle weight in the i-th subproblem. We can approximate the sum to within a constant factor by summing over $O(B)$ terms, via basic random sampling in the unweighted case or importance sampling in the weighted case (see Fact A.1), with error probability at most $1/4$ (which can be lowered by by repeating for logarithmically many trials and returning the median). Thus, the number of calls to Lemma 3.1 is $\widetilde{O}(b^d)$. The total running time is now $\widetilde{O}(n + b^{O(1)})$, which is $\widetilde{O}(n)$ by setting $b = n^\delta$ for a sufficiently small constant δ. □

Theorem A.5. *Let d and c be constants. Given a set S of n weighted objects in $\mathbb{R}^d$ of constant description complexity from a c-fat collection $\mathcal{C}$, we can compute an $O(1)$-approximation to the weight of the maximum-weight independent set for S in $\widetilde{O}(n)$ time w.h.p. by a static Monte Carlo randomized algorithm.*

Proof. This follows by a similar modification to the proof of Theorem 3.6 with importance sampling. □

B MVC: Fast Static Algorithms

B.1 Disks in $\mathbb{R}^2$

In this subsection, we give a static, near-linear-time, $(1 + O(\varepsilon))$-approximation algorithm $\mathcal{A}$ for MVC for disks under the promise that $n = O(\mathrm{OPT})$. As we are aiming for a $(1 + O(\varepsilon))$-factor approximation, we may tolerate an additive error of $O(\varepsilon n)$ because of the promise. MVC with $O(\varepsilon n)$ additive error reduces to MIS with $O(\varepsilon n)$ additive error, by complementing the solution.

PTASs are known for MIS for disks, but allowing $O(\varepsilon n)$ additive error, there are actually EPTASs that run in near linear time. For example, we can adapt an approach by Chan [Cha03] based on divide-and-conquer via separators. Specifically, we can use the following variant of Smith and Wormald's geometric separator theorem [SW98]:

Lemma B.1 (Smith and Wormald's separator)**.** *Given n fat objects in a constant dimension d, there is an axis-aligned hypercube B, such that the number of objects inside B and the number of objects outside B are both at most $(1 - \beta)n$ for some constant $\beta > 0$ (dependent on d), and the objects intersecting ∂B can be stabbed by $O(n^{1-1/d})$ points. Furthermore, B can be constructed in $O(n)$ time.*

Proof. (The following is a modification of a proof in [CH24], which is based on Smith and Wormald's original proof [SW98].)

For each object, first pick an arbitrary "center" point inside the object. Let B_0 be the smallest hypercube that contains at least $\frac{n}{2^d+1}$ center points. Let r be the side length of B_0. Let h be a parameter to be set later. For each $t \in \{\frac{1}{h}, \frac{2}{h}, \ldots, \frac{h-1}{h}\}$, let B_t be the hypercube with a scaled copy of B_0 with the same center and side length $(1 + t)r$. Since B_t can be covered by 2^d quadtree boxes of side length $< r$, the number of center points inside B_t is at most $\frac{2^d n}{2^d+1}$.

An object of diameter $\le r/h$ intersects ∂B_t for at most $O(1)$ choices of t. Thus, we can find a value t in $O(n)$ time such that ∂B_t intersects $O(n/h)$ objects of diameter $\le r/h$. On the other hand, the objects of diameter $> r/h$ intersecting ∂B_t can all be stabbed by $O(h^{d-1})$ points, because of fatness. Set $h = n^{1/d}$, and B_t satisfies the desired properties with $\beta = \frac{2^d}{2^d+1}$.

One remaining issue is how to find B_0 quickly. Let C be a sufficiently large constant. Form a $C \times \cdots \times C$ grid, so that the number of center points between any two consecutive parallel grid hyperplanes is n/C. This takes $O(Cn)$ time by a linear-time selection algorithm. Round each center point to its nearest grid point. The rounded point set is a multiset with $O(C^d)$ distinct elements. Redefine B_0 as the smallest hypercube that contains at most $\frac{n}{2^d+1}$ rounded center points (multiplicity included), which can now be computed in $C^{O(1)}$ time. For any box, the number of center points inside the box changes by at most $O(n/C)$ after rounding. So, we just need to increase β by $O(1/C)$. □

The MIS algorithm is simple: we just recursively compute an independent set for the disks inside B and an independent set for the disks outside B, and take the union of the two sets. If n is below a constant b, we solve the problem exactly by brute force in $2^{O(C)}$ time. (This is analogous to Lipton and Tarjan's original EPTAS for independent set for planar graphs [LT80].)

Since the optimal independent set can have at most $O(\sqrt{n})$ disks intersecting ∂B, the total additive error satisfies a recurrence of the form

$$E(n) \le \begin{cases} \max_{n_1,n_2 \le (1-\beta)n:\ n_1+n_2 \le n} (E(n_1) + E(n_2) + O(\sqrt{n})) & \text{if } n \ge b \\ 0 & \text{if } n < b. \end{cases}$$

This yields $E(n) = O(n/\sqrt{b})$. We set $b = 1/\varepsilon^2$. The running time of the resulting static algorithm $\mathcal{A}$ is $T(n) = \widetilde{O}(2^{O(1/\varepsilon^2)} n)$.

B.2 Rectangles in $\mathbb{R}^2$

In this subsection, we give a static, near-linear-time, $(\frac{3}{2} + O(\varepsilon))$-approximation algorithm for MVC for rectangles, by adapting Bar-Yehuda, Hermelin, and Rawitz's previous, polynomial-time, $(\frac{3}{2} + O(\varepsilon))$-approximation algorithm [BHR11].

Triangle-free case. We first start with the case when the intersection graph is triangle-free, or equivalently, when the maximum depth of the rectangles is at most 2. (The depth of a point q among a set of rectangles refers to the number of rectangles containing q; the maximum depth refers to the maximum over all $q \in \mathbb{R}^2$.)

We will design a static algorithm $\mathcal{A}$ for MVC for rectangles, under the promise that $n \le (2 + O(\varepsilon))\,\mathrm{OPT}$.

We say that a rectangle s *dominates* another rectangle s' if ∂s intersects $\partial s'$ four times, with s having larger height than s'. Bar-Yehuda, Hermelin, and Rawitz's algorithm proceeds as follows:

1. Decompose the input set S into 2 subsets S_1 and S_2, such that there are no dominating pairs within each subset S_i.

   The existence of such a decomposition follows easily from Dilworth's theorem, but for an explicit construction, we can just define S_1 to contain all rectangles in S that are not dominated by any other rectangle in S, and define S_2 to be $S \setminus S_1$. Correctness is easy to see (since the maximum depth is 2).

We can compute S_1 (and thus S_2) by performing $O(n)$ orthogonal range queries, after lifting each rectangle to a point in $\mathbb{R}^4$. In fact, computing S_1 corresponds to the *maxima* problem in $\mathbb{R}^4$, for which $O(n \log n)$-time algorithms are known [CLP11].

2. For each $i \in \{1, 2\}$, compute a vertex cover X_i of S_i which approximates the minimum with additive error at most εn.

   Since we tolerate εn additive error, it suffices to approximate the MIS of S_i with εn additive error.

   Because S_i has no dominating pairs, it forms a *pseudo-disk* arrangement (each pair's boundaries intersect at most twice). It is known [BHR11, KLPS86] that the intersection graph of any set of pseudo-disks that have maximum depth 2 and have no containment pair is in fact planar. (We can easily eliminate containment pair, by removing rectangles that contain another rectangle; and we can detect such rectangles again by orthogonal range queries.) So, we can use a known near-linear-time EPTAS for MIS for planar graphs; for example, Lipton and Tarjan's divide-and-conquer algorithm via planar-graph separators runs in $\widetilde{O}(2^{O(1/\varepsilon^2)}n)$ time [LT80].

3. Return X, the smaller of the two sets: $X_1 \cup S_2$ and $X_2 \cup S_1$.

Let X^* be the minimum vertex cover. Then

$$
\begin{aligned}
|X| &\le \tfrac{1}{2}(|X_1| + |S_2| + |X_2| + |S_1|) \\
&\le \tfrac{1}{2}(|X^* \cap S_1| + |X^* \cap S_2| + |S_1| + |S_2|) + \varepsilon n \;=\; \tfrac{1}{2}(|X^*| + n) + \varepsilon n \;\le\; (\tfrac{3}{2} + O(\varepsilon))|X^*|
\end{aligned}
$$

under the assumption that $n \le (2 + O(\varepsilon))|X^*|$. The running time is $T(n) = \widetilde{O}(2^{O(1/\varepsilon^2)}n)$.

At this point, if we apply Theorem 4.4 with $c = \frac{3}{2} + O(\varepsilon)$, $\gamma = \varepsilon$, and $\delta = \varepsilon^3$ (and $\tau_0(n) = O(\log^{O(1)} n)$ as noted in Section 4.4), we obtain:

Corollary B.2. *There is a dynamic data structure for n rectangles in $\mathbb{R}^2$ that maintains a $(\frac{3}{2} + O(\varepsilon))$-approximation of the minimum vertex cover of the intersection graph, under insertions and deletions, in $O(2^{O(1/\varepsilon^2)} \log^{O(1)} n)$ amortized time, under the assumption that the intersection graph is triangle-free.*

General case. We now design our final static algorithm $\mathcal{A}$ for MVC for rectangles that avoids the triangle-free assumption, again by building on Bar-Yehuda, Hermelin, and Rawitz's approach:

1. Remove vertex-disjoint triangles $T_1, \ldots, T_\ell$ so that the remaining set $S' := S \setminus (T_1 \cup \cdots \cup T_\ell)$ is triangle-free.

   We can implement this step greedily in polynomial time, but a faster approach is via a plane sweep. Namely, we modify the standard sweep-line algorithm to computing the maximum depth of n rectangles in $\mathbb{R}^2$ (similar to Klee's measure problem) [PS85]. The algorithm uses a data structure for a 1D problem: maintaining the maximum depth of intervals in $\mathbb{R}^1$, subject to insertions and deletions of intervals. Simple modification of standard search trees achieve $O(\log n)$ time per insertion and deletion. As we sweep a vertical line ℓ from left to right, we maintain the maximum depth of the y-intervals of the rectangles intersected by ℓ. When ℓ hits the left side of a rectangle, we insert an interval. When ℓ hits the right side of a rectangle, we delete an interval. When the maximum depth becomes 3, we remove the 3 intervals containing the maximum-depth point, which correspond to a triangle in the intersection graph, and continue the sweep. The total time is $O(n \log n)$.

2. Compute a vertex cover X' of S' which is a $(\frac{3}{2} + O(\varepsilon))$-approximation to the minimum. This can be done by Corollary B.2 in $\widetilde{O}(2^{O(1/\varepsilon^2)}n)$ time (we actually don't need the full power of a dynamic data structure, since we just want a static algorithm for this step).

3. Return $X = X' \cup T$ with $T := T_1 \cup \cdots \cup T_\ell$.

Let X^* be the minimum vertex cover. The key observation is that X^* must contain at least 2 of the 3 vertices in each triangle T_i, and so $|X^* \cap T| \geq \frac{2}{3}|T|$. Thus,

$$|X| \;=\; |X'| + |T| \;\leq\; (\tfrac{3}{2} + O(\varepsilon))|X^* \cap S'| + \tfrac{3}{2}|X^* \cap T| \;\leq\; (\tfrac{3}{2} + O(\varepsilon))|X^*|.$$

The running time of our final static algorithm $\mathcal{A}$ is $T(n) = \widetilde{O}(2^{O(1/\varepsilon^2)}n)$.